



RIO

PUC

Tese de Doutorado

Exploring Conversational Agents for Developer Guidance on Software Dependability Issues

João Lucas Marques Correia

Pontifícia Universidade Católica do Rio de Janeiro
Centro Técnico Científico
Departamento de Informática

Rio de Janeiro, September 30th, 2025



Pontifícia
Universidade
Católica do
Rio de Janeiro

Tese de Doutorado

Exploring Conversational Agents for Developer Guidance on Software Dependability Issues

João Lucas Marques Correia

Advisor: Prof. Alessandro Fabricio Garcia

Co-advisor: Prof. Rafael Maiani de Mello

Thesis presented to the Programa de Pós-graduação em
Informática, do Departamento de Informática of PUC-Rio in
partial fulfillment of the requirements for the degree of Doutor
em Ciências – Informática.

Rio de Janeiro, September 30th , 2025.



Pontifícia
Universidade
Católica do
Rio de Janeiro

Exploring Conversational Agents for Developer Guidance on Software Dependability Issues

João Lucas Marques Correia

Thesis presented to the Programa de Pós-graduação em
Informática of PUC-Rio in partial fulfillment of the
requirements for the degree of Doutor em Informática.
Approved by the Examination Committee:

Prof. Alessandro Fabricio Garcia

Orientador (a)

Departamento de Informática – PUC-Rio

Prof. Rafael Maiani de Mello

Co-Orientador (a)

Universidade Federal do Rio de Janeiro – UFRJ

Prof^a. Juliana Arriel

Departamento de Informática – PUC-Rio

Prof. Anderson Gonçalves Uchôa

Universidade Federal do Ceará – UFC

Prof. Balduino Fonseca dos Santos Neto

Universidade Federal de Alagoas – UFAL

Prof. Marco Aurélio Gerosa

Northern Arizona University – NAU

Rio de Janeiro, September 30th , 2025.



Pontifícia
Universidade
Católica do
Rio de Janeiro

Todos os direitos reservados. A reprodução, total ou parcial, do trabalho é proibida sem autorização da universidade, da autora e do orientador.

João Lucas Marques Correia

The author received his Bachelor's degree in Computer Science from the Federal University of Alagoas (UFAL), Brazil, in 2019. He earned his Master's degree in Computer Science from the Federal University of Alagoas (UFAL) in 2021. During his doctoral studies, he was a visiting researcher at Northern Arizona University (NAU), USA, in 2024. His main research interests include software dependability, software robustness, larger language models, machine learning and software development process.

Ficha Catalográfica

Correia, João Lucas Marques

Exploring Conversational Agents for Developer Guidance on Software Dependability Issues / João Lucas Marques Correia; advisor: Alessandro Fabricio Garcia; co-advisor: Rafael Maiani de Mello. – Rio de Janeiro: PUC-Rio, Departamento de Informática, 2025.

v., 276 f: il. color. ; 30 cm

Tese (doutorado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui bibliografia

1. Informática – Teses. 2. Engenharia de software – Teses. 3. Confiabilidade de Software; Assistência ao Desenvolvedor; Modelos de Linguagem de Grande Escala; Agentes Conversacionais. I. Garcia, Alessandro Fabricio. II. de Mello, Rafael. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. IV. Título.

CDD: 004

To my grandmother, who gave to her neighbors more than she could afford –
out of love, not abundance.

Acknowledgments

First and foremost, my deep gratitude to God, who has always granted me more than I deserve, for the strength and wisdom I asked for in so many difficult moments throughout this journey. I firmly believe that great battles are not won without facing great challenges, and these last four years were marked by deep learning, not only academic but also personal.

I thank my grandmother Maria de Lourdes, with whom I lived for most of my life. She, unfortunately, left this material world shortly before I completed my undergraduate degree. I am a materialized part of her hope in the education of her grandchildren. Wherever she is, I want her to know that everything turned out well.

To my brother Junior and my sister Lourdes Neta, my eternal gratitude for encouraging me and being with me since the beginning of my educational journey, back when I entered the Federal Institute. That was certainly the first step in changing the direction of my life. You were fundamental in this journey. I would also like to thank my mother, Ana Marques, who, even in the face of life's hardships, was always with me without sparing any effort whenever I needed it; and my sister Ana Paula, for the good conversations and the support she always gave me.

To my wife, Ewelín Costa — who, in other acknowledgments, has been girlfriend, fiancée, and now life partner — I thank you for walking beside me these past ten years. During this time, many of our dreams came true, and another cycle now ends with your constant and loving presence, as well as your unconditional support.

To my longtime friend, Aldo Silva, with whom I shared school benches and a roof during college years, my sincere gratitude. Thank you for believing in me, even when I didn't, and for your always encouraging advice. God has reserved a great purpose for you.

I am also grateful to my advisor, Alessandro Garcia, for the teachings and the opportunities he provided me throughout this journey. To my co-advisor, Rafael de Mello, I thank you for your contributions and help in moments of great value.

To Professor Baldoino Fonseca, my master's advisor, I deeply thank him for all the support, guidance, direction, and collaboration in projects, besides being the essential bridge that led me to meet Professor Alessandro and allowed this whole cycle to happen.

I also thank Professors Igor Steinmacher and Marco Gerosa, who welcomed me during my studies abroad. I want you to know that it was a rewarding

experience to be with you. I am extremely grateful for the opportunity to collaborate in research, for the trust you placed in me, and for the joint work. All of this vitally contributed to this Thesis, as well as to my personal and professional growth.

I would like to thank the friends who contributed so much to this work: Caio Barbosa, Daniel Coutinho, and Anderson Oliveira. And to all the friends I made at NAU: Ana Chaves, Genildo Gomes, Ítalo Santos, Jacob Penney, Kátia Felizardo, Maria Meireles, and Genildo Gomes.

I thank CAPES (88887.899310/2023-00) and CNPq (140770/2021-6) for the financial support, and to all who, by chance, were with me at some point along this journey.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Abstract

Correia, João Lucas Marques; Garcia, Alessandro Fabricio (Advisor); de Mello, Rafael (Co-Advisor). **Exploring Conversational Agents for Developer Guidance on Software Dependability Issues**. Rio de Janeiro, 2025. 276p. Tese de doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Software dependability is a critical non-functional requirement (NFR) that encompasses a system's ability to remain available, reliable, maintainable, and resilient in the presence of faults. Exception handling plays a central role in operationalizing these attributes, serving as the point where failures are detected, reported, and recovered. Despite its importance, exception handling is often under-supported by tools and under-emphasized in developer workflows. This thesis investigates how conversational agents, particularly those powered by Large Language Models (LLMs), can support dependable software development, using exception handling as a practical entry point. The thesis addresses three core research gaps: (i) the lack of empirical understanding of how exception-handling contributions are made and reviewed in collaborative settings; (ii) the limited evaluation of LLM-based assistants in real-world developer support scenarios; and (iii) the need to move beyond passive question answering toward agents capable of generating actionable and reviewable code changes. To this end, four complementary studies were conducted across open-source and industrial systems. First, exception-related pull requests (PRs) were characterized in Apache Java projects, revealing that these changes are proactive, multifaceted, and socially integrated. Second, DevMentorAI, a retrieval-augmented conversational agent, was evaluated in the Mozilla PDF.js project and shown to match or surpass human answers in many cases. Third, a comparison in Mozilla Firefox demonstrated that RAG-based models provide more helpful and comprehensive answers than generic LLMs and humans, albeit at the cost of verbosity. Finally, a multi-agent pipeline was introduced and validated in closed-source systems, showing that LLM-driven agents can detect and repair dependability issues in a way that is effective, explainable, and aligned with developer expectations. These results demonstrate that conversational agents, when grounded in project-specific context and combined with human-in-the-loop workflows, can meaningfully assist developers in identifying and resolving software dependability issues.

Keywords

Software Dependability; Developer Assistance; Large Language Models; Conversational Agents.

Resumo

Correia, João Lucas Marques; Garcia, Alessandro Fabricio; de Mello, Rafael. **Explorando Agentes Conversacionais para Orientar Desenvolvedores sobre Questões de Confiabilidade de Software**. Rio de Janeiro, 2025. 276p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

A confiabilidade de software é um requisito não funcional (RNF) crítico que abrange a capacidade de um sistema se manter disponível, confiável, manutenível e resiliente na presença de falhas. O tratamento de exceções desempenha um papel central na operacionalização desses atributos, sendo o ponto em que falhas são detectadas, relatadas e recuperadas. Apesar de sua importância, o tratamento de exceções é frequentemente negligenciado por ferramentas e pouco priorizado nas práticas dos desenvolvedores. Esta tese investiga como agentes conversacionais, especialmente aqueles baseados em grandes modelos de linguagem (LLMs), podem apoiar o desenvolvimento de software confiável, utilizando o tratamento de exceções como ponto de entrada prático. A pesquisa aborda três lacunas principais: (i) a falta de entendimento empírico sobre como contribuições relacionadas a exceções são feitas e revisadas em contextos colaborativos; (ii) a escassez de avaliações de agentes baseados em LLMs em cenários reais de suporte a desenvolvedores; e (iii) a necessidade de ir além de respostas passivas, explorando agentes capazes de gerar mudanças de código para revisão. Para isso, foram conduzidos quatro estudos complementares. Primeiramente, caracterizamos pull requests (PRs) relacionados a exceções em projetos Java da Apache, revelando que essas mudanças são proativas e multifacetadas. Em seguida, propomos o agente conversacional DevMentorAI, baseado em geração de recuperação aumentada (RAG), foi avaliado no projeto Mozilla PDF.js e demonstrou igualar ou superar respostas humanas para questões técnicas. Posteriormente, uma comparação no ecossistema do Mozilla Firefox mostrou que modelos RAG fornecem respostas mais úteis e completas do que LLMs genéricos e humanos, embora mais verbosas. Por fim, uma arquitetura multiagente foi proposta e validada em sistemas industriais, demonstrando que agentes baseados em LLMs podem detectar e corrigir problemas de confiabilidade de maneira eficaz e alinhada às expectativas dos desenvolvedores. Os resultados dão indícios que agentes conversacionais, quando fundamentados no contexto do projeto, podem apoiar significativamente os desenvolvedores na identificação e resolução de problemas de confiabilidade.

Palavras-chave

Confiabilidade de Software; Assistência ao Desenvolvedor; Modelos de Linguagem de Grande Escala; Agentes Conversacionais.

Table of contents

1	Introduction	17
1.1	Goal and Research Questions	21
1.2	Main Findings	24
1.3	Collaborations	26
1.4	Thesis Outline	26
2	Background and Related Work	29
2.1	Software Dependability	29
2.1.1	Exception Handling	29
2.2	Large Language Models	31
2.2.1	Retrieval Augmented Generation (RAG)	31
2.3	Related Work	32
2.3.1	Exception Handling in Software Systems	33
2.3.2	Large Language Models in Software Engineering	34
2.3.3	Retrieval-Augmented Generation for Project-Specific Support	34
2.3.4	LLM to Code Generation and Developer Support	35
3	On the Investigation of Exception Pull Request Characteristics: Exploring the Apache Ecosystem	37
3.1	Introduction	38
3.2	Related work	40
3.3	Study Design	42
3.3.1	Research Questions	42
3.3.2	Data Collection	43
3.3.2.1	Heuristic	44
3.3.2.2	Metrics	45
3.3.2.3	Manual Validation	45
3.3.3	Data Analysis	48
3.4	Results and Discussions	49
3.4.1	Heuristic Precision	49
3.4.2	Categorizing the Aims of Exception-PRs	50
3.4.3	<i>The Dynamics of Exception-PRs and Non-exception-PRs</i>	51
3.4.4	Developers in Exception-PRs and Non-exception-PRs	52
3.4.5	Exception Aspects in Contributions	54
3.4.5.1	Error representability	54
3.4.5.2	Error Recoverability	55
3.4.5.3	Error Detectability	56
3.4.5.4	Error Propagation	56
3.4.5.5	Other (non-exception-related code)	57
3.4.6	Multiple Exception Aspects Touched by Contributions	57
3.4.6.1	Error Recoverability and Error Representability	58
3.4.6.2	Error Detectability, Error Recoverability, and Error Representability	59
3.4.6.3	Error Propagation, Error Recoverability, Error Representability	59
3.4.7	Single versus Combined Exception Aspects	60

3.5	Threats to Validity	61
3.6	Conclusion	61
4	Unveiling the Potential of a Conversational Agent in Developer Support: Insights from Mozilla's PDF.js Project	63
4.1	Introduction	64
4.2	Related Work	66
4.3	DevMentorAI	67
4.3.1	Requirements	68
4.3.2	Architecture	68
4.3.2.1	Vector Storage Service	69
4.3.2.2	Generative Component (OpenAI)	69
4.3.2.3	History Service	70
4.3.2.4	Chain Operator	70
4.4	Methodology	70
4.4.1	Study Design	71
4.4.1.1	Project Selection	71
4.4.1.2	Ingestion of Project's Artifacts	71
4.4.1.3	Selecting the Questions	72
4.4.1.4	Expert Assessment	73
4.5	Results and Discussions	74
4.5.1	Overall Performance	74
4.5.2	Discussion About Individual Cases	74
4.5.2.1	DevMentorAI Outperforms Humans	75
4.5.2.2	DevMentorAI Performs as Good as a Human	77
4.5.2.3	DevMentorAI was Outperformed	79
4.6	Limitations and Design Trade-offs	81
4.7	Conclusion	83
5	A Comparison of Conversational Models and Humans in Answering Technical Questions: the Firefox Case	84
5.1	Introduction	84
5.2	Related Work	87
5.2.1	LLMs in Software Engineering	87
5.2.2	Supporting Newcomers to Software Projects	87
5.2.3	Improving LLM Performance with RAG	88
5.3	Study Design	90
5.3.1	Research Questions	90
5.3.2	Process for Data Collection	91
5.3.2.1	Step 1: Data Collection	91
5.3.2.2	Step 2: Question Filtering	93
5.3.2.3	Step 3: Collecting Answers from Models	94
5.3.2.4	Step 4: Expert Assessment of Answers	95
5.4	Data Analysis	97
5.5	Results and Discussions	99
5.5.1	Overall Performance	99
5.5.1.1	Helpfulness	100
5.5.1.2	Comprehensiveness	101
5.5.1.3	Conciseness	102

5.5.1.4	Practice	103
5.5.2	Statistical Analysis of Attributes	103
5.5.2.1	Helpfulness	104
5.5.2.2	Comprehensiveness	104
5.5.2.3	Conciseness	105
5.5.2.4	Findings and Discussions	105
5.5.3	Attributes Influencing Practical Preference	105
5.5.4	Evaluators' Perspectives	106
5.5.4.1	Human Answers	106
5.5.4.2	GPT Answers	108
5.5.4.3	RAG Answers	109
5.5.5	RAG for Developer Assistance	110
5.6	Threats to Validity	111
5.7	Conclusion	113
6	Developers' Perceptions of Automated Detection and Resolution of Software Dependability Issues	114
6.1	Introduction	116
6.2	Related Work	117
6.3	Study Design	119
6.3.1	Architecture for Automatic Detection and Resolution of Dependability Issues	119
6.3.1.1	Issue Detector	120
6.3.1.2	Issue Validator	121
6.3.1.3	Code Fixer	122
6.3.1.4	Code Judger	122
6.3.2	Research Questions	123
6.3.3	Data Collection	124
6.3.4	Manual Validation Process	126
6.3.5	Data Analysis	127
6.3.5.1	Quantitative Analysis	127
6.3.5.2	Qualitative Analysis.	128
6.4	Results and Discussions	129
6.4.1	Effectiveness in Resolving Dependability Issues	129
6.4.2	Quality of Generated Fixes	133
6.4.2.1	Agreement on Experts Evaluations	133
6.4.2.2	Disagreements on Experts Evaluations	134
6.5	Threats to Validity	136
6.5.1	Internal Validity	136
6.5.2	External Validity	136
6.5.3	Construct Validity	137
6.5.4	Conclusion Validity	137
6.6	Conclusion	137
7	Conclusions and Future Work	139
7.1	Motivation	139
7.2	Thesis Contributions	140
7.2.1	C1. Reframing Exception Handling as Routine (Chapter 3)	140

7.2.2	C2. Grounding LLMs in Project Artifacts Improves Answers (Chapters 4 and 5)	141
7.2.3	C3. Multi-Agent LLMs Produce Review-Ready Fixes for Real-World Dependability Issues (Chapter 6)	142
7.3	Synthesis and Lessons Learned	143
7.4	Future Work and Final Remarks	145
	Bibliography	147
A	On the Investigation of Exception Pull Request Characteristics: Exploring the Apache Ecosystem	165
A.1	Full List of Exception-PRs Identified by the Heuristic in the Druid.	165
A.2	Full List of Exception-PRs Identified by the Heuristic in the Pulsar.	171
A.3	Full List of Exception-PRs Identified by the Heuristic in the Flink.	182
A.3.1	Full List of Exception-PRs and Their Manual Validation	202
A.3.2	Extended Tables	209
B	Unveiling the Potential of a Conversational Agent in Developer Support: Insights from Mozilla's PDF.js Project	210
B.1	Full list of Questions, Answers and Experts' Assessment	210
C	A Comparison of Conversational Models and Humans in Answering Technical Questions: the Firefox Case	214
C.1	Example Form of the Assessment of Questions and Answers	214
C.2	Full List of Questions, Answers and Experts' Assessment	248
C.3	Overlapping List of Questions, Answers and Experts' Assessment	255
D	Developers' Perceptions of Automated Detection and Resolution of Software Dependability Issues	258
D.1	Full Agents' Prompts	258
D.2	Full List of Dependability Issues and Fixes	261
D.3	Full Colab Implementation for the Multi Agent Pipeline to Detect and Repair Dependability issues	264

List of figures

Figure 4.1	DevMentorAI's Pipeline.	68
Figure 4.2	Expert ratings for questions in a Likert scale.	74
Figure 5.1	Study design (The numbers in parentheses refer to the volume of data in each stage of the method).	92
Figure 5.2	Helpful answers	101
Figure 5.3	Comprehensive answers	101
Figure 5.4	Concise answers	101
Figure 5.5	Number of answers and the occurrence of attributes.	101
Figure 5.6	Correlation between answers attributes and their likelihood of being seen in practice.	106
Figure 6.1	Pipeline for automatic detection and resolution of dependability issues.	119
Figure 6.2	Data collection and validation.	124
Figure 6.3	Relevance Scores for all Issues.	131
Figure 6.4	Fixing Scores for all Issues.	131
Figure 6.5	Approval Scores for all Issues.	132
Figure 6.6	Familiarity with the Evaluated Code Scores for all Issues.	132
Figure 7.1	Flow of research questions, findings, and contributions leading to the thesis major goal of strengthening software dependability through developer support.	140

List of tables

Table 1.1	Publications as first author	27
Table 1.2	Publications as co-author	28
Table 3.1	Repositories and details about data collection.	45
Table 3.2	Pull requests' and developers' metrics description.	46
Table 3.3	Validator's demographics.	46
Table 3.4	Hypothesis test for PR's metrics	51
Table 3.5	Hypothesis test for developer's metrics	53
Table 3.6	Exception aspects and the aim of the contribution in exception-PRs.	54
Table 3.7	Combination of exception aspects and the aim of contribution in exception-PRs	57
Table 3.8	Aspects of exception-PRs appearing single or in combination.	60
Table 5.1	Illustrative form for the assessment of a question and its answers.	96
Table 5.2	Mozilla experts' demographics.	97
Table 5.3	Contingency tables for Fisher's test.	98
Table 5.4	Assigned Attributes and Most Preferred Answer per Attribute.	100
Table 5.5	Percentage of Answers Preferred to See in Practice.	100
Table 5.6	Fisher's test results per answer and attribute.	104
Table 6.1	Characterization of Target Systems by Type and Domain.	125
Table 6.2	Overview of Issue Distribution by System and Pipeline Stage.	125
Table 6.3	Developers Background	126
Table 6.4	Frequency of Detected Dependability Issues	129
Table 6.5	Mean Ratings and Likert Scale Interpretation	130
Table 6.6	Open Coding Categories Derived from Expert Feedback on Proposed Solutions	133
Table A.1	Exception aspects and the aim of the contribution in exception-PRs	209
Table A.2	Combination of exception aspects and the aim of contribution in exception-PRs	209

1

Introduction

Software dependability is a critical concern in modern software engineering. This non-functional requirement (NFR) encompasses the ability of a system to remain robust, maintainable, and resilient in the presence of failures (Laprie et al. 1992). The financial and operational consequences of insufficient dependability are significant. For example, the recent CrowdStrike outage caused an estimated total loss of \$5.4 billion across the 500 largest companies in the U.S. (excluding Microsoft), with the healthcare sector experiencing the most significant disruption, followed by banking and airlines (Parametrix Solutions, Inc. 2025).

Achieving dependability involves satisfying a set of interrelated attributes, each addressing a specific class of risks: *availability* ensures service continuity; *reliability* guarantees correct service over time; *safety* prevents faults from escalating into catastrophic consequences; *confidentiality* protects sensitive information; *integrity* maintains the accuracy and consistency of system states; and *maintainability* enables cost-effective evolution and repair (Laprie et al. 1992). The *IEEE Standard for Measures of the Software Aspects of Dependability* (IEEE Std. 928-2024) defines the following quantitative components of dependability: *availability* is the percentage of a system’s uptime relative to the total time (*i.e.*, uptime + downtime), representing its readiness for use; *reliability* is the ability of a system to operate without countable failures; *supportability* refers to the frequency, scope, and consistency of support actions required to establish, preserve, and revise the conditions and resources necessary for ongoing use; and *recoverability* denotes the ability of a system to mitigate the duration and losses caused by impairments. In this definition, some attributes are subclasses: for example, *security* failures are measured within *reliability*, and *maintainability* is encompassed by *supportability*.

These attributes are pursued through four complementary engineering strategies: *fault prevention*, *fault removal*, *fault forecasting*, and *fault tolerance* (Avizienis et al. 2004, Laprie et al. 1992). *Fault prevention* aims to avoid the introduction of faults during development by applying proper engineering practices, such as rigorous requirements analysis, formal specification, design reviews, coding standards, and the use of verified components. *Fault removal*

focuses on identifying and eliminating faults that were identified through: testing, static or dynamic analysis, inspections, or formal verification. *Fault forecasting* involves estimating the likelihood of future faults using techniques like statistical testing. *Fault tolerance* aims to ensure the system continues to operate despite the presence of faults, using error detection, exception handling, compensation, and recovery mechanisms to prevent faults from escalating into failures (Avizienis et al. 2004, Laprie et al. 1992).

Fault tolerance is the final measure to prevent a fault from escalating into a failure. It involves capturing abnormal states, managing errors, and facilitating recovery or controlled degradation. Modern programming languages provide exception handling mechanisms that are key to achieving fault tolerance. Effective exception handling enhances *reliability* by reducing the incidence of failures, supports *availability* through timely recovery, protects *safety* by guiding the system to fail-safe modes, maintains *integrity* via compensatory actions or rollback, and aids *maintainability* by localizing error handling and clarifying recovery policies. Despite its central role, robust exception handling remains a complex and error-prone task.

Decades of research have shown that designing exception-related code is inherently difficult (Petke et al. 2021, Laranjeiro et al. 2021, Shahrokni et al. 2013). As early as the 1980s, Black et al. (Black 1982) highlighted the semantic intricacies of exception mechanisms in object-oriented programming. More recently, Shah et al. (Shah et al. 2008) evaluated exception-related code in industrial systems and revealed that its quality is often substandard, despite its importance. They further argued that less experienced developers struggle to use these mechanisms effectively (Shah et al. 2010), leading them to propose the idea of a dedicated *exception engineer* to oversee exception design and maintenance.

In practice, when implemented correctly, exception-handling mechanisms are essential for ensuring system dependability. However, poor design choices can compromise these mechanisms, leading to hidden faults, the propagation of invalid states, or silent failures (Kechagia et al. 2014). In long-lived software systems, design problems often emerge gradually as the code evolves (Uchôa et al. 2020). Research has shown that exception handling problems are often linked to broader design and architectural flaws. Oliveira et al. (Oliveira et al. 2023) observed that poor exception-handling practices, such as using empty or generic catch blocks, frequently occur alongside architectural issues like *Concern Overload* and *Unwanted Dependency*. This indicates that insufficient exception-handling is not merely a localized implementation problem but is linked to more profound structural weaknesses within the system.

Several studies have investigated *robustness* of systems through exception-handling code. An empirical study of pull request (PR) discussions in the Spring ecosystem conducted by Oliveira et al. (Oliveira et al. 2024) reveals that *robustness* is referenced far less frequently than maintainability, security, and performance, remaining one of the most under-addressed NFRs in modern development (Viviani et al. 2018). It is rarely emphasized in the titles or descriptions of PRs, suggesting that it is not treated as a primary development concern. This result is consistent with earlier findings that developers often struggle to identify and prioritize exception-related or fault-tolerance concerns.

The challenges associated with maintaining exception handling mechanisms are particularly evident in the context of open-source software (OSS), where development is distributed and collaborative. In these environments, design decisions often emerge through asynchronous communication channels, such as issue trackers and pull requests (Dabbish et al. 2012). Although platforms like GitHub support collective decision-making, they also generate a large volume of discussions and frequent architectural changes (Tsay et al. 2014). These dynamics make it challenging to ensure system’s non-functional requirements, primarily because maintaining up-to-date design documentation is difficult and there is a high turnover of contributors (Guizani et al. 2025). Many contributors may lack a comprehensive understanding of the system’s design and architecture (Viviani et al. 2018, Viviani et al. 2021, Fronchetti et al. 2023). As a result, problems can gradually accumulate, undermining the system’s exception handling, fault tolerance mechanisms, and overall dependability. These issues highlight the necessity for automated tools that can help developers comprehend documentation, design, and architecture. Such tools can address technical questions about the system and ultimately assist in managing more complex problems, including those related to dependability.

Previous studies have explored issues related to exceptions (Cacho et al. 2014, Cacho et al. 2014, Zhong and Mei 2018, Barbosa et al. 2014), but they lack a systematic analysis of the collaborative processes involved with these mechanisms. In particular, they fail to address how developers propose, review, discuss, and maintain contributions related to exceptions in practice, focusing solely on problem classification and solutions. It limits our understanding of how exception-handling practices are socially constructed and technically validated in real-world collaborative development environments. Achieving a deeper understanding into these practices is crucial for designing tools that better align with developers’ actual needs and workflows.

While exception handling offers valuable insights into dependabil-

ity, the challenges it underlines are not unique. Modern software development—particularly in open-source ecosystems—requires developers to navigate incomplete documentation, evolving architectures, and high contributor turnover, all of which undermine system dependability. These challenges call for automated assistance that goes beyond static rules or generic recommendations, providing developers with timely, context-aware support that integrates seamlessly into their workflows.

Recent advances in Large Language Models (LLMs) and conversational agents present a promising avenue for delivering such support. These systems can assist developers in understanding complex code, answering technical questions, and even generating or reviewing code changes. However, their effectiveness hinges on how well they incorporate domain-specific context, how reliably they produce actionable and safe recommendations, and how their outputs are perceived and validated by human developers. To be truly effective, LLM-powered tools must bridge the gap between theoretical capabilities and the practical needs of dependable software development.

In this thesis we investigate how conversational assistants based in LLMs can support dependable software development by addressing three interconnected gaps. First, we explore how exception-handling practices manifest in collaborative development, yielding insights for tool design. Second, we evaluate project-aware LLM agents that assist with technical questions by grounding their responses in project artifacts. Third, we move beyond question-answering to examine whether these agents can generate meaningful, reviewable code improvements. To explore the extent to which agents can guide developers with software dependability issues, we propose a pipeline that automatically detects and delivers solutions to these problems. We investigate the range of detections of dependability issues in terms of their relation to the aforementioned dependability components (e.g., maintainability, reliability). We also evaluate the relevance of detected issues in the context of the project and its possible acceptance/merge by developers of the fix. We consider the perceived quality of detected dependability issues, their relevance, and the hypothetical acceptance/merging into the original code, based on a manual validation process conducted with expert developers in the evaluated systems. Grounded in real-world artifacts (e.g., practice in pull requests) and validated by expert developers, this work aims to inform advances in the design and experimentation of context-aware tools that can guide developers toward achieving system dependability in their practice.

1.1

Goal and Research Questions

In this section, we discuss three key gaps identified in this thesis: (G_1) the lack of empirical understanding of real-world exception handling contributions in pull request; (G_2) the limited evaluation of developer assistance techniques using LLMs in real-world scenarios; and (G_3) the need to shift from passive question-answering to agents that project-aware can generate actionable code changes.

G_1 Improve understanding of exception-handling practices in open-source software development. Existing research on exception handling has focused mainly on the mechanisms provided by languages to catch and handle exceptions, defect analysis, or best-practice adherence (Shah *et al.* 2008, Shah *et al.* 2010, Nakshatri *et al.* 2016, Sawadpong *et al.* 2012, Ebert *et al.* 2015). However, there is a lack of empirical insights into how exception handling mechanisms flow through modern collaborative development processes, such as PRs in distributed OSS ecosystems. The relationship between these discussions in design reviews (Viviani *et al.* 2018, Viviani *et al.* 2021) and the actual implementation of exception handling changes remains underexplored. Furthermore, the role of specialized contributors (i.e, the exception engineer) who repeatedly handle exception-related work, hypothesized in earlier literature, has not been validated in practice.

G_2 Evaluate approaches for project-aware developer assistance.

While onboarding barriers, knowledge distribution, and documentation are challenges in OSS (Käfer *et al.* 2018, Steinmacher *et al.* 2016, Steinmacher *et al.* 2015), existing support mechanisms such as mentoring and static documentation updates meet scalability and maintenance constraints. General-purpose LLMs have shown potential for aiding development tasks (Dakhel *et al.* 2023, Sallou *et al.* 2023, Balfroid *et al.* 2024), yet their effectiveness is inconsistent, often limited by lack of project-specific grounding, and prone to hallucinations or verbosity. Retrieval-Augmented Generation (RAG) offers a way to inject relevant, up-to-date context (Ibtasham *et al.* 2025, Finsås *et al.* 2024), but evaluations aimed to software engineering are occasional, and do not systematically compare against strong human or non-RAG baselines. Moreover, there is little guidance on when and how retrieval should be applied to balance coverage, accuracy, and conciseness in complex, evolving projects.

G_3 Explore the transition from information support to actionable code improvements.

While conversational agents with LLMs have shown promise in answering technical questions and even proposing code fixes, their deployment for support developers with dependability development remains under-explored. Most automated repair and multi-agent approaches focus on analyzing agents using generic bug datasets, often neglecting important challenges such as traceability, safety, and integration into professional workflows. Recent advancements in LLM-based judging (Zheng et al. 2023, Jiang et al. 2025) and dependability-oriented frameworks (Sobania et al. 2025, Wu et al. 2024, Zhang et al. 2025) demonstrate the feasibility of using LLMs for tasks like repair evaluation, code smell detection, or exception safety. However, these tools are rarely used to propose safe, reviewable patches in real-world closed-source settings autonomously. Moreover, threats like memorization of benchmark data (Ramos et al. 2024) raise concerns about generalizability and trust. There is a need to investigate whether multi-agent pipelines can proactively detect, validate, and repair dependability issues in a way that supports developer oversight and fosters integration into collaborative development practices.

These gaps highlight the necessity for strategies that effectively assist developers facing dependability challenges. By addressing these issues, we can enhance our understanding of how to maintain software dependability in large-scale collaborative environments. This will also inform the design of trustworthy, context-aware automated assistance that is rooted in real-world development practices. **The overarching goal of this Thesis is to Strengthen Software Dependability through Developer Support.**

To respond to these challenges, this thesis investigates how conversational agents can be effectively prepared and deployed to guide developers in improving software dependability. Our work addresses the three gaps through four complementary research directions: *(i)* characterizing how exception-related pull requests (exception-PRs) are handled in OSS; *(ii)* assessing the extent to which conversational agents can assist developers with technical questions; *(iii)* comparing the performance of general-purpose and retrieval-augmented conversational models with human responses in answering technical questions; and *(iv)* exploring how conversational agents can proactively enhance dependability through automated detection and remediation of code issues. These investigations span diverse contexts, including OSS ecosystems and a closed-source industrial system, enabling a comprehensive evaluation of these tools

across different development environments. To guide this investigation, we have formulated the following research questions.

- RQ₁* How do PR dynamics and contributor roles differ between exception-PRs and non-exception PRs in OSS?** This research question explores the social and procedural dynamics of contributions related to exceptions in large open-source ecosystems. Exception-related code is crucial for ensuring software reliability. However, the patterns of contribution, review practices, and levels of attention they receive are still underexplored in the context of open-source software (OSS). By systematically identifying exception-related pull requests (exception-PRs) and comparing them with non-exception PRs, we aim to characterize how these contributions emerge, how they are reviewed, and the roles that contributors assume throughout these processes. This analysis will provide a foundational understanding of where exception-related work is concentrated, how it is integrated into projects, and where opportunities exist for targeted, context-aware assistance.
- RQ_{2.1}* To what extent can a conversational assistant match human-provided answers to technical questions?** Building on the insights from *RQ₁*, which maps the landscape of exception-related contributions and their review dynamics, this question examines whether conversational assistants that incorporate project-specific context can help bridge knowledge gaps and reduce the demands of mentoring. The aim is to evaluate whether these assistants can provide responses that match the quality of answers given by humans, while also identifying the factors that most influence expert satisfaction, such as accuracy, contextual relevance, clarity, and actionability. This investigation will clarify when project-aware assistance is most beneficial, how it can complement human expertise, and how it can be integrated into the workflow to support both newcomers and experienced developers.
- RQ_{2.2}* How do RAG answers compare with a general-purpose LLM and with human answers across helpfulness, comprehensiveness, conciseness, and expert preference?** This research question expands on *RQ_{2.1}* by investigating whether retrieval-augmented generation (RAG) provides a significant advantage over general-purpose language models when offering project-specific support. We will compare responses from a RAG-based assistant, a general-purpose LLM, and human contributors. Our goal is to understand the trade-offs between contextual grounding and other factors such as verbosity, coverage, and perceived usefulness. Additionally, we aim to identify the specific areas in

which each source—RAG, GPT, and humans— excels when answering questions. Ultimately, we seek to provide actionable guidance on when retrieval is necessary, how much contextual information should be included, and what best practices can ensure scalable and maintainable AI-assisted support for large and evolving projects.

***RQ₃* How do developers perceive the ability of a multi-agent architecture to detect and fix software dependability issues?** This question builds on the progression from *RQ₁* to *RQ₂* and explores how to integrate conversational and reasoning capabilities into more complex, task-oriented development support. The focus shifts from merely answering technical questions to identifying dependability issues and suggesting specific code changes. We aim to investigate developers’ perceptions regarding the usefulness, trustworthiness, and barriers to adopting multi-agent architectures, especially in contexts such as closed-source projects or environments with strict resource and compliance constraints. This includes evaluating preferences for different levels of autonomy (e.g., advisory versus automated change proposals), the significance of traceable reasoning, and the practicality of integrating such agents into existing continuous integration and code review workflows. The overarching goal is to assess the potential of multi-agent systems in supporting dependable software development under real-world conditions.

1.2

Main Findings

***RQ₁* How do PR dynamics and contributor roles differ between exception-PRs and non-exception PRs in OSS?**

Statistical analyses showed no significant differences in metrics such as reviews, comments, and requests to reviewers between exception-PRs and non-exception-PRs, suggesting that exception-handling changes do not inherently demand more attention. Manual inspection revealed that most exception-PRs are proactive improvements (55.12%) rather than bug fixes (40.55%), with the most frequently addressed exception aspects being external error representability (42.52%) and effective error handling (36.22%), followed by error detectability, internal error representability, program continuity, and error propagation. Exception-PRs often addressed multiple aspects in combination, reflecting the interconnected nature of exception management. Findings challenge prior assumptions about the need for specialized “exception engineers,” highlight that many improvements focus on enhancing error messages rather than handler logic, and show cultural or procedural differences across projects

in how exception-related work is initiated. Overall, the study enriches understanding of how developers manage robustness-related code changes in practice, providing an empirical basis for future tools and processes to support dependable software development.

***RQ_{2.1}* To what extent can a conversational assistant match human-provided answers to technical questions?**

DevMentorAI, a Retrieval-Augmented Generation (RAG) conversational assistant integrated with the Mozilla PDF.js project, was able to match or surpass human-provided answers in the majority of evaluated technical Q&A cases, outperforming humans in 8 out of 14 questions, tying in 3, and being outperformed in only 3. Its main strengths lie in delivering more complete and contextually grounded explanations, often drawing on project documentation and source code to reveal existing features that some human respondents overlooked. In contrast, humans tended to perform better in situations requiring personal experience, nuanced debugging, or workarounds for undocumented behavior. The assistants' reliance on potentially outdated or incorrect documentation occasionally led to confidently stated but inaccurate answers. The study demonstrated that a well-integrated, project-specific RAG assistant could significantly support maintainers by providing consistent and accessible responses in open-source support channels, thereby reducing workload and improving newcomer onboarding.

***RQ_{2.2}* How do RAG answers compare with a general-purpose LLM and with human answers across helpfulness, comprehensiveness, conciseness, and expert preference?**

When applied to the Mozilla Firefox ecosystem, our enhanced version of the previous conversational agent, which is based on retrieval-augmented generation (RAG), consistently produced more comprehensive and contextually accurate answers than both a general-purpose LLM and human respondents. However, this came at the cost of conciseness. While Firefox Engineers assessed the RAG responses as the preferred option in a substantial number of cases, human responses remained competitive in terms of conciseness and occasionally in helpfulness, especially for narrow-scope questions that required minimal elaboration. General-purpose LLMs performed adequately for general or well-documented topics, but often overlooked project-specific nuances. This comparative evaluation highlights that RAG fit coverage and accuracy when grounded in rich, up-to-date project data, but may need optimization to balance verbosity with clarity. These findings refine the deployment considerations suggested in *RQ_{2.1}*, emphasizing when retrieval is essential and how it can be adjusted for maximum benefit.

***RQ*₃ How do developers perceive the ability of a multi-agent architecture to detect and fix software dependability issues?**

In an industrial, closed-source context, the proposed multi-agent architecture, comprising distinct LLM components for issue detection, validation, fixing, and judgment, was both relevant and effective in supporting the remediation of software dependability issues. Across three production systems, professional developers rated the pipeline’s outputs highly in terms of problem relevance (mean=4.47/5) and fix effectiveness (mean=4.35/5). The qualitative feedback highlighted that the pipeline particularly excelled at uncovering integrity, reliability, and maintainability concerns, offering changes with transparent rationale and adherence to project-specific exception-handling conventions. However, they stressed the necessity of human validation for complex or architecture-sensitive patches. These findings illustrate that the integration of retrieval-augmented and reasoning-based agents can extend beyond Q&A tasks (as explored in *RQ*_{2.1} and *RQ*_{2.2}) to deliver relevant and context-aware code improvements. Moreover, the study provides empirical grounding for the practical deployment of such agents within real-world development.

1.3 Collaborations

During my Ph.D. research, I actively collaborated with researchers from multiple institutions on studies that were directly or indirectly related to this thesis. In addition to working with colleagues from the Opus Research Group, part of the Software Engineering Laboratory (LES) at PUC-Rio, I collaborated with researchers from the Federal University of Rio de Janeiro (UFRJ), Northern Arizona University (NAU), North Carolina State University (NCSU), Federal University of Alagoas (UFAL) and Mozilla Corporation. To better present these scientific contributions, Tables 1.1 and 1.2 organize these publications. Table 1.1 lists publications where I am the first author and its status. In addition, Table 1.2 also lists papers in which I contributed as a co-author, reflecting collaborations that enriched this thesis with complementary perspectives and broader community engagement.

1.4 Thesis Outline

This chapter provides an overview. This thesis is structured as a compilation of four studies conducted during the course of this Ph.D. research. Each study addresses a specific research question presented in the introductory chapter. Chapters 3, 4, 5, and 6 detail the work reported in each study,

Table 1.1: Publications as first author

Year	Title	Authors	Status
2025	Developers' Perceptions of Automated Detection and Resolution of Software Dependability Issues	Correia, João ; et al.	In Progress
2025	A Comparison of Conversational Models and Humans in Answering Technical Questions: the Firefox Case	Correia, João ; Coutinho, Daniel; Castelluccio, Marco; Barbosa, Caio; Steinmacher, Igor; Gerosa, Marco; Garcia, Alessandro; de Mello, Rafael; Sarma, Anita	Accepted ICSE 2026
2024	On the Investigation of Exception Pull Request Characteristics: Exploring the Apache Ecosystem	Correia, João ; Coutinho, Daniel; Garcia, Alessandro; de Mello, Rafael; Barbosa, Caio; Oliveira, Anderson; Assunção, Wesley K. G.; Pereira, Juliana Alves; Steinmacher, Igor; Gerosa, Marco; Souza, Jairo; Arriel, Johny	Published SCAM 2024
2024	Unveiling the Potential of a Conversational Agent in Developer Support: Insights from Mozilla's PDF.js Project	Correia, João ; Nicholson, Morgan C.; Coutinho, Daniel; Barbosa, Caio; Castelluccio, Marco; Gerosa, Marco; Garcia, Alessandro; Steinmacher, Igor	Published AIware 2024

providing in-depth discussions, analyses, and results that progressively build toward the thesis contributions.

The chapters are organized as follows. **Chapter 2** presents the background and related work that supports this thesis, covering software dependability, exception handling, conversational agents, retrieval-augmented generation (RAG), and multi-agent systems.

Chapter 3 presents the study addressing RQ_1 , where we investigate the prevalence and characteristics of exception-related pull requests (PRs) in OSS, comparing them with non-exception PRs. This study analyzes how contributions emerge, how they are reviewed, and the roles assumed by contributors, providing a foundational understanding of exception-related development in large open-source ecosystems.

Chapter 4 addresses $RQ_{2.1}$, where we evaluate whether a project-aware conversational agent grounded in technical documentation can provide answers to technical questions that match human-provided responses. We also analyze which contextual and question-specific factors most influence expert satisfaction.

Chapter 5 addresses $RQ_{2.2}$, where we compare responses from a RAG-based assistant, a general-purpose LLM, and human contributors in a large open-source ecosystem. We assess the answers on helpfulness, comprehensiveness, conciseness, and overall expert preference, clarifying the trade-offs and

Table 1.2: Publications as co-author

Year	Title	Authors	Status
2025	PRemo: A Dataset of Emotions Found on Pull Request Discussions	Coutinho, Daniel; Pereira, J. A.; Neves, B.; Correia, João ; Barbosa, Caio; Assunção, Wesley K. G.; Steinmacher, Igor; Gerosa, Marco; Baffa, A.; Garcia, Alessandro	Published SBES 2025
2025	Case Study: Applying Fuzzing to REST APIs in a Large-Scale Industrial Setting	Fernandes, L.; Rocha, J.; Wiese, I.; Correia, João ; Barboza, E. A.; Torres, R.; Trindade, I.; Fonseca, Balduino; Ribeiro, M. M.	Published ISSRE 2025
2025	Evaluating the Noise Tolerance of Cloud NLP Services Across Amazon, Microsoft, and Google	Barbosa, Juliano; Fonseca, Balduino; Ribeiro, Márcio; Correia, João ; da Silva, Leandro Dias; GheyI, Rohit; Baia, Davy	Published Computers in Industry
2024	Understanding Developers' Discussions and Perceptions on Non-functional Requirements: The Case of the Spring Ecosystem	Oliveira, Anderson; Correia, João ; Assunção, Wesley K. G.; Pereira, Juliana Alves; de Mello, Rafael; Coutinho, Daniel; Barbosa, Caio; Libório, Paulo; Garcia, Alessandro	Published FSE 2024
2023	Don't Forget the Exception! Considering Robustness Changes to Identify Design Problems	Oliveira, Anderson; Correia, João ; Sousa, Leonardo; Assunção, Wesley K. G.; Coutinho, Daniel; Garcia, Alessandro; Oizumi, Willian; Barbosa, Caio; Uchôa, Anderson; Pereira, Juliana Alves	Published MSR 2023
2021	Look Ahead! Revealing Complete Composite Refactorings and their Smelliness Effects	Bibiano, Ana Carla; Assunção, Wesley K. G.; Coutinho, Daniel; Santos, Kleber; Soares, Vinicius; GheyI, Rohit; Garcia, Alessandro; Fonseca, Balduino; Ribeiro, Marcio; Oliveira, Daniel; Barbosa, Caio; Correia, João ; Oliveira, Anderson	Published ICSME 2021

best practices for deploying retrieval-based AI assistance.

Chapter 6 presents the study for RQ_3 , where we explore how a multi-agent architecture can detect exception-related issues and propose code changes in a closed-source environment. We evaluate expert acceptance rates, the effect on exception-handling quality, and the associated review effort.

Finally, **Chapter 7** concludes this thesis, summarizing its contributions, discussing implications for research and practice, and presenting directions for future work.

2

Background and Related Work

2.1

Software Dependability

Software dependability is the level of trust in a system's ability to consistently perform its intended functions, even under faults or unexpected operating conditions (Avizienis et al. 2004, Laprie et al. 1992). It includes several key attributes: *reliability*, which refers to the ability to deliver correct service consistently; *availability*, meaning the system is ready for use; *safety*, which entails preventing catastrophic consequences; *integrity*, ensuring that data and processes remain accurate and uncorrupted; and maintainability, reflecting the capacity for predictable and efficient repairs or enhancements (Avizienis et al. 2004, Laprie et al. 1992). Understanding the relationship between faults, errors, and failures is crucial. Latent defects are referred to as faults, erroneous internal states as errors, and externally observable deviations from correct service as failures (Laprie et al. 1992, Knight 2012). To engineer dependable software, it is important to adopt proactive strategies for fault prevention, implement mechanisms for the detection and management of faults, anticipate potential issues through forecasting, and incorporate techniques for fault tolerance to ensure continuous service (Lyu 2007).

2.1.1

Exception Handling

Exception handling is a fundamental aspect of software development that enables the management of internal errors and keeps the expected behavior of systems. Nowadays, languages such as Java and Python provide structured methodologies for exception handling. These techniques enable developers to address unexpected situations that may arise during code execution, documenting the points of failure within the call stack, appropriately releasing resources, and communicating error information across various components of an application.

Effective exception management significantly enhances the reliability of software systems. It mitigates the risk of errors propagating throughout a sys-

tem to the final user, and more importantly, reduces predictable application behavior. Maintaining the detail of effective exception management is the implementation of precise exception handlers. Such handlers should be sufficiently specific to address only those exceptions for which they are designed. The use of overly broad catch-all statements, such as `catch(Exception)` in Java or a simplified `except:` in Python, may obscure problems, thereby complicating troubleshooting efforts (Bloch 2018). It is critical to provide detailed diagnostic messages that convey context when logging information about the occurrence of exceptions. Such messages should encompass actionable information (i.e., operation identifiers and failed inputs) while ensuring the protection of sensitive data. This practice enables developers to determine the root cause of an issue without compromising user privacy.

Recoverability is another critical aspect of exception management. Whenever feasible, handlers should incorporate controlled retry mechanisms alongside intelligent strategies and establish time limits for operations. Design patterns such as circuit breakers or fallback responses contribute to the maintenance of application stability in the event of failures (Nygard 2018). Additionally, it is essential to ensure that operations are independent, allowing them to be safely repeated without causing adverse effects.

Clear communication is vital, particularly in differentiating internal exceptions from user-facing messages. Consistent mappings of internal exceptions to documented error responses are necessary to ensure that user-visible messages are clear and accurately reflect the information logged and monitored within the system. This level of transparency nurtures user trust and aids in the troubleshooting process by providing a coherent understanding of the state of the system (Avizienis et al. 2004).

Lastly, rigorous testing of recovery pathways is non-negotiable. All conceivable scenarios (i.e., including negative and edge cases) must be validated using automated testing methodologies and fault injection techniques. This comprehensive diligence ensures that exceptions are categorized, retries operate as intended, and the outcomes presented to users align with expectations. Such rigorous testing is vital in enhancing the overall reliability and resilience of software systems.

Adopting these practices, developers align with the objectives of ensuring software reliability, minimizing the frequency of incidents, expediting recovery processes, managing latency, and maintaining robust rollback and compensation mechanisms. Well-designed systems articulate clear exception flows, preserve the relationships among various errors, and convert exceptions into stable, predictable error messages (Lyu 2007).

2.2

Large Language Models

Large Language Models (LLMs) are a class of neural network architectures, typically based on the Transformer model (Vaswani et al. 2017), that are pre-trained on massive textual corpora and subsequently adapted to a variety of downstream tasks. Their training leverages self-supervised learning objectives, such as masked language modeling or next-token prediction, enabling them to acquire statistical representations of language without the need for extensive labeled data (Devlin et al. 2019, Brown et al. 2020). The scaling of such models in terms of parameters, data, and computation has led to the emergence of capabilities not observed in smaller models, including few-shot and zero-shot learning (Brown et al. 2020, Wei et al. 2022).

The Transformer architecture introduced by (Vaswani et al. 2017) employs multi-head self-attention and position-wise feed-forward layers to process sequences in parallel, offering improved scalability and efficiency compared to recurrent architectures. Pre-trained models such as BERT (Devlin et al. 2019), GPT-3 (Brown et al. 2020), and PaLM (Chowdhery et al. 2022) exemplify the diversity of design choices in pre-training objectives, context lengths, and parameter scales. Surveys, such as those by (Zhou et al. 2023) and (Bommasani *et al.* 2021), have highlighted the versatility of LLMs, documenting their applications in text generation, summarization, translation, code generation, and question answering.

Despite their impressive performance, LLMs present well-documented limitations. They may generate factually incorrect but fluent text, a phenomenon referred to as hallucination (Ji et al. 2023), and are prone to reproducing societal biases present in their training data (Bender et al. 2021). Moreover, their large-scale training imposes substantial environmental and computational costs (Strubell et al. 2019). Ongoing research is exploring methods to mitigate these issues, including instruction-tuning, reinforcement learning from human feedback, and RAG (Lewis et al. 2020).

2.2.1

Retrieval Augmented Generation (RAG)

Large Language Models (LLMs) capture linguistic patterns from extensive training; their knowledge is inherently limited to the data available at training time and may become outdated or insufficient. Retrieval-Augmented Generation (RAG) is an approach designed to address these limitations by combining parametric knowledge stored in the model’s weights with non-parametric knowledge retrieved from external sources at inference time (Lewis

et al. 2020). In this paradigm, a retriever component selects relevant documents from a knowledge base, and an LLM states its output on both the user query and the retrieved context.

A typical RAG architecture consists of two main stages: (i) a retrieval module that identifies relevant documents using similarity metrics in a vector space, and (ii) a generative model that incorporates the retrieved context into its prompt. This hybrid approach offers several advantages over relying solely on the LLM’s internal representations, including the ability to access up-to-date information and reduced hallucination rates (Lewis et al. 2020). Furthermore, retrieval-based conditioning allows models to dynamically adapt to new domains without full retraining, making RAG particularly attractive for tasks where knowledge evolves rapidly or is domain-specific.

In software engineering contexts, RAG can be used to enhance developer assistance tools by grounding LLM outputs in project-specific artifacts such as code repositories and issue trackers. This capability enables the model to generate recommendations that are consistent with the system’s actual architecture, coding standards, and operational history. By retrieving examples of past exceptions, such as corresponding fixes and relevant design guidelines, they can be generated.

Despite its benefits, RAG introduces challenges. The retrieval component’s effectiveness depends on the quality and relevance of the indexed knowledge base, as irrelevant or noisy documents may mislead the generation process. Additionally, retrieved evidence may contain conflicting or obsolete information, requiring mechanisms to assess reliability and resolve inconsistencies. From a dependability perspective, incorrect data could result in erroneous recommendations for critical routines. While RAG can significantly enhance the reliability and relevance of LLM outputs, its deployment demands validation of both the retrieval and generation stages.

2.3

Related Work

This section reviews the literature relevant to our four research questions, organized into three thematic areas: (i) exception handling in software systems, (ii) large language models (LLMs) in software engineering, and (iii) retrieval-augmented generation (RAG) for project-specific developer support.

2.3.1

Exception Handling in Software Systems

Exception handling plays a critical role in connecting internal errors to observable system behavior and ensuring robustness in software systems. Shah *et al.* (Shah *et al.* 2008) conducted an empirical study to characterize exception-handling practices in industrial projects. Through developer interviews, they found widespread dissatisfaction with language-imposed exception mechanisms (e.g., in Java) and limited organizational support for implementing exception-handling constructs. They proposed the role of an *exception engineer*, dedicated to designing, implementing, and maintaining exception-related code. In a subsequent study, Shah *et al.* (Shah *et al.* 2010) reported that novice developers often overlook exceptions, using them primarily for debugging, while experts integrate them into both normal and exceptional workflows to convey meaningful information to users.

Nakshatri *et al.* (Nakshatri *et al.* 2016) validated and extended these findings by comparing exception-handling practices in Java projects against best practices from *Effective Java* (Bloch 2018). They observed that developers frequently disregard checked exceptions and favor higher-level exception classes. Similarly, Sawadpong *et al.* (Sawadpong *et al.* 2012) showed that exception-handling code in Eclipse had a defect density three times higher than the overall code base, reinforcing the notion that such code is often neglected. Ebert *et al.* (Ebert *et al.* 2015, Ebert *et al.* 2020) further examined exception-handling bugs in Eclipse and Tomcat, classifying 220 bugs by frequency, severity, and fix difficulty. Their survey of 154 developers revealed that only 27% of organizations had formal practices for implementing exception-handling mechanisms.

Beyond defects, exception handling is closely linked to broader robustness concerns. Viviani *et al.* (Viviani *et al.* 2018, Viviani *et al.* 2021) manually classified pull request (PR) discussions in three open-source systems, identifying *maintainability*, *code*, and *robustness* as the most frequent design-related topics. These findings suggest that robustness—while often underrepresented in coding activities—remains a central concern in design discussions. Building on this gap, our first study RQ_1 systematically analyzes exception-related PRs in open-source ecosystems, comparing them with non-exception PRs to understand their social and procedural dynamics, contributor roles, and integration patterns.

2.3.2

Large Language Models in Software Engineering

The advent of LLMs such as Gemini (Gemini et al. 2023), BERT (Devlin et al. 2019), and GPT-3 (Brown et al. 2020) has introduced new possibilities for supporting software engineering (SE) activities. These models can generate human-like text, summarize content, and perform diverse natural language processing tasks. Recent research (Dakhel et al. 2023, Sallou et al. 2023, Wang et al. 2024, Pinto et al. 2023) has explored their applications and limitations in SE contexts. For example, Dakhel *et al.* (Dakhel et al. 2023) evaluated GitHub Copilot’s ability to provide correct, efficient, and reproducible programming solutions, finding mixed results compared to human-written code. Sallou *et al.* (Sallou et al. 2023) highlighted concerns around closed-source models, data leakage, and reproducibility. Balfroid *et al.* (Balfroid et al. 2024) investigated automated code tours for onboarding, noting issues with vagueness and redundancy in generated content.

Studies have also examined AI-assisted code completion and comprehension. Pudari *et al.* (Pudari and Ernst 2023) reported that Copilot often fails to follow best practices, while Sauvola *et al.* (Sauvola et al. 2024) emphasized the ethical and operational challenges of integrating generative AI into development workflows. Other works have explored the role of LLMs in supporting newcomers. Steinmacher *et al.* (Steinmacher et al. 2016) showed that structured, project-specific guidance reduces onboarding barriers, while more recent studies (Azanza et al. 2024, Balfroid et al. 2024, Richards et al. 2024, Adejumo et al. 2024) have tested LLM-driven assistants for onboarding, with varying success due to quality, context relevance, and hallucination issues.

Despite these advances, much prior work has focused on either general-purpose capabilities or small-scale evaluations. Our second study RQ_2 builds on this literature by designing a project-specific conversational assistant that integrates with different LLMs, incorporating source code and documentation via RAG to answer developer questions. We evaluate its responses against human-provided answers, focusing on quality dimensions critical to developer support.

2.3.3

Retrieval-Augmented Generation for Project-Specific Support

RAG enhances LLM capabilities by grounding responses in external, contextually relevant sources, such as project documentation and code (Xu et al. 2024, Ibtasham et al. 2025, Finsås et al. 2024). In SE contexts, Ibtasham *et al.* (Ibtasham et al. 2025) demonstrated that RAG can achieve 70%

relevance in release management tasks, while Finsaas and Maksim (Finsås et al. 2024) improved technical support accuracy through relevance feedback and multi-agent collaboration. However, applications to large-scale onboarding and developer Q&A remain underexplored.

Correia *et al.* (Correia et al. 2024) conducted the first systematic comparison of RAG-enhanced LLM responses and human answers for newcomer questions in an OSS project (PDF.js). Their in-house RAG system outperformed human responses in 8 of 14 cases, matched in 3, and underperformed in 3. However, their study was limited by its small scale, single evaluator, and narrow project scope.

Our third study RQ_3 substantially extends this line of research. We compare responses from (i) a general-purpose LLM, (ii) a RAG-enhanced LLM built on an open-source framework (Cognita), and (iii) human developers. Using 52 real-world questions from Firefox and evaluations from eight experienced engineers, we assess helpfulness, comprehensiveness, conciseness, and overall preference. This design enables us to isolate RAG’s contribution and provide actionable insights into when retrieval is necessary and how much context to include.

2.3.4

LLM to Code Generation and Developer Support

Our fourth study RQ_4 shifts focus from Q&A support to proactive dependability enhancement through a multi-agent architecture. LLMs have shown strong capabilities in code generation, bug fixing, and question answering. Initial research emphasized single-turn generation tasks using datasets like HumanEval (Chen et al. 2021) and MBPP (Google Research), which provided a controlled but limited view of model performance. As capabilities matured, attention shifted to more complex programming activities, including multi-function generation (Du et al. 2024), code summarization (Zhang et al. 2025), and multi-turn assistance (Wu et al. 2024).

Beyond generation, LLMs have been explored as interactive developer support agents. Tools such as GitHub Copilot and DevMentorAI leverage retrieval-augmented techniques to ground model responses in project-specific context. This shift toward project-aware support has enhanced model utility in practical settings—especially in environments where documentation is incomplete or contributors lack deep familiarity with system architecture. However, most studies emphasize Q&A scenarios, with limited focus on proactive interventions that improve software quality.

Recent advances push LLM use further into the software engineering

lifecycle. Wu *et al.* (Wu et al. 2024) introduced hybrid tools that combine static analyzers and LLMs for smell detection and refactoring, while Seeker (Zhang et al. 2025) adopts a multi-agent strategy to identify and repair weak exception paths. In the realm of software security, Zhou *et al.* (Zhou et al. 2024) and Zibaeirad *et al.* (Zibaeirad et al. 2024) have surveyed and evaluated LLM-based vulnerability detection and patching pipelines. These works illustrate a growing interest in deploying LLMs to support non-functional software qualities—but often in isolated tasks or benchmark settings.

Our work builds on this evolution by moving from passive Q&A assistance to proactive dependability improvement through a multi-agent architecture. While multi-agent frameworks have been employed in robotics and planning, their application to software engineering—particularly for dependability—remains underexplored. We introduce a pipeline that not only generates code, but also detects dependability issues, validates them via critique and voting agents, proposes patches, and uses an LLM-as-a-Judge to assess fixes. Uniquely, we embed this pipeline in real industrial systems and engage developers to evaluate trust, usefulness, and integration feasibility. This positions our approach as a step toward deployable, self-healing systems that go beyond suggestion to actionable, high-quality contributions.

On the Investigation of Exception Pull Request Characteristics: Exploring the Apache Ecosystem

A version of the work in this chapter appears in the **Proceedings of the 2024 IEEE International Conference on Source Code Analysis and Manipulation (SCAM)**

As mentioned in Chapter 1, fault tolerance is a critical characteristic for ensuring that software operates correctly under adverse conditions. Exception-handling mechanisms provide developers with a structured approach to managing unexpected situations, which enhances system dependability. However, implementing exception-handling code presents significant challenges, often resulting in complex maintenance issues and unforeseen faults. Exception-related pull requests (exception-PRs) offer valuable insights into how developers address these concerns over time.

To investigate exception-PRs in OSS, we conducted an empirical study on Java repositories within the Apache ecosystem. We systematically analyzed 988 exception-PRs using a heuristic-based classification, supplemented by a manual validation process with experienced developers. Our study revealed that exception-PRs exhibit no statistically significant differences in engagement metrics, such as the number of reviews and comments, compared to non-exception-PRs. Additionally, we observed that most exception-PRs focus on system improvements rather than bug fixes, suggesting that robustness-related changes are often proactive rather than reactive.

It is crucial to recognize that the term exception-related contributions, used throughout this chapter, closely linked to the overall dependability of software. Exception-PRs often affect various levels of software architecture, affecting how errors are represented, managed, and resolved during software recovery. Given their significant influence, these contributions are essential for ensuring the long-term dependability of software (e.g, robustness, reliability, and maintainability).

When dealing with exception-related challenges, developers must evaluate various software artifacts, such as error messages, handlers, and propagation mechanisms, to identify and resolve issues. This process can be particularly challenging in projects with limited documentation. Consequently, developers

primarily rely on source code and historical commit data to infer best practices and identify exception-handling improvements.

Specific patterns in exception-PRs can indicate recurring robustness challenges (see Section 3.4). While previous studies have explored exception-handling defects and best practices (Shah *et al.* 2008, Ebert *et al.* 2015), there is limited empirical evidence on how developers benefit from exception-PRs in practice. To bridge this gap, we conducted a mixed-methods study, including statistical analyses and qualitative evaluations of PRs discussions.

This chapter presents the findings of our study, which is documented in the paper *On the Investigation of Exception Pull Request Characteristics: Exploring the Apache Ecosystem* (Correia *et al.* 2024). **This study represents the first significant contribution of this Ph.D. thesis, addressing the question: *How do PR dynamics and contributor roles differ between exception-PRs and non-exception PRs in OSS?***

3.1 Introduction

Software robustness is of paramount importance to the success of a system. Robustness is described as the “degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions” (ISO/IEC/IEEE 24765:2017). Thus, the essential part of a program to achieve software robustness is the one that manages such erroneous inputs or conditions (Shah *et al.* 2008, Shah *et al.* 2010, Ebert *et al.* 2015, Garcia *et al.* 2001, Cacho *et al.* 2014). Certain programming languages offer a specific mechanism, the so-called *exception mechanism* or *exception handling mechanism* (Buhr *et al.* 2000, Robillard *et al.* 2003, Jakobus *et al.* 2015, Garcia *et al.* 2001), to deal with these erroneous conditions and structure this part of the program.

Java is an example of a programming language with a built-in exception-handling mechanism. Java enables developers to define `try`, `catch`, and `finally` blocks as well as raise or propagate exceptions with `throw` and `throws` statements (Asaduzzaman *et al.* 2016).¹ With these language features, developers can ensure the system’s recoverability (*i.e.*, catch or handle exceptions) from invalid or stressful conditions (Buhr *et al.* 2000, Chung *et al.* 2012). Developers can also represent these conditions as *exceptions*, as well as propagate them to upper layers. They can also implement the system’s cleanup actions through `finally` blocks (Hagen *et al.* 2000, Jakobus *et al.* 2015). The part of

¹<https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html>

the program that manages (*i.e.*, raise, protect, clean up, and handle) exceptions can be called the *exception-related code*.

Over the decades, various studies have investigated the inherent challenges in properly implementing exception-related code (Petke *et al.* 2021, Laranjeiro *et al.* 2021, Shahrokni *et al.* 2013). In the early 80s, Black *et al.* (Black 1982) warned developers about the complexity of exception-related code in object-oriented programs. In the late 2000s, Shah *et al.* (Shah *et al.* 2008) examined the quality of exception-related code in certain industrial software projects, concluding that, despite its importance to program robustness, its quality might be overlooked by developers. Further analysis of the same authors suggests that using exception mechanisms is hard for less experienced developers (Shah *et al.* 2010). Given these reasons, Shah *et al.* (Shah *et al.* 2008) even proposed the introduction of a new role of *exception engineer* to industrial software projects; this role would be dedicated to the design, implementation, and maintenance of the exception behavior of a software system.

In the context of open-source systems, one might question the nature of the contributions to the exception-related code and how they differ from other types of contributions. However, no empirical study has characterized and classified contributions to the exception-related code in open-source systems. Previous studies have focused on analyzing the frequency of changes to different sections of the exception-related code (Jakobus *et al.* 2015) and how these changes relate to bugs (Cacho *et al.* 2014, Cacho *et al.* 2014). While some studies (Ebert *et al.* 2015, Zhong and Mei 2018, Barbosa *et al.* 2014) have investigated the nature of exception-related bugs, other studies (*e.g.*, (Cabral and Marques 2011, Barbosa *et al.* 2012, Barbosa *et al.* 2012, Cornu *et al.* 2015, Barbosa and Garcia 2018)) focused on providing automatic repair for such bugs.

This paper reports a study we performed aimed at understanding the nature of contributions to the exception-related code, or *exception-related pull requests* (*exception-PRs*, for short), in Java projects of the Apache ecosystem. Our study also aimed to analyze the attention devoted by developers to exception-PRs as compared to other types of contributions in these projects. To achieve our aims, we implemented a heuristic to detect exception-PRs that, according to our validation, reached a precision of 90.7% (254) of the 280 manually evaluated PRs. Our study included several metrics, such as number of reviews, the number of requests for reviews, the number of comments. Our statistical analysis to compare the dynamics of exception-PRs with non-exception-PRs is inconclusive in determining differences in developers working on these two categories of PRs.

We also manually analyzed the information in these 254 exception-PRs to understand the types of contributions made by developers. The results of this analysis reveal the multifaceted aspects of these contributions. The principal aim for opening an exception-PR is improvement, accounting for 55.12% (140 out of 254) of the PRs. Bug fix is the second most frequent aim, representing 40.55% (103 out of 254) of the PRs. The most frequently modified aspect of exception-PRs involves the external error representation, appearing at least once in approximately 42.52% (108 out of 254) of the PRs. This means that the primary focus of these PRs is enhancing the clarity of error messages for users. Effective error handling is the second most frequent focus of the PRs, occurring in 36.22% (92 out of 254). Effective error handling concerns the proper implementation of exception handlers (catch blocks) to ensure the system's recoverability. Although addressed less frequently, contributions related to error detection and propagation were still notable, appearing in 23.62% (60 out of 254) and 16.54% (42 out of 254) of the PRs. Notably, when looking at these aspects of the exception-related code, external error representation often appears alone in about 18.50% of cases and roughly twice as frequently as part of combined modifications in 35.04% of PRs. Similarly, error detection and propagation are more often involved in combined contributions.

3.2 Related work

Shah *et al.* (Shah *et al.* 2008) conducted an empirical study to characterize exception-handling practices in industrial projects. The authors interviewed developers to understand their perspective on exception handling and their strategies for handling exception constructs. The results revealed developers' dissatisfaction with exception mechanisms in programming languages (such as Java) mainly due to the language imposition to implement such constructs. Also, developers claimed to have limited support in implementing exception-handling constructs. Based on these results, the authors propose the new role of *exception engineer*, who should focus on designing, implementing, and maintaining the exception-related code in software systems.

In a following study, Shah *et al.* (Shah *et al.* 2010) interviewed developers from the industry to characterize the problems and obstacles they face when designing and implementing exception handling. The results revealed a trend that novice developers usually ignore exceptions and frequently use them primarily to support debugging activities. In contrast, experts consider both regular and exception-related functionalities similarly important, using

exception-handling mechanisms to convey important information to users.

Nakshatri *et al.* (Nakshatri *et al.* 2016) conducted an empirical study to validate the findings from Shah *et al.* (Shah *et al.* 2008). The authors examined common practices for exception handling in Java projects, comparing them to the best practices outlined in the book *Effective Java* (Bloch 2018). Furthermore, they analyzed the comparison results alongside the evidence provided by Shah *et al.* (Shah *et al.* 2008). The results revealed that most developers often overlook checked exceptions and tend to use higher-level exception classes more frequently than lower-level ones.

Sawadpong *et al.* (Sawadpong *et al.* 2012) conducted an exploratory study investigating the defect density in exception handling and the overall source code. The authors explored the code of six releases in the Eclipse IDE and computed the defect ratios. Their results suggested that the exception defect density is roughly three times the overall code defect density. This observation reinforces previous works' findings that exception handling is usually neglected.

Ebert *et al.* (Ebert *et al.* 2015, Ebert *et al.* 2020) conducted an exploratory study on exception-handling bugs. They classified 220 bugs from Eclipse and Tomcat repositories in terms of frequency, severity, and difficulty in fixing them. Next, they surveyed 154 developers to understand organizational policies and perceptions about exception handling bugs. As a result, the authors proposed a categorization for exception-handling bugs and their causes. Also, they found in their analysis that exception-handling bugs are less ignored than other bugs. According to the survey, only 27% of the respondents claimed that their organization has practices for implementing exception-handling mechanisms.

Viviani *et al.* (Viviani *et al.* 2018, Viviani *et al.* 2021) investigated pull request discussions through manual classification of three open-source systems to find design-related topics. The results revealed that the three most frequent topics were *maintainability*, *code*, and *robustness*. Moreover, while the studies above suggest that exception-related code is often overlooked in coding activities, robustness remains a crucial concern of design discussions. These findings present an intriguing opportunity for further exploration, which could lead to a more comprehensive understanding of the challenges faced by implementing exception-related code in software systems.

Our present study differs from the existing literature in various ways. First, we examine the content of exception-related pull requests to understand the types of contributions to exception-related code in open-source systems. Second, we study how the dynamics of exception-related and non-exception-

related contributions differ. Third, we depict and classify the aspects of exception-related code contributions.

3.3 Study Design

This section describes the research questions and the methodology we employed to answer them. We gathered data from three Apache systems. Subsequently, we categorized the PRs into exception-PRs and non-exception-PRs and conducted quantitative and qualitative analyses, as described below.

3.3.1 Research Questions

We aim to analyze the characteristics of contributions concerning the exception-related code in open-source projects. To this end, we compared PRs related to exceptions (exception-PRs) with the other PRs (non-exception-PRs). Additionally, we examined the dynamics among developers within these contributions to identify the presence of exception experts. Finally, we also examined the exception aspects commonly addressed in exception-PRs. Our study is structured around the following research questions:

RQ₁: How does the dynamics of exception-PRs and non-exception-PRs differ?

To answer this research question, we compare exception-PRs metrics with non-exception-PRs ones. We hypothesize that exception-PRs are more complex than non-exception-PRs due to some characteristics. Firstly, exception-related code often addresses issues to ensure the resilience and stability of software systems under diverse operational conditions. Thus, a deeper understanding of the system's architecture and the potential fault lines that could compromise its integrity is required. Furthermore, exception-related code usually requires rigorous testing activities to cover a broader range of scenarios, including edge and uncommon cases. Concluding that exception-PRs are more complex may imply that they require more attention from team managers regarding resource allocation and process optimization.

In this *RQ*, we also aim to understand whether developers' behavior changes between exception-PRs and non-exception-PRs. For instance, we want to understand whether developers put more effort into exception-PRs than non-exception-PRs and identify developers whose team often relies on performing exception-related code changes. This analysis is grounded in results from previous work (Viviani *et al.* 2018) that reveal the experts' participation in discussions related to robustness. Also, Shah *et al.* (Shah *et al.* 2008) suggests

that developers working on the system’s exception-related code should have specialized knowledge.

RQ₂: What are the aspects addressed by exception-PRs? Given that robustness requirements are often retrofitted during development, and previous research has strongly emphasized bug-fixing of exception-related code, this research question is driven by the need to understand why developers submit exception-related code contributions. We seek to systematically classify the aims of exception-PRs, determining whether they introduce new functionalities, enhance existing ones, or address defects. Additionally, we aim to comprehend the aspects of exception-related code frequently addressed by such contributions, focusing on error representability, error recoverability, error detection, error propagation, and their subcategories. By answering this research question, we aim to enrich our understanding of how contributions manage exception-related code issues and whether existing tools designed to aid in writing exception-related code effectively support developers. To our knowledge, no previous research has undertaken efforts to understand the characteristics of exception-related code contributions in this manner.

3.3.2 Data Collection

We focused our studies on a set of projects that satisfied the following criteria:

Open-source systems. As aforementioned, the primary focus of our investigation lies on open-source systems, which also favors data availability and the future replication of this research. Open-source systems are typically maintained through version control systems, which provide unrestricted access to historical data. This access would be difficult to have in closed-source systems. We selected projects with repositories in GitHub since the platform has an application programming interface (API) for retrieving repository data, including pull requests, issues, comments, users, and commits. We rely on this data to answer the aforementioned RQs.

Java language. This study focuses on Java projects, as the language provides mechanisms that push for robustness guarantees (Jakobus *et al.* 2015, Cacho *et al.* 2014, Cacho *et al.* 2014). Java also has an exception-handling mechanism, allowing developers to determine exception-related conditions and explicitly handle them. This mechanism helps developers build resilient applications that recover from exception-related situations without crashing. As the language provides explicit exception-handling features, we can more objectively analyze how systems deal with robustness requirements. Java is a

statically typed language, meaning that type-checking is performed at compile-time. Many exceptions are checked for having at least a handler defined at compile time. This language characteristic may help catch various errors early in development, reducing the likelihood of runtime errors and enhancing the program's robustness.

Apache ecosystem. The Apache ecosystem comprises a collection of open-source systems developed and maintained by a diverse community. Due to factors such as the relevance of systems, the number of contributors, and the need for robust non-functional requirements, several studies have chosen the Apache ecosystem to empirically investigate reliability (Yan et al. 2023, Chen et al. 2019, Piantadosi et al. 2019). In this study, we selected three projects from this ecosystem considering the following criteria: (i) the repository should have more than 5,000 commits; (ii) the repository contains commits pushed a month before our data collection; (iii) the repository should have recent discussions on PRs and issues; and (iv) the repositories are from different domains.

Under the criteria above, we selected the following repositories: Apache Druid, Apache Pulsar, and Apache Flink. We systematically mined and collected data from these repositories, covering their entire history until July 2023. By choosing repositories from different domains, we aspire to foster heterogeneity in our study.

3.3.2.1 Heuristic

After defining the repositories, we designed a heuristic to filter contributions related to exceptions. This heuristic is aimed to be conservative regarding labeling contributions as exception-related in an attempt to enhance precision. Therefore, the heuristic comprised three conditions:

- C_1 The title must contain the keyword *exception* or *throw* or *catch* or its derivatives ²; and
- C_2 Body must contain the keyword *exception* or *throw* or *catch* or its derivatives; and
- C_3 The PR must not have been opened by a *bot*.

We employed the heuristic approach for each PR within the repository to determine its association with exception-related code. Initially, the heuristic analyzed the title and description of the PR to check its adherence to the criteria C_1 and C_2 . Subsequently, the criterion C_3 was checked by leveraging

²One of the considered words with the addition of a prefix or a suffix.

a feature provided by the GitHub API v3. Thus, the heuristic identified PRs initiated by bots and disregarded them in an attempt to remove noise from our dataset. Table 6.1 provides an overview of the collected data. The 1st column shows the repository name. The 2nd column shows the amount and percentage of exception-PRs in the repository. The 3rd column shows the total amount of PRs in the repository. Finally, the 4th column describes the domain of the system. As the data implies, we obtained 988 or 2.12% of the PRs classified as exception-PRs. In related work, Ebert *et al.* (Ebert *et al.* 2015) obtained a similarly low number when analyzing only exception-handling bugs from Bugzilla in the Eclipse and Tomcat repositories.

Table 3.1: Repositories and details about data collection.

Repository	exception-PRs	Total PRs	Domain
druid	126 (1.30%)	9675	Real-time analytics database
pulsar	294 (2.07%)	14198	Messaging/streaming platform
flink	568 (2.49%)	22728	Distributed processing engine

3.3.2.2 Metrics

To support the analyses to answer RQ_1 , we collected a set of metrics from pull requests and the developers. Table 3.2 presents these metrics and their descriptions. For each pull request, we computed the *number of reviews*, the *number of requests to review*, and the *number of comments*. For each developer, we calculated the *percentage of pull requests*, *percentage of reviews*, *percentage of requests to review*, and *percentage of comments*. Before computing these metrics, we split the PRs and developers according to their relation to exceptions using our heuristic. For instance, we calculated the developers' metrics for exception-PRs and non-exception-PRs, instead of computing their metrics in all PRs. In other words, the developer's metric *reviews ratio* has one value for exception-PRs and another for non-exception-PRs. In this way, we can perform statistical metrics analyses for both scenarios.

3.3.2.3 Manual Validation

We conducted a manual validation process with experienced validators, focusing on two main objectives. The first was to evaluate the precision of our heuristics in identifying real exception-PRs, thereby ensuring confidence in the analysis conducted for RQ_1 . The second, addressing RQ_2 , was to characterize the aim of exception-PRs and exception-related code aspects impacted by contributions.

Table 3.2: Pull requests' and developers' metrics description.

	Metric	Description
Pull request	Number of reviews	Number of reviews made by contributors on that pull request.
	Number of comments	Number of comments made by contributors on that pull request.
	Number of requests to review	Number of contributors marked as reviewers on that pull request.
Developer	Pull requests ratio	Percentage of pull requests opened by that user in the sample.
	Reviews ratio	Percentage of reviews made by that user in the sample.
	Requests to review ratio	Percentage of requests to review for that user in the sample.
	Comments ratio	Percentage of comments made by that user in the sample.

We initially calculated the sample size of exception-PRs needed for manual validation, ensuring it would represent our entire population of 988 with 95% confidence. The estimated sample size was 277 PRs. Therefore, we randomly selected 280 exception-PRs from the dataset, distributed as follows: 160 from Flink, 82 from Pulsar, and 38 from Druid.

We requested participants to collaborate in the validation process by sending an invitation message. The message contained the invitation and instructions for participating. Eight developers joined the manual validation process, each evaluating 70 PRs. Table 3.3 provides the demographics of these developers in terms of self-rated experience with software development, software exceptions, and pull request collaborations. Notice that we managed the validation process so that we had a double validation for each of the 280 PRs.

Table 3.3: Validator's demographics.

Validator	Experience with Software Development	Experience with Software Exceptions	Experience with Pull Requests
V1	Moderate	Moderate	Very high
V2	Very high	High	Very low
V3	High	Moderate	Moderate
V4	High	Low	High
V5	Very high	High	Very high
V6	Moderate	Moderate	Low
V7	High	Moderate	Moderate
V8	High	Very high	High

Below, we present the three questions that validators answered for each PR. We have the complete description of options in our supplementary material (Correia et al. 2024).

Q₁ What was the aim of the pull request? Single choice question, with the following alternatives:

- **New feature:** addition of a completely new feature to the system.
- **Improvement:** extension or refactoring of existing code structures.
- **Bug fix:** correction of a malfunction.
- **Test:** addition/change in code for test.
- **Other:** other non-covered aim.

Q₂ Is this pull request related to an exception? Single choice question, with the following alternatives:

- **Yes:** the pull request is related to exception.
- **No:** the pull request is not related to exception.

Q₃ Which exception-related aspect(s) was touched by the code contribution? Multiple choice question, with the following alternatives:

- **Error representability:** whether errors are properly represented/specified.
 - * **Internal:** whether errors are properly represented in the software.
 - * **External:** whether information about error detection, propagation, or handling is properly specified, traced, or logged for further consideration by the developer or user.
- **Error recoverability:** whether proper exception handling and clean-up actions are executed after an exception is raised, and the program's normal behavior returns to a consistent/safe state after those actions are executed.
 - * **Effective error handling:** whether the proper handlers and their actions are statically or dynamically attached to exceptions.
 - * **Program continuity:** whether the continuation of the program after error recovery is returned to the proper place in the program or the program abruptly stopped.
 - * **Cleanability:** whether the clean-up actions are properly chosen and activated in the program through the exception flow.
- **Error detectability:** whether errors are properly detected/thrown, *i.e.*, whether errors are raised and with the proper raising conditions.
- **Error propagation:** whether the error is properly propagated by either remapping to another error type or directly propagated to upper levels.

- **Other (non-exception-related code):** whether the error was solved without touching any of the aforementioned attributes.

As we performed a double validation process, disagreements were expected. To resolve disagreements on questions Q_1 and Q_2 , four authors collaboratively reviewed and reconciled differences. For Q_3 , we consolidated responses by merging the responses of both validators, since we understand that one code contribution could address complementary aspects perceived by different validators.

3.3.3 Data Analysis

This section describes the methods for analyzing the collected data and answering the RQs. We explain the steps performed in our data analysis as follows.

Step 1. As illustrated in Table 3.2, we gathered a set of metrics for each repository in this study. We then analyzed these metrics in two subsets: exception-PRs and non-exception-PRs. Both sets were created using the heuristic described in Section 3.3.2.1. We were expecting in our initial hypothesis that exception-PRs would demonstrate greater complexity than non-exception-PRs. By complexity, we refer to a high degree of interaction between developers and the code review process. Consequently, we conducted hypothesis tests to evaluate each PR metric (Table 3.2) to address RQ_1 . Below, we present the null and alternative hypotheses considered in our tests:

H_0 The PR's metric in exception-PRs is not greater than non-exception-PRs.

H_a The PR's metric in exception-PRs is greater than non-exception-PRs.

We conducted the *Mann-Whitney test* (Dodge 2008 (Mann-Whitney Test)) for each metric of PRs. In this scenario, the null hypothesis (H_0) suggests that the PR' metric in exception-PRs is not greater than that in non-exception-PRs. In contrast, the alternative hypothesis (H_a) indicates that the PR' metric in exception-PRs is greater than in non-exception-PRs.

In addition to the hypothesis test, we applied *Cliff's delta* analysis (Zaiontz 2024 (Cliff's Delta)) to obtain valuable insights into the magnitude of the differences observed between the PR metrics in exception-PRs and non-exception-PRs. While the Mann-Whitney test would establish a statistically significant difference between the two groups of metrics, *Cliff's delta* com-

plements this analysis by quantifying the effect size, thereby offering a more comprehensive understanding of the practical significance of the findings.

Step 2. In the previous step, we examined the complexity of exception-PRs and non-exception-PRs. Here, we aim to investigate whether there is a difference in developers' behavior when dealing with exception and non-exception PRs. As mentioned in Section 3.3.2.2, we computed the developer's metrics for both subsets of exception-PRs and non-exception-PRs. We rely on these metrics (Table 3.2) to observe the developers' behavior. To compare the resulting measures, we conducted a statistical test based on the following hypotheses:

- H_0 The developer's metric in exception-PRs is not greater than non-exception-PRs.
- H_a The developer's metric in exception-PRs is greater than in non-exception-PRs.

Since we expected to compare the mean values of variables in the same population, we employed the paired *Wilcoxon test* (Rey and Neuhäuser 2011) for statistical analysis. This test is well-suited for analyzing matched pairs of data. Additionally, we utilized *Cliff's Delta* to assess the magnitude of the observed results.

3.4 Results and Discussions

3.4.1 Heuristic Precision

As outlined in Section 3.3.2.3, a manual validation process was conducted to assess the heuristic precision in detecting exception-PRs. The validators agreed that 90.7% (254 out of 280) of the PRs analyzed were related to exception, in contrast to 9.3% (26 out of 280) that were not. These results confirm the heuristic's high precision in identifying exception-PRs, bolstering confidence in our analyses for RQ_1 .

We gathered further insights by analyzing cases where validators disagreed on their relationship to exceptions. The analysis provided valuable information on the limitations of the heuristic and helped us refine our understanding of what constitutes an exception-related contribution. Notice PR #17078³ from Flink, which involved a change in test procedures. Specifically, it threw the original exception rather than failing with a JUnit assertion if

³<https://github.com/apache/flink/pull/17078>

OpenSSL was not found. This change, impacting only test files and not system behavior, underscores the heuristic’s challenge in distinguishing between changes that affect system exception-related code and those that do not. Another example, PR #14207⁴ from Pulsar, involved unnecessary state transition handling in the `TransactionCoordinatorClientImpl.startAsync` method. The decision to remove the redundant exception also does not affect the system behavior, illustrating the nuanced nature of contributions that impact the system’s exception-related code. In summary, most of the non-exception-PRs classified by validators, cited exceptions. However, these contributions did not change the system’s functional behavior; therefore, the validators considered them unrelated to exception management.

3.4.2

Categorizing the Aims of Exception-PRs

During the manual validation, we requested participants to classify the developers’ aim for opening PRs. We obtained the following results by examining only the exception-PRs (254). The principal aim for opening an exception-PR was *improvement*, accounting for 55.12% (140) of the PRs. *Bug fix* was the second most frequent aim, representing 40.55% (103) of the PRs. *Test* was the aim of 3.15% (8) of the PRs, followed by *new feature* as the fourth most frequent aim, with 0.79% (2) of the cases. Finally, only 0.39% (1) PR was attributed to *other* miscellaneous reasons.

This result was interesting from the perspective of the existing literature on mining exception changes. Even though most of the previous studies focus on investigating the nature of exception bugs and their fixes (e.g., (Ebert *et al.* 2015, Ebert *et al.* 2020, Sawadpong *et al.* 2012, Barbosa *et al.* 2014)), such fixes were only the second most frequent aim in the analyzed PRs. In the projects analyzed, developers more frequently perform a wide range of exception-related improvements (than bug fixes), which we will discuss in more detail in Section 3.4.5. These improvements may vary from enhancing preventive or clean-up actions to altering error messages. This leads to our first finding.

Finding 1: While most empirical studies in the literature focus on characterizing and classifying exception bugs (e.g., (Ebert *et al.* 2015, Ebert *et al.* 2020, Sawadpong *et al.* 2012, Barbosa *et al.* 2014)), there is a limited understanding about the most recurring forms of exception-related improvements made in open source projects.

⁴<https://github.com/apache/pulsar/pull/14207>

Table 3.4: Hypothesis test for PR’s metrics

	Metric	p-value	Cliff’s Delta	Outcome
Druid	Number of reviews	0.412	negligible	Fail to reject H_0 .
	Number of requests to review	0.074	negligible	Fail to reject H_0 .
	Number of comments	0.033	negligible	H_0 rejected.
Flink	Number of reviews	0.999	negligible	Fail to reject H_0 .
	Number of requests to review	0.071	negligible	Fail to reject H_0 .
	Number of comments	0.810	negligible	Fail to reject H_0 .
Pulsar	Number of reviews	0.565	negligible	Fail to reject H_0 .
	Number of requests to review	0.354	negligible	Fail to reject H_0 .
	Number of comments	0.937	negligible	Fail to reject H_0 .

Another surprising result was the low occurrence of exception-PRs aimed at creating new features. Our rationale is that since Java forces developers to handle exceptions as soon as functional features are added, new features related to robustness (i.e., creating a new exception type) come up with regular features. Thus, since the PR is meant to introduce the new functional feature, the robustness aspect of the contribution can be left aside in the PR’s title and description. Then, when our heuristic looks for the exception-PRs, contributions aimed at new features are less frequent.

Finding 2: New features concerning the exception-related code are often introduced as a secondary or tertiary aim; they are often intertwined with the introduction of functional features.

3.4.3

The Dynamics of Exception-PRs and Non-exception-PRs

In this section, we aim to present the answers for RQ_1 . With our heuristic proven reliable for detecting exception-PRs, we selected two datasets of respective exception-PRs and non-exception-PRs. We investigated PR’s metrics and aimed to find significant differences, supporting the hypothesis that exception-PRs are more complex than non-exception-PRs. This complexity was quantified using metrics like the *number of reviews*, the *number of requests to review*, and the *number of comments* on PRs. We hypothesized that exception-PRs would involve more detailed discussions and feedback, reflected in higher values for these metrics. The results for the hypothesis tests are presented in Table 3.4.

The statistical analysis fails to reject the H_0 across the three repositories for most metrics, except for the *number of comments* for Druid. In this repository, the number of comments on exception-PRs showed a statistically significant difference compared to non-exception-PRs, with a *p-value* of 0.033,

indicating rejection of the H_0 . However, the effect size was negligible as measured by *Cliff's Delta*.

For the other two repositories, Flink and Pulsar, the hypothesis tests for all the metrics did not result in rejecting the H_0 . The *p-values* were substantially higher than the conventional alpha level of 0.05, which would be required to declare a statistically significant difference. The effect sizes were negligible, suggesting that any differences between the exception-PRs and non-exception-PRs for these metrics are of minimal practical significance. Such results imply that while exceptions are an essential aspect of software development, we cannot reject H_0 in favor of H_a , which suggests more complexity in exception-PRs. This leads to our third finding.

Finding 3: Contributions to exception-related code are so intertwined with the implementation of regular features that naturally lead to both receiving a similar amount of review comments and catching the attention of the same developers. This finding does not seem to back up the need for the role of “exception engineering” (Section II), as a previous study (Shah *et al.* 2008) suggested.

3.4.4

Developers in Exception-PRs and Non-exception-PRs

The earlier analysis indicates that the lack of difference is apparent between the composition of exception-PRs and non-exception-PRs when considering PR metrics. Looking further, we also explored developer metrics across these two categories of PRs. We still wanted to understand whether an “exception engineering” occurs in these projects. To this end, we calculated individual metrics for developers and applied a paired statistical hypothesis testing method. This approach helped identify significant differences in developers’ activities when working on exception-PRs versus non-exception-PRs. Table 3.5 presents the results for the hypothesis tests; in the following paragraphs, we discuss them in detail.

For both systems Druid and Pulsar, the H_0 that there is no difference in the developers’ *pull requests ratio* between exception-PRs and non-exception-PRs was rejected (p-values of 0.008 and 0.006, respectively). Initially, it suggests that developers in these projects are significantly more likely to open exception-PRs than non-exception-PRs. This result could indicate a specialization or higher focus on exception-related issues within these projects. However, for Flink, the *p-value* of 0.160 suggests no significant difference in the engagement level in exception-PRs compared to other non-exception-PRs.

Table 3.5: Hypothesis test for developer’s metrics

	Metric	p-value	Outcome
Druid	Pull requests ratio	0.008	H_0 rejected.
	Reviews ratio	0.846	Fail to reject H_0 .
	Requests to review ratio	0.965	Fail to reject H_0 .
	Comments ratio	0.353	Fail to reject H_0 .
Flink	Pull requests ratio	0.160	Fail to reject H_0 .
	Reviews ratio	0.990	Fail to reject H_0 .
	Requests to review ratio	0.430	Fail to reject H_0 .
	Comments ratio	0.990	Fail to reject H_0 .
Pulsar	Pull requests ratio	0.006	H_0 rejected.
	Reviews ratio	0.645	Fail to reject H_0 .
	Requests to review ratio	0.429	Fail to reject H_0 .
	Comments ratio	0.611	Fail to reject H_0 .

Across all three projects, the *p-values* for *reviews ratio*, *requests to review ratio*, and *comments ratio* were high, leading to a failure to reject the H_0 . This indicates that the number of reviews conducted, requests to review made, or comments posted between exception-PRs and non-exception-PRs is not different. This consistent lack of significant differences across these metrics suggests that despite there being developers that might be more likely to initiate exception-PRs in specific projects, once these PRs are created, they do not attract more reviews or comments than other PRs, reinforcing the observations in the Section 3.4.4.

The significant findings in the *pull requests ratio* for Druid and Pulsar, but not for Flink, could indicate varying cultural or procedural norms in these projects or differences in the complexity or impact of exception-related code issues identified in these contexts. It suggests that projects having developers with a higher propensity to submit exception-PRs might benefit from specialized training or resources to enhance the quality and effectiveness of these contributions, thereby improving the overall exception management of the software.

In summary, our analysis for RQ_1 reveals that we were unable to reject the H_0 in favor of the H_a , indicating no differences in interaction levels, such as *number of reviews*, *requests for reviews*, and *number of comments* between exception-PRs and non-exception-PRs, except a slight increase in comments within the Druid repository. Interestingly, despite this statistical significance, the *Cliff’s Delta* for this metric was negligible, suggesting that the practical impact of these differences is minimal. In contrast, developer metrics performed notable differences in the *pull request ratios* for Druid and Pulsar, suggesting a propensity for developers to initiate exception-PRs in these projects.

Finding 4: Although exception-PRs do not generally attract more engagement than non-exception-PRs, the results indicate that specific projects may culturally or procedurally highlight the importance of triggering exception-PRs.

3.4.5

Exception Aspects in Contributions

Table 3.6: Exception aspects and the aim of the contribution in exception-PRs.

Exception aspect occurrence			Pull Request Aim				
Aspect	%	#	Bug fix	Improvement	New feature	Test	Other
External error representability	42.52%	108	19.44%	77.77%	1.85%	0.92%	0.00%
Error recoverability - Effective error handling	36.22%	92	48.91%	46.73%	2.17%	1.08%	1.08%
Error detectability	23.62%	60	40.00%	56.67%	0.00%	3.33%	0.00%
Internal error representability	20.47%	52	36.53%	63.46%	0.00%	0.00%	0.00%
Error recoverability - Program continuity	20.47%	52	55.76%	42.30%	0.00%	1.92%	0.00%
Other (non-exception-related code)	19.69%	50	68.00%	26.00%	0.00%	6.00%	0.00%
Error propagation	16.54%	42	26.19%	69.05%	2.38%	2.38%	0.00%
Error recoverability - Cleanability	7.48%	19	36.84%	63.15%	0.00%	0.00%	0.00%

Previous studies focus only on characterizing exception bugs (Section 3.2). In the following sections, we present results related to RQ_2 , where we observed a much wider variety of exception contributions. This research question aims to determine which aspects exception-PRs frequently address. In this section, we present frequencies for the following exception aspects: *Error representability (internal or external)*, *Error recoverability (effective error handling, program continuity or cleanability)*, *Error detection*, and *Error propagation*. In the next section 3.4.6, we analyze the PRs that address multiple of these aspects combined as part of a single contribution.

During our manual validation, we categorized each exception-PR according to up to three exception aspects. After that validation, we calculated the prevalence of exception aspects across the 254 exception-PRs. Table 3.6 illustrates our results. The 1st column lists each exception aspect, 2nd and 3rd respectively present the percentage and number of PRs where that aspect shows up. The subsequent columns detail the aim of the PRs having that aspect.

3.4.5.1

Error representability

is a critical aspect of software robustness, ensuring that errors are accurately and effectively communicated both within the system (internal representation) and to its users (external representation), see Section 3.3.2.3. Ideally, each project should have its own guidelines for crafting internal and external error messages, though not all do, or they may rely on very generic

guidelines. Writing effective error messages that adhere to these guidelines can be challenging. Such representations must consider the semantic context of where the error was raised and propagated, as well as align with changes in the overall system.

External error representability: This exception aspect was the most frequently modified, accounting for 42.52% (108 PRs). The primary aim for these changes was overwhelmingly for program improvements, representing 77.77% of the cases. Bug fixes constituted 19.44%, while new feature implementations and tests were relatively minor. The high percentage of improvements suggests a significant focus on enhancing the clarity and usefulness of error messages for end-users. This leads us to another interesting finding.

Internal error representability: This aspect was the fourth most modified, accounting to 20.47% (52 PRs). Improvements were again the dominant aim, making up 63.46%, followed by bug fixes at 36.53%. There were no new features or tests associated with these changes. The focus on improvements here indicates efforts to refine how exceptions are represented internally within the system, which is important for better supporting program debugging and maintenance.

Finding 5: Most automated solutions in the literature (Barbosa and Garcia 2018, Barbosa and Garcia 2018, Barbosa *et al.* 2016, Rocha *et al.* 2013, Barbosa *et al.* 2012, Barbosa *et al.* 2012) focus on assisting developers in improving or repairing catch-block handlers. However, most improvements in exception-PRs aim to enhance the usability and usefulness of error messages.

3.4.5.2

Error Recoverability

Is crucial for maintaining system stability and reliability in the face of errors. It is usual for developers to postpone the implementation of comprehensive error-handling practices until the latter stages of development, prioritizing the integration of new features. Also, in the latter stages, the developer has more insights into error patterns. Effective *Error recoverability* involves error handling practices and considerations for program continuity and cleanability (see Section 3.3.2.3). Below, we discuss the frequency of each sub-aspect of recoverability.

Effective error handling: Overall, this was the second most frequently touched aspect, appearing in 36.22% (92 PRs). The distribution of aims was balanced, with 48.91% for bug fixes and 46.73% for improvements. New

features and tests were minimal. The significant proportion of bug fixes may indicate a reactive approach to addressing error-handling issues as they arise, while the substantial number of improvements points to ongoing efforts to enhance error recovery mechanisms.

Program Continuity: Touched by 20.47% (52 PRs), this attribute saw 55.76% of changes aimed at bug fixes and 42.30% at improvements. The high frequency of bug fixes highlights the need to ensure the system continue operating smoothly even when errors occur, minimizing user disruptions.

Cleanability: This attribute was the less touched, appearing in only 7.48% (19 PRs). The contributions to aspects aimed at improvements (63.15%) and bug fixes (36.84%). The focus on improvements suggests efforts to enhance the system's ability to recover from errors and clean up appropriately, preventing error accumulation and potential system degradation over time.

In summary, effective error handling, the second most addressed exception aspect, shows a notable focus on bug fixes, maybe indicating a reactive approach to resolving issues that disrupt the correct system functioning. Program continuity and cleanability, though less modified, reflect efforts to maintain uninterrupted system functionality and improve error recovery mechanisms. These findings underscore the system's proactive behavior in mitigating disruptions caused by errors and enhancing overall reliability.

3.4.5.3 Error Detectability

Ensures that errors are identified promptly and accurately; see the Section 3.3.2.3. This aspect was the third most modified accounting 23.62% (60 PRs). Improvements were the principal aim (56.67%), followed by bug fixes (40.00%) and tests (3.33%). The higher numbers of improvements may indicate an effort to enhance the system's ability to catch errors early since issues with *Error detectability* may result in failures raising to end-user. Conversely, the percentage of bug fixes may suggest challenges in managing error detection in early development phases.

3.4.5.4 Error Propagation

Concerns the management of error propagation throughout the system (refer to Section 3.3.2.3. This aspect received less attention, accounting for 16.54% (42 PRs). The changes aimed at improvements constituted 69.05%, with bug fixes at 26.19%, and both new features and tests at 2.38%. The significant focus on improvements may suggest a delayed attention to error prop-

agation. One rationale is that errors are initially only handled locally (where they occur), and as the system evolves, they are propagated and handled in the proper layers. Since error propagation is primarily an architectural concern, the low percentage of bug fixes may be due to this aspect not directly impacting the end-user experience as significantly as other issues do.

3.4.5.5

Other (non-exception-related code)

Not all exception-related contributions necessarily modify the exception-related aspects of the code. For example, a `FileNotFoundException` might be thrown during a file read operation, and the fix could involve inserting an if block to check for the file's existence before attempting to read it. For this and other scenarios, we allowed validators to classify some exception-PRs touching other aspects. *Other* was the sixth most modified aspect, accounting for 19.69% (50 PRs) of the cases. Unlike the other aspects discussed, *Other* was related to bug fixes 68.00% of the time, improvements 26.00% of the time, and tests 6.00% of the time.

Our analysis reveals a stronger emphasis on improvements across all exception aspects, indicating a proactive approach to enhancing the system's exceptions. Bug fixes also constitute significant changes, underscoring the ongoing need to address existing issues. The relatively low focus on new features may also suggest that, for Java, exception management is more about refining and solidifying existing capabilities rather than introducing new ones. While present, tests are less frequent since validators classified most PRs as only handling test files unrelated to exceptions.

3.4.6

Multiple Exception Aspects Touched by Contributions

Table 3.7: Combination of exception aspects and the aim of contribution in exception-PRs

Exception aspects occurrence (combinations)			Pull Request Aim				
Aspect	%	#	Bug fix	Improvement	New feature	Test	Other
Error recoverability, Error representability	12.99%	33	39.39%	57.58%	3.03%	0.00%	0.00%
Error detectability, Error recoverability, Error representability	7.09%	18	44.44%	55.56%	0.00%	0.00%	0.00%
Error propagation, Error recoverability, Error representability	5.51%	14	42.86%	50.00%	7.14%	0.00%	0.00%

In the previous section, we examined the frequency exception aspects disregarding combinations. In this section, we focus on analyzing the combinations of exception aspects. By exploring how these elements co-occur within exception-PRs, we aim to gain deeper insights into the integrated strategies employed to enhance system exceptions. The Table 3.7 illustrates the top three

most frequent combinations in exception-PRs. The complete list of combinations is available in our supplementary material (Correia et al. 2024). The 1st column lists each exception aspect combination, 2nd and 3rd respectively present the percentage and number of exception-PRs where that aspect occurred. The following sections discuss the top three combinations with real examples.

3.4.6.1

Error Recoverability and Error Representability

That was the most common combination found in our manual validation, constituting 12.99% (33 PRs). This combination mainly characterizes contributions that bond how errors are classified, handled, and reported. The principal aim of contributions with this combination were improvements in 57.58% and bug fixes in 39.39%, revealing a slight tendency for improvement in these aspects rather than bug fix issues. For instance, we identified examples on *improving the understandability of the exception handling*. In one of these cases⁵, a developer stated that “It’s hard to investigate [issue number]. Adding the thread dump might help us to identify the cause, or at least having a pointer to what to focus on”. In this case, the developer was investigating an issue and found that the existing error reporting mechanisms were insufficient, so he made changes to improve that. We also observed cases where developers encountered the opposite situation (i.e., excessive error reporting). An example of this was a situation described by a developer in a PR #1107⁶: “This [situation described in the issue] can print several thousand exception stack traces.” Since he found this behavior excessive, he proposed a solution to reduce the number of exceptions logged.

Another prevalent scenario that we observed was where the existing class modeling an exception was not satisfactory (*e.g.*, because it does not adequately describe the error, there is a more fitting exception that already exists elsewhere in the code⁷, etc.). While fixing a bug⁸, one developer described this as adding a “meaningful exception” to the error handling. We also observed cases⁹ where the existing exceptions in the code were insufficient to express (and report) a specific error, so a new exception class was introduced.

⁵<https://github.com/apache/flink/pull/21463>

⁶<https://github.com/apache/pulsar/pull/1107>

⁷<https://github.com/apache/flink/pull/4701>

⁸<https://github.com/apache/flink/pull/2969>

⁹<https://github.com/apache/flink/pull/8509>

3.4.6.2

Error Detectability, Error Recoverability, and Error Representability

This pattern was the second most frequent in our analysis, representing 7.09% (18 PRs), and its distribution in terms of aim is similar to the previous combination, improvements 55.56% and bug fixes 44.44%. In our observation, this combination mainly emerged for two reasons: (i) disagreement between validators and (ii) large and complex PRs. While these reasons can be observed separately, they blend into each other, as most disagreements emerged from PRs that were not easy for the validators to understand. This is understandable, given that while the validators had extensive language experience, they may not be experts in the project.

PRs exhibiting these aspects of the combination were typically very complex, with varied characteristics. However, most shared the common trait of affecting multiple parts of the exception pipeline. For example, consider a PR #2264¹⁰ submitted to the Druid project. When an exception occurred, the system would simply return a generic error to the user. To address this, the developer proposed a significant change by introducing a module called `CustomExceptionMapper`, designed to handle specific exception types. This module generates a detailed error message tailored to the particular exception and uses it to provide the user with more informative feedback. Additionally, the module was designed to be generic, enabling it to handle other exception types similarly. This change exemplifies the pattern by addressing three key aspects: detectability, recoverability, and representability.

3.4.6.3

Error Propagation, Error Recoverability, Error Representability

This was the third most frequent combination, mirroring the characteristics described in the previous pattern. This specific pattern accounts for 5.51% (14 PRs) of the cases, with its goals distributed as follows: 50.00% aimed at improvement and 42.86% at bug fixes. We observe that contributions to that aspect also address complex issues, primarily on *Error propagation*. Propagation is often a challenging topic for developers (Qiu *et al.* 2010), as it requires in-depth knowledge of how exceptions are handled at an architectural level. This handling is often far from trivial in large systems like those we analyzed, a fact noted by validators. This complexity also makes validating these types of PRs more difficult, especially since the validators were not project experts. Consequently, PRs adhering to this pattern often propose extensive and intricate changes, such as altering handlers so that exceptions are managed at a

¹⁰<https://github.com/apache/druid/pull/2264/>

different architectural level (e.g., client-side instead of server-side¹¹), or fixing exception handling within asynchronous environments¹².

3.4.7

Single versus Combined Exception Aspects

Table 3.8 illustrates that exception aspects frequently occur in combination rather than isolation. For example, *Error representability* appears in combination in 35.04% of PRs, compared to only 18.50% focusing exclusively on this aspect. Similarly, *Error recoverability* is observed in 33.46% of PRs in combined scenarios, contrasting with 12.20% as a single aspect. This trend suggests developers often address multiple exception-related aspects simultaneously when suggesting exception-related contributions.

Table 3.8: Aspects of exception-PRs appearing single or in combination.

	Single	Combination
Error representability	18.50%	35.04%
Error recoverability	12.20%	33.46%
Error detectability	5.51%	18.11%
Error propagation	2.76%	12.20%
Other	17.32%	2.35%

The prevalence of combined exception aspects may indicate two points about exception-related contributions. First, it underscores the interconnection of exception aspects. For example, improving error handling may concurrently require adjustments in *Error representability* to maintain message consistency. Second, it highlights the complexity of modern software systems, where exception management is not achieved through isolated contributions but through integrated strategies. Interestingly, we observe an inverse pattern for the *other* category, which happened more separately than combined. This result suggests that contributions not touching the exception-related code occur in isolation without impacting other exception aspects.

Finding 6: Developers frequently address multiple exception aspects simultaneously in exception-PRs, emphasizing the interrelation and complexity of exception management in modern software systems.

¹¹<https://github.com/apache/pulsar/pull/7430>

¹²<https://github.com/apache/flink/pull/9115>

3.5

Threats to Validity

Internal Validity. Our study analyzed three projects from Apache. While this limited scope might suggest a potential threat to the generality of our findings, the selected projects provided a substantial amount of data from different domains, allowing for robust analysis within a specific community. Regarding the heuristic, which was designed to select exception-PRs, it may not capture all pertinent changes. However, our heuristic was specifically designed to capture changes focused on exception, aligning closely with our research objectives. Additionally, by concentrating on human code contributions, we prioritized the quality and relevance of the data, ensuring its significance for our study.

External Validity. The results of this study are based on a carefully selected set of projects. In this matter, this selection process ensures depth and relevance in the studied context, even while limiting the applicability of our findings to similar environments or communities. We conducted manual validations by specialized validators using a sized sample. This may induce a subjective interpretation, potentially affecting the replicability of the validation process. However, the process included a double validation, where four authors resolved conflicts collaboratively. This approach minimizes human error and bias, ensuring high reliability of the data validation.

3.6

Conclusion

In this study, we explored the nature of exception-PRs in Java projects within the Apache ecosystem to understand the contributions to exception-related code and the attention they receive from developers compared to other types of contributions. Using a heuristic, we collected 988 exception-PRs from three repositories and selected 280 for manual analysis. We confirmed a precision of 90.7%, identifying 254 accurate exception-PRs out of the 280 evaluated. Our statistical analysis found no significant differences in developer engagement metrics, such as reviews and comments, between exception-PRs and non-exception-PRs. A detailed manual analysis of the 254 exception-PRs revealed that 55.12% aimed at improvements, 40.55% focused on bug fixes, with the most frequently modified aspect being external error representation (42.52%), followed by effective error handling (36.22%).

These results suggest that, while exception handling is recognized for its critical role in software robustness, it does not necessarily command more or less attention from developers in the review process compared to other types of

contributions. This finding raises important questions about the prioritization and attention of exception-related code, requiring further investigation. Moreover, by understanding the nature and characteristics of exception-PRs, we can better support developers in managing erroneous conditions and improving software robustness. Future work could explore automated tools and techniques to assist developers in creating and maintaining high-quality, exception-related code, ultimately contributing to more reliable and robust software systems.

Unveiling the Potential of a Conversational Agent in Developer Support: Insights from Mozilla’s PDF.js Project

A version of the work in this chapter appears in the **Proceedings of the 1st ACM International Conference on AI-Powered Software (AIWARE)**.

As discussed in Chapter 1, open-source software (OSS) projects are inherently collaborative, relying on frequent communication and coordination among contributors. Core maintainers, in particular, often face a high volume of technical questions, which compete with their already demanding responsibilities related to reviewing, merging, and maintaining code. To help alleviate this burden, Large Language Models (LLMs) have been explored as conversational agents capable of providing automated developer support across various domains.

To evaluate the feasibility and effectiveness of such agents in addressing project-specific technical inquiries, we conducted an empirical study using DevMentorAI, a Retrieval-Augmented Generation (RAG) system designed to assist developers in OSS contexts. DevMentorAI integrates project-specific documentation and source code, enabling it to generate more contextually relevant and up-to-date responses than general-purpose models. When assisting developers, such agents must retrieve appropriate materials, interpret the surrounding technical context, and compose accurate, actionable answers—a particularly valuable capability in OSS settings, where documentation is often fragmented, inconsistent, or outdated.

Our study focused on the Mozilla PDF.js project, a widely used OSS library for rendering PDF files in the browser. We compared responses generated by DevMentorAI to those provided by human contributors within the project’s development chat. A Mozilla expert evaluated the responses across 14 real-world developer questions. DevMentorAI’s answers were judged to be more satisfactory in 8 cases, equally satisfactory in 3, and less satisfactory in only 3 cases. These results indicate that a well-configured RAG-based assistant can effectively complement human expertise by providing accurate, timely, and context-aware support—ultimately reducing the cognitive and communicative load on maintainers.

While this study centers on question answering, it represents a pivotal step in a broader research trajectory: transitioning from reactive information support to proactive, code-oriented interventions that enhance software dependability. In OSS environments, where the lack of up-to-date documentation and high contributor turnover can obscure architectural intent and introduce subtle flaws, even small gaps in shared understanding can compromise attributes such as maintainability, reliability, and robustness. By helping developers navigate exception-handling behavior, hidden system constraints, and undocumented design decisions, conversational agents like DevMentorAI not only support daily collaboration but also help prevent long-term degradation of system quality.

This insight lays the foundation for the next phase of this thesis, which investigates whether such agents can move beyond answering questions to identifying and repairing dependability-related issues directly in the code. In doing so, we explore the potential of conversational agents not just as assistants in communication but as active contributors to the production of resilient, maintainable, and trustworthy software systems.

This chapter presents the findings of this study, which are documented in the paper *Unveiling the Potential of a Conversational Agent in Developer Support: Insights from Mozilla's PDF.js Project* (Correia et al. 2024). **The study constitutes a key contribution of this Ph.D. thesis and directly addresses the research question: *To what extent can a conversational assistant match human-provided answers to technical questions?***

4.1

Introduction

Open source Software (OSS) projects depend on open collaboration environments, in which a distributed community of developers and developers work in sync to develop the project (Brisson *et al.* 2020, Kramer *et al.* n.d., Bergquist 2020). For these projects to thrive, they must create a sustainable ecosystem in which communication between users and developers is constant and healthy (Brisson *et al.* 2020, Tsay *et al.* 2014, Dabbish *et al.* 2012). As such, these communities often employ communication tools such as forums, chat clients, and issue trackers, enabling users and developers to more easily perform communication tasks, such as asking and answering questions that may arise throughout the development or usage of the project.

Furthermore, newcomers willing to join these projects have overcome several barriers to onboard (Steinmacher *et al.* 2015), and rely on community resources to overcome them. This is often done through informal mentoring (Feng

et al. 2022), a practice in which more senior developers volunteer their time to answer questions from newcomers. However, this practice does not scale, as the time and effort required from the project members are significant (Feng *et al.* 2022, Steinmacher *et al.* 2021), often to the detriment of other important project-related tasks.

Communities have used different ways to scale up the way they provide resources for users and newcomers, such as creating documentation that covers different facts (Steinmacher *et al.* 2014). However, it is known that the documentation in OSS projects is usually spread, disorganized, and lacking logical relationship (Fronchetti *et al.* 2023). As a result, communities continue to depend on humans to convey the information that is needed by newcomers effectively. Nevertheless, communities have continued to experiment with alternative ways to provide this information that are more efficient than utilizing human labor.

Foundation models, such as GPT-4 (Achiam *et al.* 2023), have demonstrated their ability to provide accurate and satisfactory responses to a wide range of general inquiries (Dakhel *et al.* 2023, Sallou *et al.* 2023, Wang *et al.* 2024, Pinto *et al.* 2023), and thus are being extensively utilized by developers to help them solve problems related to the projects they participate (Ross *et al.* 2023, Stack Overflow 2023). Since these models can understand different types of content related to the project (e.g., source code, documentation, diagrams, etc.) they can efficiently organize information from disparate sources of information to provide information that helps new developers and users in answering their questions (Pinto *et al.* 2023, Pudari and Ernst 2023).

However, most widely available tools utilized fixed, general, and sometimes outdated datasets as their primary source of information (Azaria *et al.* 2024). Since software projects have specific information, processes, and code-bases—which are constantly evolving—it can be difficult to ensure that the information these tools provide is accurate (Mylavarapu *et al.* 2019). Most of these tools are proprietary, providing limited experience in reporting how these foundation model-based assistants can help developers with project-specific questions, especially more specific questions requiring specialized knowledge.

To address this challenge, we introduce DevMentorAI, a conversational assistant that uses continuous integration to use project contextual information (reducing the risk of providing incorrect information). To accommodate for the specifics of different communities, DevMentorAI utilizes a modular approach, and thus can be easily integrated with different communication tools, such as forums, chat clients, collaborative development environments, etc.

To tackle the lack of experimental reports about how these conversational

agents are helping developers, this paper presents a case study. In this study, DevMentorAI was applied to answer 14 real questions that were asked and answered by human developers in the chat room for `PDF.js`,¹ an OSS project managed by the Mozilla Foundation. A project expert from Mozilla (the main developer) was then tasked with evaluating how satisfied they were with each of the answers (*i.e.*, the human's and DevMentorAI's answers).

In summary, the expert perception was favorable towards DevMentorAI's answers in most questions (8 of 14 questions). In 3 questions, they were considered equivalent, and only in 3 questions, the expert was more satisfied with the human answers. To better understand these results, we also discuss individual cases and point out factors that may have influenced both the performance of DevMentorAI and the human, while also pointing at some factors that may have influenced the expert's evaluation.

Contributions: This paper has two main contributions: (i) we present DevMentorAI, a conversational agent that can help developers by utilizing real-time information, therefore helping reduce the time and effort expended by human developers in mentoring, and (ii) we report a case study in which DevMentorAI was utilized to answer 14 real questions asked by developers and users from a large open-source project (`PDF.js`). We then tasked a project maintainer from the project, with comparing how satisfied he was with the answers provided by the humans and by DevMentorAI.

4.2

Related Work

This section presents related work on large language models used to support software development activities.

The development of Large Language Models (LLMs) has accelerated significantly over the past few years, marked by the introduction of models such as Gemini (Gemini et al. 2023), BERT (Devlin et al. 2019), and GPT-3 (Brown et al. 2020). These models can generate human-like text, summarize content, and perform a wide range of natural language processing tasks. Recent work (Dakhel et al. 2023, Sallou et al. 2023, Wang et al. 2024, Pinto et al. 2023) have been investigating the potential of LLMs in supporting or acting as assistants for SE activities. Dakhel *et al.* (Dakhel et al. 2023) explore Copilot's effectiveness in providing correct, efficient, and reproducible solutions for fundamental programming tasks, and assess whether Copilot's solutions are competitive with those created by humans. Sallou *et al.* (Sallou et al. 2023) emphasizes concerns such as closed-source models, data leakage, and

¹<https://github.com/mozilla/pdf.js>

the reproducibility of research findings involving LLMs. Moreover, Balfroid *et al.* (Balfroid et al. 2024) explored the potential of using LLMs for generating annotated code tours to help the onboarding process of new developers in software projects. Their findings show that the LLM, while able to generate code and present solutions, often gives vague information or gives the same content present in the documentation.

Pudari *et al.* (Pudari and Ernst 2023) studied the current capabilities and limitations of AI-supported code completion tools like GitHub's Copilot. It explores how these tools deal with programming idioms and code smells. The study finds that Copilot often does not suggest the most idiomatic or best practice code, indicating areas for future improvement in AI-supported programming tools. Moreover, Sauvola *et al.* (Sauvola et al. 2024) investigated the impact of generative AI on software development, analyzing the opportunities, challenges, and ethical concerns surrounding AI in enhancing productivity, and automation in software development. The study emphasizes the need for new tools and guidelines to navigate the world of generative AI to better understand its role in software development practices and operations. Finally, Pinto (Pinto et al. 2023) explores the use of ChatGPT for grading and providing feedback on open-ended questions within technical training contexts. Their findings suggest that ChatGPT can identify semantic-related details in responses and it is generally aligned with expert corrections, indicating its potential use in the feedback process for open questions in educational environments.

Our study differs from prior research in two main ways: (i) we designed a conversational assistant that supports interfacing with different generative models (currently GPT-3.5 Turbo). Using these models, we employed a Retrieval-Augmented Generation (RAG) approach, focusing specifically on documentation and source code from the target project. As such, our assistant is specialized in answering questions related to the project (beyond code); (ii) we assessed the quality of the responses of our custom assistant by comparing them with actual real-user answers, which were subsequently evaluated by a project expert.

4.3 DevMentorAI

In this section, we present the details about DevMentorAI. The source code is available on our replication package (Replication Package 2024).

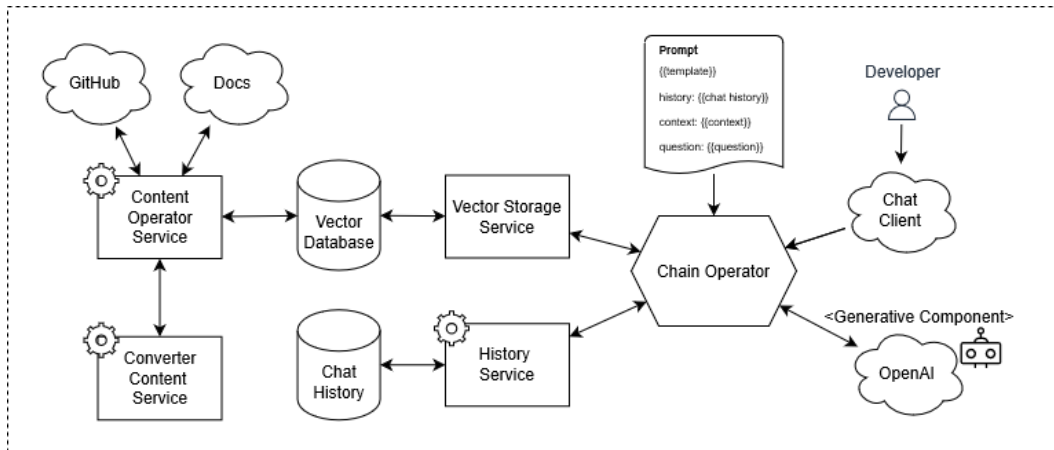


Figure 4.1: DevMentorAI's Pipeline.

4.3.1 Requirements

We designed DevMentorAI as a conversational assistant aimed at aiding developers by offering relevant responses to project-specific questions. We focused on creating a tool that would enable compatibility with various platforms, like chat clients, forums, and issue tracker, while maintaining a swift response time to seamlessly integrate into the developer's workflow. Additionally, a core aspect of DevMentorAI's functionality is providing accurate and current data, ensuring that developers have access to the latest information relevant to their projects.

4.3.2 Architecture

Given the goals presented above, we opted to design DevMentorAI by leveraging the hexagonal architectural framework (Stenberg 2014). By using this architecture, DevMentorAI's execution follows the pipeline summarized in Figure 4.1.

The use of the hexagonal framework simplifies the development of new adapters for different data sources (*e.g.*, scrapers, PDF readers, git)—via the content operator and converter content services (left of Figure 4.1)—and enables smooth integration with different communication clients (*e.g.*, GitHub issue tracker, Matrix Chat, Discord)—which can be plugged to the chain operator (right of Figure 4.1). This design enables extensibility in terms of inputs and outputs, and easier maintenance for different projects and contexts. This novel way of building a contextual agent makes DevMentorAI's architecture a good fit for several projects, regardless of the source of documentation and communication channels they use.

Next, we will discuss the individual components developed for DevMentorAI, shown in the figure.

4.3.2.1

Vector Storage Service

The first step to creating a contextual agent is collecting the information from the data sources. For retrieving the up-to-date information that DevMentorAI utilizes to answer questions, we opted for the Retrieval-Augmented Generation (RAG) strategy (Lewis et al. 2020). As such, we developed a component that ingests the projects' documentation and source code for the assistant's usage. This occurs in two steps. First, this component processes the artifacts from the project, and second, it then identifies similarities between the questions and the project context (via a vector space). We will detail each of these steps as follows.

Obtaining the raw sources. DevMentorAI utilizes the *Content Operator Service* to capture the target project source code and documentation (by integrating with git/GitHub) and other documentation web pages (through web scraping). When needed, a universal text format converter is utilized (*Content Converter Service*, which currently utilizes *Pandoc*) to standardize raw text into Markdown. The processed content is then broken down into smaller chunks for targeted analysis of source code or documentation relevant to specific questions. This approach avoids comparing entire documents or code modules. Programming language and text semantics are respected during this stage to maintain coherence and syntax.

Content Vectorization. Each excerpt of raw content is then transformed into a vector and stored in a database, thereby updating the project's context. This vectorization process is done to optimize searching for documents or source code chunks similar to the question being asked (*i.e.*, providing context for an answer).

4.3.2.2

Generative Component (OpenAI)

DevMentorAI's architecture was planned to accommodate different generative AI tools—we call them “Generative components” in the scope of our architecture. For the sake of this study, we utilized OpenAI's *GPT-3.5 Turbo* with a 4K token context window as our core generative model. Our selection of OpenAI's GPT model was driven by its status as the leading language model available during our study. Additionally, we opted for *GPT-3.5 Turbo* over *GPT-4* due to monetary constraints. The communication between our assis-

tant and the generative model was executed using the reference API provided by OpenAI².

4.3.2.3 History Service

To increase the richness of the contextual information input provided to the generative model, we provide a rolling history of the conversation to the prompt. To do so, we adopted the `ConversationBufferMemory`³ construct from LangChain (Chase et al. 2022). This construct was chosen given its ability to provide a history without the necessity to first use Generative AI to summarize historical data. This strategy also reduces the response time and increases the accuracy of the answers.

4.3.2.4 Chain Operator

On the other side of the architecture, the user posts questions that are handled by a conversation chain, which is operated by a “chain operator.” When the user asks a question, this operator is invoked and passes the question to the pipeline. The operator has a pre-set prompt template that needs to be filled with certain variables, as in the top-left of the example in Figure 4.1. This prompt has details about the persona, which is information related to how the assistant should behave. The prompt also contains relevant details the assistant should consider to generate the answer. This template requires three pieces of information: *history*, which is the history of previous interactions, *context*, which is the knowledge relevant for answering the question (*i.e.*, documents and source code chunks similar to the question asked), and, of course, the *question* posted by the user.

The *history* is filled by the *history service*. It keeps a rolling window of previous interactions with the user. The *context* is filled by asking the *vector storage service* to search for chunks that are similar to the vector representation of the question asked. Once the relevant vectors are retrieved, the full contents of these chunks are restored from the metadata to fill the context variable.

4.4 Methodology

The goal of this study is to assess how DevMentorAI answers compare to those written by humans in a project-specific context. So, we conducted a case

²<https://platform.openai.com/docs/api-reference>

³<https://python.langchain.com/docs/modules/memory/types/buffer>

study using real past questions in a development chat channel and analyzed how DevMentorAI performs in those questions. Thus, our research question is:

RQ: How does DevMentorAI compare to humans when responding to project-specific questions?

4.4.1 Study Design

We conducted four main methodological steps to answer our research questions, described in the following sections.

4.4.1.1 Project Selection

The Mozilla Corporation maintains multiple open-source projects in several domains, most of which could be suitable for our investigations. To assess our approach we selected the project to conduct our evaluation based on three criteria: (i) the discussions on the project should be openly available, (ii) our collaborators at Mozilla should have contact with the engineers of the project, and finally (iii) the project should be ingested by our model in a reasonable time.

Here is some rationale supporting our criteria. First, we needed access to the discussion channel of the project. Therefore, we limited the projects' universe to those with open discussions on Matrix—the Mozilla platform for discussions. Second, since our collaborators were supposed to evaluate the DevMentorAI answers for real questions, we restricted the projects to those in which we could reach the engineers of the project. Third, ingesting the project's repository artifacts takes a long time, depending on the project size, so we opted for small or medium size projects. Therefore, observing these criteria, we selected the `PDF.js` as the subject for our study.

4.4.1.2 Ingestion of Project's Artifacts

Our assistant ingested technical documentation and source code extracted from `PDF.js` to conduct this case study. We collected all the artifacts from the project `mozilla/pdf.js` hosted on GitHub. It is worth mentioning that our assistant supports ingesting source code from several languages in addition to markup files. Aiming to standardize the file's vector computing, we employed Pandoc (Pandoc) to convert the markup files such as `.html` to `.md`.

In general, for the PDF.js, we ingested source code files, wiki pages, and web pages comprising files in the following formats: *js*, *jsx*, *mjs*, *cjs*, *md*, *html*, *pdf*, etc. To assess the effectiveness of artifact ingestion, we conducted some preliminary isolated tests, which produced contextualized outcomes when asking questions about the PDF.js.

4.4.1.3
Selecting the Questions

With the subject project chosen, we started gathering the conversations in the chat channel, which project developers and users use. We accessed the PDF.js Matrix room⁴ and exported the chat history leveraging a built-in functionality. We collected all messages and their metadata until January 16, 2024, in a JSON format.

Since our interest was to obtain textual questions and their answers, we made an initial data processing using Python scripts. We created as output a spreadsheet containing (i) the message timestamp, (ii) whether the message was a reply from a previous message, and (iii) its textual content. This process resulted in a spreadsheet containing 3492 messages. Next, one author manually inspected each message to determine whether that was a question with a proper human response. This process was conducted by one researcher and validated by a second.

During the manual analysis, we identified several questions throughout the chat history. However, the preponderance of these questions proved unsuitable for inclusion in our evaluation. Initially, we discarded cases where questions were posed, and other users requested more information, leading to threaded discussions to completely understand the question’s context. Additionally, we disregarded unanswered or questions that required access to external resources. Finally, non-technical queries were excluded from consideration. Below, we have an instance of an ignored question and the reasons presented above.

Question discarded in the manual analysis
Hi all, I’m trying to implement the pdfjs on a certain project. I was able to implement the example on the official site that uses the pdfjs-dist script, but I am a little confused as to how I can implement this example: https://mozilla.github.io/pdf.js/web/viewer.html in a similar manner?

We ignored the aforementioned example predominantly due to its lack of answers. Furthermore, accessing the link to check the code example provided in

⁴<https://chat.mozilla.org/#/room/#pdfjs:mozilla.org>

the question is necessary to gain context and provide an accurate answer. Since the model did not have access to external artifacts, that was another factor to classify that question as unsuitable. After the manual analysis, we ended up considering 14 questions and their respective answers for this preliminary analysis. It is worth noting that some users asked questions using multiple messages. In these situations, we concatenated the messages to conceive the question. The same approach was adopted for the answers. Still, follow-up questions were not considered.

4.4.1.4 Expert Assessment

After selecting the fourteen questions, we submitted each of them to DevMentorAI. We implemented an integration with Discord (Discord) and performed the following steps for each question. First, we started a new chat, which created a new and empty context for the conversation. Next, we copied the question we identified in the chat and submitted it for the assistant consideration. DevMentorAI processed the data going through our vectorial database and sending the contextual prompt to the foundational model. Finally, we received the answers.

We structured the data from the previous steps in a spreadsheet with three columns. The first column contained the question posted; the second had the human answer as posted in Matrix; and the third presented the responses generated by DevMentorAI. Subsequently, we added two more columns to that spreadsheet to enable the specialist assessment, which we describe in the following paragraph.

Since we conducted this study in collaboration with the Mozilla Corporation, the answers were assessed by an expert with vast experience in programming (15 years), 4 years experience on the target project (*i.e.*, PDF.js), and that is currently the project's main developer. We asked this expert to answer the following questions for each question: "How satisfied are you with the human answer?" and "How satisfied are you with the bot's answer?". Both questions enabled answers following a 5-point Likert scale, as follows: *not at all satisfied* (0), *slightly satisfied* (1), *moderately satisfied* (2), *very satisfied* (3), and *completely satisfied* (4).

To further understand the context and details of the conversations, we explored the sequence of the chat analyzing the messages that followed the question and answer on Matrix. To do so, we went back to Matrix and manually analyzed the messages exchanged about the topic, providing us with further evidence of the satisfaction and usefulness of the answers provided in the chat.

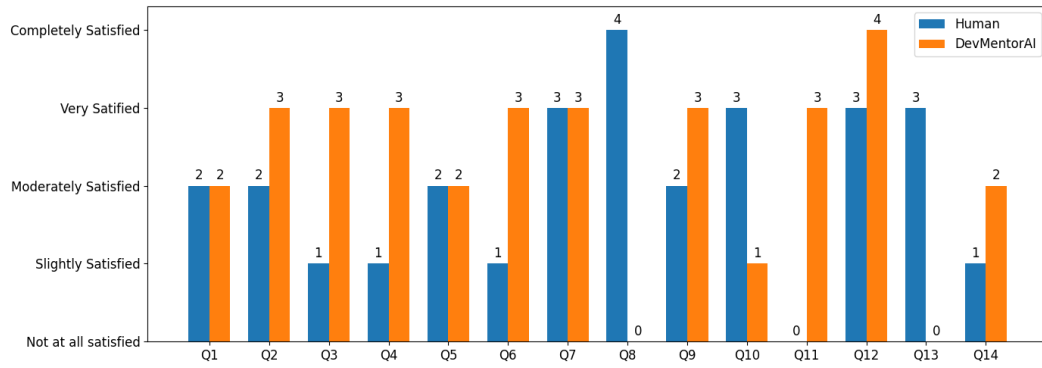


Figure 4.2: Expert ratings for questions in a Likert scale.

4.5 Results and Discussions

In this section, we present the results of our study to characterize the DevMentorAI performance in answering project-specific questions and the factors that influence their responses.

4.5.1 Overall Performance

As described in Section 4.4.1.4, we compared the performance of DevMentorAI with the actual humans' answers when addressing questions regarding PDF.js. For that, a PDF.js expert from Mozilla gauged its satisfaction with the answers the humans and our assistant provided. This expert has extensive experience in programming and on PDF.js, featuring as the project's main developer.

Figure 4.2 summarizes the results from the comparative analysis. The x-axis shows the questions, while the y-axis provides the ratings for answers. The performance evaluation yielded the following results: (i) DevMentorAI surpassed the human in 8 questions; (ii) in 3 questions, both exhibited comparable performance; and (iii) in 3 questions, the human outperformed DevMentorAI. One can observe that in all the cases in which the expert was more satisfied with the human's answer, the difference was at least two points on the Likert scale. On the other hand, DevMentorAI had this degree of superiority in 5 cases (out of the 8 cases mentioned in (i)).

4.5.2 Discussion About Individual Cases

In this section, we discuss the perceived performance of DevMentorAI in specific cases. We also point at factors that may have influenced both

how DevMentorAI responded and how the expert evaluated the case. These examples will be divided into three categories. First, the cases in which the response from DevMentorAI was perceived as better than that from the human. Second, the ones in which the response from the human was perceived as better. Third, those cases in which they were perceived as similar. Although we explore exemplary cases for each category, all the questions and answers are available in our replication package (Replication Package 2024).

4.5.2.1
DevMentorAI Outperforms Humans

This scenario was observed in 8 out of the 14 questions analyzed. In this section, we present three exemplary cases in which the expert ranked DevMentorAI’s answer as more satisfactory than the human’s answer.

Question 11
<p>Hi, I have a very large pdf in size (200mb) and I want to download it faster. How to do that in PDF.js viewer?</p> <p>1. Pdf is linearized. 2. Using rangeChunkSize (1024 1024).</p> <p>But it is downloading chunk in sequentially but I want them to be downloaded parallely. Is there any configuration available to control the number of parallel requests?</p>
<p>Human’s Answer:</p> <p>Download it and use an other viewer.</p>
<p>DevMentorAI’s Answer:</p> <p>No, there is no configuration available in the PDF.js viewer to control the number of parallel requests for downloading chunks of a large PDF file. The viewer automatically handles the downloading and rendering of the PDF file, and the number of parallel requests is determined by the browser’s network stack.</p>

In Question 11, someone asked if they could use the project in a configuration that is not supported. The human who answered provided a very short and direct answer, which one could perceive as ungentle or even uncaring, due to the dismissive way it was phrased. By analyzing DevMentorAI’s answer, we observe that it explains to the participant the specifics of why it was not possible. Consequently, the expert evaluated that they were *not at all satisfied* (0) with the human’s answer, and *very satisfied* (3) with DevMentorAI’s. When we went in-depth into the conversation, we observed that the person asking the question provided more information after getting the rough answer, and the

same respondent posted other rough comments, including an image of Taylor Swift with the following excerpt of her song: "I will never shut up about."

Question 12
Is there a pdf.js associated inspector-like tool for viewing pdf internals? I recall seeing something like this before.
Human's Answer: yep: http://brendandahl.github.io/pdf.js.utils/browser/ .
DevMentorAI's Answer: Yes, there is a PDF Object Browser tool available for inspecting the internal object structure and viewing raw values of a PDF document. You can access the PDF Object Browser using this link: https://brendandahl.github.io/pdf.js.utils/browser .

In Question 12, the user asked for a tool included in the project. The human responded precisely, with a short confirmation and a link to the tool the user was looking for. DevMentorAI also replied with the same link but briefly described the tool. The expert was satisfied with both answers and slightly preferred DevMentorAI's answer. They reported being *very satisfied* (3) with the human response and *completely satisfied* (4) with DevMentorAI's response.

We can observe that, in both cases presented here, the humans provided short answers, with poor context or explanations. This behavior may have multiple possible causes, including stress and lack of time. Moreover, this problem may have varying levels of severity. For instance, responses were perceived as ungentle or even toxic in the first question, but acceptable ones as the second one. This kind of behavior shows a problem inherent to human responses: the human might not always be fully available to provide a complete response. In these cases, we observe an advantage that is inherent to assistants such as DevMentorAI. Not only they are always available to answer the user but also provide an answer to the full extent of content they have access.

Question 3
Hello! I was trying to switch to reader view on FF on a badly typo-graphed PDF. Can this be done somehow? if its not a feature already, where can I suggest it?
Human's Answer: I don't know how <i>*technically*</i> feasible it is, but I just got to https://www.supremecourt.gov/opinions/20pdf/18-956_d18f.pdf actually that's not a bad idea :) beyond the fact that it's all white space and small font, I get a lot of light sensitivity so prefer a dark background (which reader mode allows) You can suggest it by filing a

bug on bugzilla.mozilla.org

DevMentorAI's Answer:

To switch to reader view in Firefox for a PDF document, you can follow these steps:

1. Open the PDF document in Firefox.
2. Click on the "Reader View" icon in the address bar (it looks like an open book).
3. The PDF will be displayed in a simplified, distraction-free reader view.

If you have any suggestions or feature requests for Firefox, you can submit them to Mozilla's Bugzilla platform. Here is the link to file a new bug report: https://bugzilla.mozilla.org/enter_bug.cgi?product=Firefox

Make sure to provide a clear description of your suggestion and any relevant details that can help the developers understand and evaluate your idea.

Question 3 illustrates an inherent advantage that virtual assistants can have over humans. In this case, the human was not aware of a feature that was already implemented in the software. Therefore, they only instructed the user on how they could suggest this feature to the development team. On the other hand, since the virtual assistant has access to several sources of up-to-date information about the project, it was aware of the existence of this feature, thus it both instructed the user on how to use the feature and how to suggest new features (the user also requested this information). As such, the expert was slightly satisfied (1) with the human's answer, and very satisfied (3) with DevMentorAI's answer.

4.5.2.2

DevMentorAI Performs as Good as a Human

Here, we explore two of the three cases in which the expert was equally satisfied with the answers from the human and DevMentorAI.

Question 7
Hi folks, If I have a PDF that's 100MB and 100 pages long, and I want to direct a user directly to page 50 for example. Can PDF.js load that page only without fully loading the entire PDF? Or load only that range of pages...
<div><div>Human's Answer:</div><div>If your sever supports range requests, then yes pdf.js should be able to only load part of the pdf.</div></div>
<div>DevMentorAI's Answer:</div>

PDF.js can load specific pages or a range of pages from a PDF without loading the entire document. This is achieved through the use of HTTP Range Requests, which allow for fetching only the required portions of the PDF needed for rendering the visible pages. However, the ability to utilize HTTP Range Requests depends on browser support and the web server returning the appropriate headers. If these conditions are met, PDF.js will automatically start fetching the necessary portions of the PDF for rendering, allowing for a more efficient and faster loading experience.

In Question 7, both the humans and DevMentorAI provided answers with very similar content. However, one can quickly notice that the answer from the agent is more verbose, at least 4 times bigger than the human’s. This is similar to what we observed in the previous section, with DevMentorAI providing more details and explanations about the answer. The expert reported to be *very satisfied* (3) with both answers. We believe that DevMentorAI’s answer can benefit less experienced people or those with a comprehensive information processing style.

Question 5
Hey everyone! I’m having a frustrating issue where I’m trying to allow PDFs to be previewed in browser. I’m including them via URL in an ‘embed‘ or ‘object‘ HTML tag currently and while it works after installing the browser extension, it still downloads every PDF file that gets previewed on the page. Is there a way to prevent that?
Human’s Answer: To preview anything, it first needs to be downloaded. Nobody reads online. All content is downloaded and read locally from browser cache.
DevMentorAI’s Answer: No, there is no built-in way to prevent PDF files from being downloaded when previewed in the browser using an ‘embed‘ or ‘object‘ HTML tag. These tags are used specifically for embedding content in web pages and allow users to download the file if they choose. If you want to restrict the downloading of PDF files, you would need to implement custom logic on the server-side or use other methods like DRM (Digital Rights Management) to control access to the files.

For Question 5, both the human and DevMentorAI acknowledge that it is not possible to perform the action requested by the user. Once again we see that the human is more objective and direct than the agent. The expert reported to be *moderately satisfied* (2) with the answer. This may mean that the expert believes that the answers are not complete in any of the cases. For the human it may represent a shortage of knowledge, and for DevMentorAI, a lack of appropriate and accurate information in the project repository. By investigating the chat logs, we see that the conversation kept moving, and,

at one point the human mentioned “*Sorry I am not expert,*” while they kept trying to sort out the program together.

4.5.2.3
DevMentorAI was Outperformed

Finally, we discuss two of the three cases in which the answer from the human was more satisfactory than the ones from DevMentorAI. One important observation before we dive into the details, is that in these cases the answer from the agent received really low scores. The expert was not satisfied at all (zero score) in two cases.

Question 8
Hi, I wonder how to use the zoom commands, if I want to zoom into a pdf say 400 and define a left and top offset? https://mozilla.github.io/pdf.js/web/viewer.html?file=compressed.tracemonkey-pldi-09.pdf&#zoom=300,10,1 brings a zoomed version with a left offset but jumps to the second page. What do I have to change to get a top offset of 200px?
Human’s Answer: In the pdf coordinate system 0,0 is the bottom, left of a page.
DevMentorAI’s Answer: To modify the zoom commands in the PDF.js viewer to achieve a top offset of 200px, you can use the ‘zoom’ option with the desired values. The format for the ‘zoom’ option is ‘[zoom],[left offset],[top offset]’. Here is an example URL that demonstrates how to set the zoom level and top offset: https://mozilla.github.io/pdf.js/web/viewer.html#zoom=200,0,200 . In this example, the zoom level is set to 200 and the top offset is set to 200px. You can adjust the values as per your requirements. Please note that this modification will only affect the viewer for the current session and will not permanently change the default behavior of the PDF.js viewer.

Question 8 is a complex case. As in the previous cases, the answer from the human is very short and simple. It just points out to the participant that the 0,0 position is the bottom left. The expert is *completely satisfied* (4) with this answer, understanding that the user who posted the question would be able to figure out how to go to the specific place in the PDF. Interestingly, DevMentorAI gives misleading information, saying that “*The format for the ‘zoom’ option is ‘[zoom],[left offset],[top offset]’.*” Notice that the two last parameters here point to the **top**-left as the 0,0 position, while the human pointed to the **bottom**-left. This caused the dissatisfaction (zero score) of the expert. However, when analyzing the document presented in DevMentorAI’s answer and the thread on Matrix, we see that (i) DevMentorAI found the appropriate resource and presented the information as provided in the documentation; and

(ii) the information was incorrect on the documentation, as per the follow-up from the human respondent: *“hmm...I think that’s wrong [...], but it should be changed or described in the wiki as well...”* Due to this divergence—caused by the issue with the misleading information in the documentation—the human reported to be *completely satisfied* (4) with the human answer and *not at all satisfied* (0) with DevMentorAI’s answer.

Question 10
<p>Hi everyone, I’m running into a bit of an issue with PDF.js and chrome. I’m not sure I’ve got enough information for a true github issue, but I wanted to drop it here to see if you might have any insight into the issue.</p> <p>In the image above you can see a sample of the problem: On my coworkers machine PDF’s are often rendered like this, with random bits and pieces of the text characters missing. This seem to happen on his machine and at least one other but not on mine, and it only happens in Chrome and in Edge but not when using Firefox.</p> <p>It seems to happen to many, but not all, the PDF’s, and it isn’t consistent: a page reload can cause the anomalies to end up entirely somewhere else in the document. I have attached an example PDF that exhibits the problem for my coworker.</p> <p>This goes wrong in our own application and in the slack PDF viewer (which seems to use PDF.js), but in the PDF.js demo page (opening the example PDF) does work correctly. https://mozilla.github.io/pdf.js/web/viewer.html</p> <p>I initially blamed my coworkers machine, but as we have now also seen it on another machine it is suspicious. Have you every seen anything like this? I’m pretty stumped. Any insight on this matter is greatly appreciated!</p>
<p>Human’s Answer:</p> <p>Maybe you should try to test in enabling/disabling hardware acceleration in Chrome I tried myself on Chrome on Windows 11 with enabled/disabled ‘Accelerated 2D canvas’ and it works well.</p>
<p>DevMentorAI’s Answer:</p> <p>I’m sorry, but I don’t have access to personal experiences or encounters. However, it is possible to encounter rendering issues with PDF.js in different browsers. If you are experiencing this specific issue, it would be best to report it to the PDF.js development team with the necessary details such as the PDF file, browser version, operating system, and any error messages you may have encountered.</p>

In Question 10, a developer contextualized the issue explaining that they are experiencing rendering problems with PDF.js and Google Chrome on particular machines and asked for insights into what could be causing the problem. The human answered with something that seemed like a personal experience. They guided the user to turn off the hardware acceleration in Google Chrome to solve the problem. In contrast, DevMentorAI answered that it cannot access personal experiences (given the personal question asked: *“Have you every [sic!] seen anything like this?”*). The expert classified the

human's answer as *very satisfied* (4) and the DevMentorAI's answer as *slightly satisfied* (2). The satisfaction with the human answer goes beyond the expert, the user replied to the human in the thread saying " *I tried myself on Chrome on Windows 11 with enabled/disabled 'Accelerated 2D canvas' and it works well.*"

4.6

Limitations and Design Trade-offs

In this paper, we presented a case study to assess how satisfactory the answers of a conversational agent would be in the context of an OSS project. Our decisions on the study conception may have an impact on the results. First, we had to choose a subject project to run the study, which is a key decision in a case study. While many factors may influence this decision, we focused on finding a project maintained by Mozilla (a partner on this research) with experts on their reach, and that was not too big to ingest into our model for our preliminary evaluations. We acknowledge that our research results are influenced by the specific context of our case study (PDF.js) and that it limits the generalizability of our results. However, we preferred to keep more control and work on a single and smaller project, supported by the maintainer, which entitled us to be more careful in the data collection and better understand the context.

A second point regards evaluation from the expert. We could have conducted in-depth interviews or collected multiple opinions to get stronger evidence. Once again, considering the factors that led us to decide, we wanted to guarantee that we would get appropriate responses that would not require too much time or effort from the expert. Still, it is well known in the literature that recruiting is not an easy task in Software Engineering, especially when we deal with OSS (Kokinda *et al.* 2023, Wagner *et al.* 2020). Therefore, since we are collaborating with Mozilla, and we had access to an expert, we preferred someone to provide us with accurate feedback. Although we would have a more in-depth understanding using other methods, we are very comfortable that the input received is reliable and accurate, given the expertise and willingness to help from this person.

We also had to decide on a specific LLM to use. While this may pose a threat, we used OpenAI's *GPT-3.5 Turbo*, since GPT was the state-of-the-art LLM at the moment of the study, and we opted for *3.5 Turbo* instead of *GPT-4* due to budget constraints. Even using a previous version, the results proved to be very encouraging. Still related to the LLMs, we acknowledge that they are subject to continuous updates and modifications, and threaten the

consistency and replicability of our results. Nonetheless, in contrast to related studies, large language models (LLMs) rapidly evolve and enhance their models daily. Consequently, we anticipate that the quality of responses may improve progressively over time.

Some decisions made during the data curation process are also worth discussing. Given the context of the project under study and the choice of the LLM used, we had to dismiss questions that embedded images. This is an important decision because `PDF.js` is a PDF rendering tool, and screenshots and PDFs were part of a non-negligible number of messages. Although we missed some questions, our sample size was not too small and enabled us to understand different aspects of DevMentorAI performance. In future work, we plan to use Foundation Models that read images and documents to understand how they improve the answers.

We acknowledge that DevMentorAI answers may be limited for some lack of context. DevMentorAI had no access to the complete chat history. Therefore, any contextual information provided before those messages used to compose the question was not accessible by the agent. Moreover, since the Matrix channel is open, the human's answers might not be from developers from `PDF.js`, but from users. This may have resulted in suboptimal answers coming from them as well. However, this does not pose a threat since the answers analyzed are the ones that were actually posted to the questions.

Another point is that we did not control documentation in terms of accuracy and up-to-dateness of the information ingested. There may be cases in which the answers could be improved. Although these points might have influenced the quality of the output, our results were very satisfactory.

Still, two decisions related to the design and implementation of DevMentorAI may pose limitations. First, we decided to ingest data available on GitHub to our conversational agent. There may be more documentation outside of GitHub boundaries that were not considered. This may have reduced the accuracy of our responses. However, once again, the preliminary assessment is very encouraging, and this may be included in the following steps of this research. Second, we used cosine distance to compare the query and the vectors stored in our database as a step to create the prompt sent to the LLM. Although we acknowledge that other options would offer different results, we opted to keep a conservative approach to assess how the agent would perform with these conditions.

4.7

Conclusion

This paper introduces DevMentorAI, a conversational assistant that uses the Retrieval Augmented Generation (RAG) architecture to facilitate developers' interactions with software projects by integrating project-specific data. By doing a case study of the `PDF.js` project, DevMentorAI proved its capability to provide answers that, in most cases, were better than the ones from human participants. Our goal was to, by integrating project-specific data, assess if DevMentorAI could make the onboarding process smoother for new developers and automate responses to project-specific questions.

Moreover, this study highlights the importance of integrating detailed and context-specific information into conversational agents to improve their effectiveness. As foundation models continue to evolve, their integration into various aspects of software engineering and development promises to enhance productivity and facilitating processes. The findings from this study not only demonstrate the feasibility and utility of such models in practical applications but also pave the way for future research.

For future work, we aim to explore its application across a wider range of software projects to understand its adaptability and impact on developer productivity in diverse development environments.

Acknowledgements

This work was supported by the National Science Foundation grants 2236198, 2247929, and 2303042. Also, CAPES: 88887.899310/2023-00, 88887.900069/2023-00 and 88887.915794/2023-00. CNPq: 140770/2021-6, 141180/2021-8, 140771/2021-2, 315711/2020-5, 141276/2020-7 and 141054/2019-0. CAPES/PROCAD: 175956. CAPES/PROEX: 88887.373933/2019-00. FAPERJ: 200.510/2023, 211.033/2019 and 202.621/2019. Finally, we thank the Mozilla `PDF.js` maintainer for analyzing the answers and giving their feedback.

A Comparison of Conversational Models and Humans in Answering Technical Questions: the Firefox Case

A version of this work was accepted in the 48th IEEE/ACM International Conference on Software Engineering (ICSE'26).

In Chapter 4, we introduced a Retrieval-Augmented Generation (RAG) assistant designed to integrate project-specific documentation and source code to provide contextually accurate and relevant responses for a smaller-scale Mozilla Firefox project. In that study, we compared the answers generated by the RAG assistant with those provided by humans, evaluating quality based on the expert satisfaction attribute.

In the current study, we extend this work by comparing the RAG assistant with two additional sources: a general-purpose GPT and human responses from the Firefox developer community. We evaluated 52 real-world technical questions, with responses assessed by Mozilla engineers across four key attributes: helpfulness, comprehensiveness, conciseness, and overall preference. The results indicate that RAG answers were more comprehensive than human answers (62.5% vs. 54.1%) while achieving comparable helpfulness (75.0% vs. 79.1%). However, they tended to be more verbose, suggesting conciseness as an area for improvement. Notably, RAG responses were preferred over GPT responses in 39.5% of cases, compared to GPT's 25.6%, underscoring the benefits of integrating project-specific knowledge.

This chapter presents the detailed findings of this comparative study, which is documented in the paper *A Comparison of Conversational Models and Humans in Answering Technical Questions: the Firefox Case* to be published in the proceedings of ICSE'26. **This study constitutes a key contribution of this Ph.D. research addressing the question: *How do RAG answers compare with a general-purpose LLM and with human answers across helpfulness, comprehensiveness, conciseness, and expert preference?***

5.1

Introduction

The integration of conversational agents into software engineering workflows is transforming how developers interact with code and documentation. Large Language Models (LLMs) tools, such as GitHub Copilot and ChatGPT, are now widely used to support code comprehension, code completion, and debugging (Dakhel et al. 2023, Sallou et al. 2023, Wang et al. 2024, Pinto et al. 2023). These tools have the potential to significantly reduce the time spent searching online documentation, waiting for human guidance, or interpreting error messages (Guo et al. 2024).

Despite their advantages, the utility of general-purpose LLMs can be limited when addressing the specific context of an individual software project (Fan et al. 2023). Models available to the public are usually trained on a broad range of data. Consequently, their responses may lack context, be outdated, incomplete, or even misleading, especially when dealing with large and actively evolving systems.

To address these limitations, Retrieval-Augmented Generation (RAG) has emerged as a promising approach. RAG architectures enrich language models with retrieval mechanisms that dynamically fetch relevant information, such as project documentation, code snippets, or archived conversations, from curated repositories before generating a response. This grounding enables the model to produce more accurate and context-sensitive answers tailored to a target project (Lewis et al. 2020, Hao et al. 2025).

Such contextualized answers, tailored to a specific task, are particularly important in Open-Source Software (OSS) projects that depend on a large set of contributors, many of whom are newcomers learning on their own (Fagerholm et al. 2014, Steinmacher *et al.* 2014), to sustain project growth. Newcomers to OSS projects often struggle with the complexity of the software, the diversity of contributions, and the lack of available mentors (Steinmacher *et al.* 2015, Steinmacher *et al.* 2021, Feng *et al.* 2022, Fagerholm et al. 2014). Although experienced contributors try to support newcomers via pull request reviews or chat interactions, such personalized support may not be sustainable as the project grows. It can divert experienced developers, already overwhelmed (Guizani et al. 2025), from their efforts to make core contributions (Feng *et al.* 2022, Steinmacher *et al.* 2021).

RAG-based assistants, while not a replacement to human mentors, are a promising solution to help newcomers overcome project-specific barriers, but implementing and maintaining scalable RAG systems is resource-intensive, requiring substantial engineering effort for data curation, indexing, and retrieval integration. While *Correia et al.* took an important first step by comparing RAG-generated answers to human responses, their study did not assess

whether such augmentation is necessary. Specifically, they did not include a baseline comparison with general-purpose LLMs (e.g., GPT). Moreover, their evaluation relied on a limited dataset, a single expert assessor, and a narrow, one-dimensional satisfaction metric. As a result, we still lack a multidimensional understanding of how general-purpose and RAG-based models differ in practice, what trade-offs they present, and whether the added engineering complexity of RAG delivers proportional benefits in large OSS ecosystems.

We conducted a comparative study that analyzed answers provided by different assistants to real technical questions within the context of the Mozilla Firefox project. We introduced a retrieval-augmented generation (RAG) approach that integrates project-specific knowledge to deliver timely, relevant support to developers. Our study compared this RAG system with two other sources: (i) GPT, a general language model assistant, and (ii) Human answers from the Firefox developer community. We evaluated answers to 52 questions, incorporating assessments from Mozilla engineers based on four key attributes: helpfulness, comprehensiveness, conciseness, and overall preference. This expert evaluation offers a more nuanced understanding of each assistant’s strengths and limitations, providing a realistic assessment of their effectiveness in practical development scenarios.

Our study reveals that *RAG*-assisted answers were more comprehensive than human responses (62.5% vs. 54.1%) while maintaining a comparable level of helpfulness (75.0% vs. 79.1%). However, *RAG* responses were generally more verbose and not as concise as human answers. Additionally, *RAG* responses were selected as the preferred option in practice more often than those from *GPT* alone (39.5% vs. 25.6%), highlighting the benefits of integrating project-specific knowledge into AI-driven developer assistance. While *RAG* responses offered more contextually relevant and informative answers, conciseness remains an area for improvement.

Thus, this paper presents the following contributions: (i) we evaluate the effectiveness of *RAG* in assisting developers within the Mozilla Firefox project, comparing it to human responses and a standard *GPT* model; (ii) we analyze the trade-offs between helpfulness, comprehensiveness, and conciseness in human, *GPT*, and *RAG* responses, demonstrating that *RAG* provides more detailed and informative answers while maintaining high helpfulness scores; (iii) we identify the strengths and limitations of *RAG*-based approaches for developer assistance, showing that while *RAG* enhances comprehensiveness, response verbosity remains a challenge, and; (iv) we provide insights into the practical application of AI-assisted developer support, emphasizing how *RAG*-based models can alleviate the burden on maintainers in large-scale OSS

projects.

5.2 Related Work

Recent research has explored how large language models can assist with software development tasks and help new developers onboard on existing projects.

5.2.1 LLMs in Software Engineering

LLM tools have become increasingly integrated into software development workflows, with recent empirical studies revealing diverse application patterns beyond simple code generation (Mathews et al. 2024). Khojah et al. (Khojah et al. 2024) investigated how professional engineers utilize ChatGPT during work tasks, finding that developers primarily leverage LLMs for conceptual guidance rather than direct code generation, although concerns about hallucinations and legal limitations affect their adoption. Nam et al. (Nam et al. 2024) demonstrated that LLM-based code understanding tools, when integrated directly into IDEs, can significantly improve task completion rates compared to traditional web search, with benefits varying by experience level. Recent advances have addressed repository-level challenges, with CodePlan (Bairi et al. 2024) introducing planning algorithms that enable LLMs to perform complex multi-file edits, and ClarifyGPT (Mu et al. 2024) improving code generation through interactive requirement clarification. Studies of human-AI interaction patterns (Nguyen et al. 2024) show that beginning programmers struggle with the full prompt-and-edit cycle even for problems at their skill level.

5.2.2 Supporting Newcomers to Software Projects

Research on newcomer support has evolved from identifying barriers (Steinmacher *et al.* 2015, Steinmacher *et al.* 2014) to developing technological interventions that address both technical and social integration challenges. Steinmacher et al. (Steinmacher *et al.* 2016) developed an information portal that provides newcomers with project-specific guidance, finding that structured information delivery significantly reduces initial barriers to contribution. Traditional support mechanisms have included mentoring programs (Canfora et al. 2012, Steinmacher *et al.* 2021, Feng et al. 2025), documentation improvements (Treude et al. 2011, Franchetti *et al.* 2023), and social integration

strategies (Dagenais et al. 2010) that emphasize human-to-human knowledge transfer. Early technological approaches focused on recommendation systems for finding suitable tasks (Xiao et al. 2022, Santos et al. 2021) and potential mentors (Steinmacher et al. 2012). However, these approaches often struggle with scalability limitations, outdated information, interaction barriers, and the substantial effort required from experienced team members.

Recent work has also explored LLM applications for newcomer support, with Azanza et al. (Azanza et al. 2024) emphasizing the need for self-service resources that reduce mentor workload, while Balfroid et al. (Balfroid et al. 2024) found that automated code tour generation shows promise but suffers from quality issues, including redundancy and overly detailed explanations. Richards et al. (Richards et al. 2024) proposed ToMMY, a code understanding assistant that leverages Theory of Mind (ToM) to infer the mental state of a developer and personalize responses. Emerging AI-assisted approaches (Adejumo et al. 2024) propose using LLMs to address documentation complexity and information overload. However, delivering accurate, contextually relevant assistance at scale remains challenging for off-the-shelf LLMs, which often lack access to up-to-date or project-specific information and may produce generic or hallucinated responses.

5.2.3

Improving LLM Performance with RAG

Retrieval-Augmented Generation (RAG) is a promising approach to enhance LLM effectiveness with application in different domains, such as customer support (Xu et al. 2024) and software engineering (Ibtasham et al. 2025, Finsås et al. 2024). RAG combines the strengths of information retrieval and language generation by allowing LLMs to access and incorporate external sources, such as project documentation or source code, thereby grounding their responses in accurate, question-relevant content rather than relying solely on pre-trained knowledge.

Recent research demonstrates RAG’s effectiveness in software engineering contexts. Ibtasham et al. (Ibtasham et al. 2025) investigated the implementation of RAG in software release management through ReqRAG, a specialized chatbot. Their industry-based study revealed that RAG-based responses achieved 70% relevance and usefulness ratings. Similarly, Finsås and Maksim (Finsås et al. 2024) explored RAG for technical support through relevance feedback and multi-agent interactions. Their research demonstrated that incorporating relevance feedback mechanisms significantly improved response accuracy. However, the potential for RAG to facilitate newcomer onboarding

in software projects remains largely unexplored.

To the best of our knowledge, *Correia et al.* is the only previous work to systematically compare RAG-enhanced LLM responses with those provided by experienced developers for newcomer questions in OSS. Their study introduced DevMentorAI, evaluated using 14 technical questions sourced from a small OSS project, PDF.js. One expert conducted a blind assessment, comparing human-generated and RAG-generated responses. DevMentorAI’s answers were considered more satisfactory in 8 cases, equally satisfactory in 3, and less satisfactory in 3. While their work evidences the potential of using RAG-enhanced LLMs for supporting newcomers to OSS, its evaluation was limited, and the assessments were relatively small-scale. Besides, the RAG system used in *Correia et al.* was developed in-house, presenting a significant technical barrier to adoption across different projects. Our study significantly extends *Correia et al.*’s work in multiple aspects: (i) We provide a more refined understanding of the usefulness of different answers, with four evaluation attributes that are critical to answers to developer questions (Kou et al. 2023)—helpfulness, comprehensiveness, conciseness, and overall preference—providing a deeper understanding of each approach’s strengths and weaknesses, whereas *Correia et al.* relied on a single binary generic criterion. Our evaluation methodology includes both individual assessment of standalone attributes and comparative analysis, providing a more robust assessment of answer quality. (ii) By including GPT, a readily available model, we better isolate RAG’s contribution through direct comparison of RAG-enhanced LLM, general-purpose LLM, and human responses. This three-way comparison enables more precise identification of RAG’s unique advantages for technical developer inquiries. (iii) We expanded the scale and methodological rigor by involving 8 experienced engineers rather than a single evaluator, assessed 52 real development questions versus 14, conducted agreement and consistency analyses to mitigate individual biases, and our target project, Firefox, represents a larger, more widely-used open source project compared to PDF.js’s smaller repository and knowledge base. (iv) We adapted Cognita, an open source RAG framework, contrasting with the in-house RAG solution employed in the prior study, thereby ensuring greater reproducibility and applicability. The open source framework was modified to support the ingestion of diverse artifacts from a large-scale project and include a more comprehensive set of knowledge sources across multiple formats. In summary, our research provides a more comprehensive and methodologically rigorous comparison of human-generated, GPT-generated, and RAG-enhanced GPT-generated answers. By leveraging project-specific artifacts through an open source RAG framework, engaging multiple expert evaluators, and significantly

expanding the question set, we offer deeper insights into the practical effectiveness of these systems for supporting newcomer developers with technical questions.

5.3 Study Design

This section outlines the research questions and the methodology employed to address them. Our target project is Mozilla Firefox, a large-scale open source project. We collected technical questions from three Firefox developers' chat rooms hosted on Matrix (Matrix). We also collected the human answers for these questions directly in these chat rooms. We generated answers to the same questions using two automated approaches: GPT-4o (*GPT*) alone and GPT-4o enhanced with (*RAG*)—leveraging relevant Mozilla Firefox project artifacts as contextual information. Then, Mozilla Firefox engineers assessed the quality of the answers. Their feedback was analyzed quantitatively and qualitatively. In this section, we further detail each of the steps of our study.

5.3.1 Research Questions

Our study is structured around the following research questions:

RQ₁: How do experts perceive answers from Humans, GPT, and RAG provided to real technical questions?

We hypothesize that the perceived quality of answers—measured by helpfulness, comprehensiveness, and conciseness (Kou et al. 2023)—will differ due to the inherent characteristics of each source. Firstly, Human responses are expected to embed nuanced, context-specific insights drawn from real-world experience, while *GPT* responses rely on general pre-trained knowledge that may not fully capture project-specific details. In contrast, *RAG* answers by incorporating retrieval mechanisms, could offer a more context-aware perspective. Concluding that these differences significantly impact expert perceptions may imply that AI-based approaches require tailored enhancements to match the practical value of human-provided assistance. In this RQ, we also aim to understand whether experts' evaluations vary not only in overall quality but also in the specific attributes of helpfulness, comprehensiveness, and conciseness, building on previous findings that underscore the importance of contextual relevance in technical communication.

RQ₂: How do helpfulness, comprehensiveness, and conciseness influence experts' preferences for answers?

Given that the effectiveness of a technical answer is often determined by its specific attributes, this research question is driven by the need to understand which characteristics most strongly influence expert preferences. We seek to systematically classify the attributes of responses—specifically helpfulness, comprehensiveness, and conciseness—provided by human developers, *GPT*, and *RAG*. In doing so, we aim to determine whether differences in these attributes, such as the depth of context or the brevity of explanations, impact the overall quality of an answer. Additionally, we intend to comprehend how these attributes interplay and whether certain strengths in one attribute might compensate for weaknesses in another. By answering this research question, we aim to enrich our understanding of how the intrinsic qualities of technical answers affect their practical applicability in real-world software development, ultimately guiding improvements in AI-assisted support tools.

RQ₃: Which are the positive and negative aspects of answers from Humans, GPT, and RAG?

Since technical answers can exhibit a range of qualitative strengths and weaknesses, this research question is driven by the need to understand the positive and negative aspects of responses provided by human experts, *GPT*, and *RAG*. We seek to conduct a systematic qualitative analysis to classify the benefits and drawbacks of each approach. In particular, we aim to identify aspects such as clarity, contextual relevance, and technical accuracy as positives, while also exploring limitations like verbosity, potential inaccuracies, and a lack of nuanced understanding as negatives. However, it is important to note that qualitative analysis inherently cannot capture every subtle facet of answer quality—such as the evolving perceptions of individual experts or the contextual dependencies that emerge in real-world interactions. By examining these qualitative dimensions, we aspire to provide actionable insights into how each type of response supports or hinders effective communication in technical discussions, ultimately informing the refinement of AI-based assistance tools and guiding future research to enhance their practical usability in software development environments.

5.3.2

Process for Data Collection

5.3.2.1

Step 1: Data Collection

All the data analyzed is publicly available and was downloaded using the platform feature. The Mozilla community utilizes Matrix.org for communication and question answering, which motivated our choice to analyze

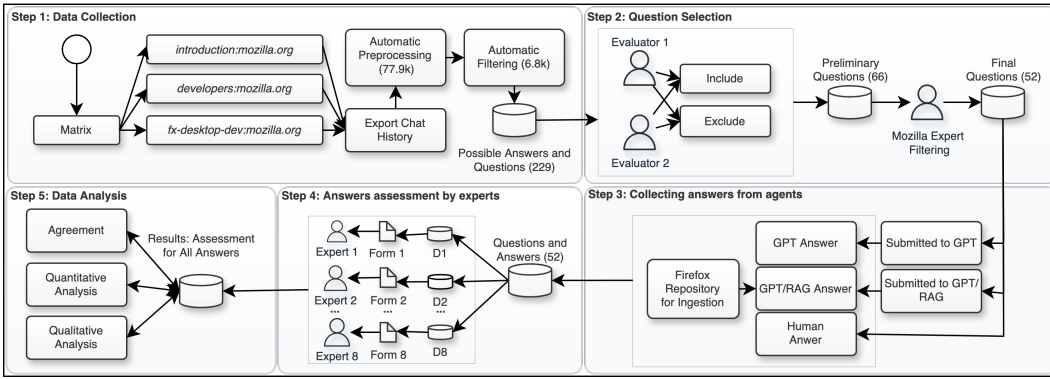


Figure 5.1: Study design (The numbers in parentheses refer to the volume of data in each stage of the method).

chats hosted on this platform. To begin our data collection, we explored all Mozilla-related public channels available on Matrix.org. Subsequently, one of the authors—a Mozilla Firefox developer—selected key channels explicitly focused on Firefox. Ultimately, we collected data from three specific Mozilla Matrix.org channels: *introduction:mozilla.org*, *developers:mozilla.org*, and *fx-desktop-dev:mozilla.org*.

For each selected channel, we exported a raw file containing the complete chat history, including messages, events, and attachments. The chat histories were exported in *JSON* format using the platform’s *Export Chat History* feature. Considering the fragmented nature of the chat conversations and the difficulty in identifying questions and answers, we applied preprocessing using Python over each chat history.

The preprocessing involved the following steps:

1. *Message Aggregation*: Sequential messages from the same author were merged into a single entry, preserving the timestamp of the first message.
2. *Text Normalization*: Replacement of double quotes with single quotes to maintain consistency, removal of excessive whitespace, standardization of line breaks, and conversion to UTF-8 encoding to handle special characters
3. *Message Categorization*: Regular expressions were used to classify each message into one of four categories.
 - **Reply**. Messages containing at least one line beginning with ">" indicating an explicit reference to a previous message
 - **Question**. Messages containing one or more question marks (?) signaling interrogative intent
 - **Reply to Question**. Messages where a line begins with ">" and terminates with "?" indicating direct responses to earlier questions

- **General Purpose.** Messages not matching any of the above patterns or consisting solely of whitespace

4. *Manual Inspections and Adjustments:* Manual inspection and corrections were performed to ensure data quality and accuracy of the preprocessing results.

5.3.2.2

Step 2: Question Filtering

We compiled the processed messages into a structured spreadsheet containing Message Type, Timestamp, Author, and Message Content, resulting in approximately 77,900 messages. We then applied temporal filtering to focus on messages from 2024 (the year in which we conducted the study), reducing the dataset to 6,895 messages. Next, we filtered for messages categorized as *Reply to Question*, which yielded 229 messages for 2024. These messages were manually reviewed to ensure they contained both a clear question and a corresponding human answer, with no exclusions made during this review process. This final dataset of 229 question-answer pairs represents 3.3% of all 2024 messages and 10% of all *Reply to Question* messages in the complete dataset.

Expert Filtering. After compiling the initial set of possible *Reply to Questions* messages, individually, two evaluators (*Evaluator 1* and *Evaluator 2* in Figure 5.1) manually reviewed the messages to determine which ones would be included or excluded from the study. The evaluators excluded questions that had one or more of the following characteristics: unrelated to Firefox; not addressing source code, documentation, features, or contributions; addressing follow-up of another question; lacking sufficient context for meaningful evaluation of experts or AI models; time-sensitive questions (e.g., version-specific queries); messages that were not questions.

After the individual assessments, the evaluators engaged in an in-person discussion to reach a consensus and resolve differences regarding 15 questions. A question was only included in the study dataset if both evaluators agreed to it. Ultimately, we generated a total of 66 preliminary questions. After this process, a developer from the Firefox project conducted a final review, leading to a refined set of *Final Questions* (52) to be used in the study (22.7% of the initial dataset of 229 questions). The expert provided the following reasons for excluding additional questions: questions with answers that redirect the user to another channel or user, remaining questions unrelated to Firefox that

the first two evaluators had not identified, and questions that were misleading or irrelevant. The questions and the exclusion criteria are available in the supplementary material.

5.3.2.3

Step 3: Collecting Answers from Models

Our goal is to compare responses from three sources: original human answers (Human), GPT-4o-generated responses (GPT), and GPT-4o responses enhanced with Retrieval-Augmented Generation (RAG). Human answers were obtained directly from chat messages, while GPT and RAG responses were generated using the prompt specification described below. This prompt was refined through initial testing; early GPT responses tended to be overly detailed and verbose, which is unsuitable for the concise nature of chat environments. To address this, we added a directive limiting responses to a single paragraph, ensuring consistency and practical applicability across all three answer types.

LLM Prompt to Collect the Answers

The question below was taken from a Mozilla Firefox developer's chat. Answer the question directly and in one paragraph, as long replies may disturb the chat.
Question: <question>.

To collect answers using RAG, we utilized an open source Framework named Cognita (Cognita). Initially, we adapted and extended the embedding components within Cognita to support the ingestion of file formats that were originally unsupported, such as source code files. These improvements allowed us to successfully ingest technical documentation and source code from our target project, i.e., **Firefox**. All artifacts were ingested exclusively from the publicly available Gecko-Dev GitHub repository (<https://github.com/mozilla/gecko-dev>). Notably, our modified component supports the ingestion of source code in multiple programming languages and markup formats. Specifically, we ingested a variety of artifacts, including source code, wiki pages, and web documents in formats such as *js*, *jsx*, *mjs*, *cjs*, *md*, *html*, *pdf*, among others. To verify the effectiveness of the artifact ingestion process, we ran a set of representative queries covering key Firefox development topics and manually inspected the top retrieved chunks to confirm that the system consistently returned semantically relevant and accurate information from

various known artifact types (e.g., source code, documentation, wiki pages). All the tests demonstrated that the ingestion and retrieval components reliably grounded the RAG model in project-specific knowledge.

Using the Cognita Framework for RAG purposes, we enhanced the previously mentioned prompt by incorporating a specific context section. This section includes relevant information retrieved from project artifacts based on the semantic similarity between the user’s question and the ingested content, which includes source code comments, documentation, and wiki pages (sources suggested by the Firefox developer). The framework embeds the user query and searches for the most semantically similar chunks using a vector database, which are then appended to the prompt to support context-aware generation. We submitted queries manually to both GPT and RAG using their web interfaces, and the resulting responses were systematically organized into spreadsheets for analysis. A complete list of the RAG’s answers, along with the contextual data retrieved for each, is available in our supplementary material (Correia et al. 2025). Below, we summarize the approaches we used to collect answers from the different sources.

- **Human answer:** extracted directly from the chat history.
- **GPT answer:** extracted using the large language model (GPT-4o) through the web platform chat.openai.com.
- **RAG answer:** extracted using an extended version of the Cognita Framework, through a locally deployed web platform augmented with artifacts from the Firefox repository.

The primary cost stemmed from generating embeddings for the large Firefox codebase, which amounted to around USD 28 and was completed in approximately 24 hours. The study required only two model inferences per question, and each inference took less than 5 seconds to execute. The total cost for inferences was USD 3.

5.3.2.4

Step 4: Expert Assessment of Answers

A panel of eight Mozilla engineers (referred to as *Expert 1* through *Expert 8* in Figure 5.1) evaluated the answers. Each expert received a form with 10 questions with corresponding answers generated by *GPT*, *RAG*, and *Humans*. From these, we randomly assigned six uniquely ones to each expert. The remaining subset of four questions was assigned to all experts. We intentionally included the overlapped evaluations to measure agreement among the eight experts during their assessments.

Before the evaluation, the experts received detailed instructions outlining the evaluation procedure. Specifically, they received clear definitions of the criteria used to assess the sets of answers given to the technical questions. For each question, experts reviewed multiple alternative answers and evaluated them according to three primary attributes:

- **Helpfulness:** The degree to which the answer provides meaningful assistance in addressing the question.
- **Comprehensiveness:** The extent to which the answer includes all essential elements required to respond thoroughly to the question.
- **Conciseness:** The quality of conveying all necessary information succinctly, without including unnecessary or redundant words.

The evaluators followed a structured method to assess each question, allowing for a thorough and unbiased comparison of multiple answers. To reduce potential bias, all questions and answers were anonymized and presented in a random order. The assessment process was divided into three main phases, as detailed in Table 5.1.

In the first phase, each answer was evaluated individually on a binary scale based on three predefined attributes (e.g., helpful or not helpful). This scale was defined in consultation with a Mozilla manager to reduce workload and allow for a more comprehensive evaluation. This approach ensured that answers were evaluated on their own merits before any direct comparisons were made. While a Likert scale could have offered finer-grained feedback, the binary method allowed for clear distinctions among sources and was supplemented by comparative selections in the next phase (e.g., identifying "the most helpful" answer). This supported both absolute and relative quality assessments without adding unnecessary complexity.

Table 5.1: Illustrative form for the assessment of a question and its answers.

Question X:	<i><Question content>.</i>
	<i>Multiple Choice</i>
Answer 1: <i><Answer content></i>	<input type="checkbox"/> Helpful <input type="checkbox"/> Comprehensive <input type="checkbox"/> Concise
Answer 2: <i><Answer content></i>	<input type="checkbox"/> Helpful <input type="checkbox"/> Comprehensive <input type="checkbox"/> Concise
Answer 3: <i><Answer content></i>	<input type="checkbox"/> Helpful <input type="checkbox"/> Comprehensive <input type="checkbox"/> Concise
	<i>Single Choice</i>
Most helpful information:	<input type="checkbox"/> Answer 1 <input type="checkbox"/> Answer 2 <input type="checkbox"/> Answer 3
Most comprehensive information:	<input type="checkbox"/> Answer 1 <input type="checkbox"/> Answer 2 <input type="checkbox"/> Answer 3
Most concise without being incomplete:	<input type="checkbox"/> Answer 1 <input type="checkbox"/> Answer 2 <input type="checkbox"/> Answer 3
Preferred answer in practice:	<input type="checkbox"/> Answer 1 <input type="checkbox"/> Answer 2 <input type="checkbox"/> Answer 3
Open feedback	Please highlight any particularly relevant or irrelevant aspects of the preferred answer(s), or note any missing elements.

In the second phase, the evaluators selected which answer stood out as the most helpful, comprehensive, and concise. It allowed evaluators to compare the three answers provided to each question. In addition, they indicated which answer from the three alternatives they would prefer to see in practice, i.e., which answer they would most likely rely on in a real-world scenario.

In the third phase, we asked evaluators to provide feedback through an open question. This gave them the chance to highlight particularly strong or weak aspects of the answers, suggest improvements, and note any missing elements that could enhance the answers. This evaluation and open feedback aimed to capture objective judgments and subjective insights from experienced professionals.

Each evaluator followed the same process described above for the ten questions. In the end, they were also asked to provide demographic information, which is presented in Table 5.2. The 1st column shows the evaluator’s ID; the 2nd indicates their confidence level in the assessment, measured on a Likert scale; the 3rd reports their years of contribution to Mozilla, also on a Likert scale; the 4th refers to their gender, an open and optional field; and the 5th column shows their self-rated proficiency in answering questions in Matrix channels, offering context on their familiarity with developer support practices.

Table 5.2: Mozilla experts’ demographics.

ID	Confidence	Experience	Contributing to Mozilla	Gender	Q&A Proficiency
E1	3	10+ years	10+ years	Man	5
E2	4	4-7 years	4-7 years	–	5
E3	3	10+ years	1-3 years	Man	3
E4	3	10+ years	1-3 years	–	4
E5	4	10+ years	10+ years	Woman	5
E6	5	8-10 years	4-7 years	Man	5
E7	3	10+ years	10+ years	–	3
E8	4	10+ years	8-10 years	Man	5

5.4 Data Analysis

As described in Section 5.3.2, experts evaluated responses from *Humans*, *GPT*, and *RAG* based on three primary attributes: helpfulness, comprehensiveness, and conciseness. Each expert assessed a subset of questions, with four questions evaluated by all experts to allow inter-rater agreement analysis. *Fleiss’ Kappa* was used to measure agreement among the eight experts on these common questions, yielding kappa values of 0.67 (Substantial), 0.69 (Substantial), 0.31 (Fair), and 0.45 (Moderate). These findings indicate an

overall acceptable level of agreement while also revealing variability that may stem from differences in question clarity or expert interpretation. See complete data in our supplementary material (Correia et al. 2025).

To compare expert perceptions of responses from the three sources, we computed the proportion of answers exhibiting each attribute. Additionally, for each question, we identified the answer that best represented each attribute (i.e., the most helpful, the most comprehensive, and the most concise). To determine statistical significance in these variations, we performed pairwise comparisons using Fisher’s Exact Test, evaluating whether differences between sources were significant. The contingency tables used in this analysis are presented in Table 5.3. These tables provide the frequency distributions of assigned attributes across answer sources used for statistical comparisons. This analysis directly addresses RQ_1 by quantifying how experts assess answer quality across sources and identifying notable differences in response attributes.

Table 5.3: Contingency tables for Fisher’s test.

Helpfulness			Comprehensiveness			Conciseness		
	Human	GPT		Human	GPT		Human	GPT
Yes	38	26	Yes	26	19	Yes	12	14
No	10	22	No	22	29	No	36	34
Total	48		Total	48		Total	48	
	Human	RAG		Human	RAG		Human	RAG
Yes	38	36	Yes	26	30	Yes	12	11
No	10	12	No	22	18	No	36	37
Total	48		Total	48		Total	48	
	GPT	RAG		GPT	RAG		GPT	RAG
Yes	26	36	Yes	19	30	Yes	14	11
No	22	12	No	29	18	No	34	37
Total	48		Total	48		Total	48	

Each assessed question yielded four key indicators: the most helpful answer, the most comprehensive answer, the most concise answer, and the answer the expert would prefer in practice. To analyze the relationship between these indicators and the experts’ preferences, we conducted a non-parametric correlation analysis using Spearman’s rank correlation. This method was selected due to the ordinal nature of our data and because it does not assume normal distribution. Spearman’s correlation is suitable for understanding whether higher scores on individual attributes are associated with increased likelihood of an answer being preferred in practice, helping to address RQ_2 . This allowed us to quantify the relative importance of these attributes in real-world usage scenarios, revealing whether experts prioritize detailed and informative responses over concise but potentially less informative answers.

Additionally, experts provided qualitative feedback on the responses, offering insights into both the strengths and weaknesses of each source. Three researchers manually analyzed this feedback, using an open coding approach. After they reached consensus on the codes, they applied thematic analysis to identify recurring patterns and themes. Codes were first generated inductively from the data and then grouped into broader categories that reflected strengths and weaknesses in each response source. Through this process, we identified recurring patterns in how each answer source performed under different conditions, including cases where human responses outperformed the models due to domain-specific expertise, and cases where RAG responses excelled in effectively integrating documentation. Similarly, we noted instances where *GPT* responses, despite being concise, lacked necessary depth, leading to lower expert preference.

These qualitative insights form the basis of our assessment for *RQ₃*, helping us contextualize the statistical findings with real-world expert opinions. By combining both quantitative and qualitative analyses, we provide a holistic evaluation of how different answer sources contribute to technical discussions, ultimately guiding future improvements in AI-assisted developer support.

5.5 Results and Discussions

In this section, we present the results of our study, which aimed to explore answers to real questions related to the Mozilla Firefox project, comparing answers provided by *Humans*, *GPT*, and a *RAG*. We evaluated the answers with the help of Mozilla Firefox engineers, who evaluated each response based on its helpfulness, comprehensiveness, and conciseness. In addition, the evaluators determined the answer that they thought best revealed each of these attributes and picked their preferred answer.

5.5.1 Overall Performance

In Table 5.4, we present the percentage of each attribute assessed per source. The 1st column lists the attributes; the 2nd, the answer source; the 3rd, the percentage of questions where the attribute was assigned; and the 4th, where that source was considered *the most* for that attribute (e.g., *the most comprehensive*). Additionally, Table 5.5 shows the percentage of questions in which each source was preferred for practical use.

Table 5.4: Assigned Attributes and Most Preferred Answer per Attribute.

Attribute	Answer Source	Assigned (% Questions)	The Best (% Questions)
Helpfulness	Human	79.1%	30.2%
	GPT	54.1%	23.2%
	RAG	75.0%	46.5%
	<i>Total</i>	—	100.0%
Comprehensiveness	Human	54.1%	20.9%
	GPT	39.5%	23.2%
	RAG	62.5%	55.8%
	<i>Total</i>	—	100.0%
Conciseness	Human	25.0%	33.3%
	GPT	29.1%	35.7%
	RAG	22.9%	30.9%
	<i>Total</i>	—	100.0%

Table 5.5: Percentage of Answers Preferred to See in Practice.

	Human	GPT	RAG	Total
Preferred to See in Practice (% Questions)	34.8%	25.5%	39.5%	100.0%

5.5.1.1 Helpfulness

Human answers were assigned this attribute in 79.1% of the evaluated questions. Closely following were the *RAG* answers with 75.0% deemed helpful, and *GPT* was rated helpful in only 54.1% of the cases. When comparing which source produced the **most helpful** answer, *Human* responses were selected in only 30.2% of the cases, whereas *RAG* was chosen in 46.5%, and *GPT* in 23.2%. This contrast suggests that, while *Human* responses were usually considered helpful, they did not always provide the best assistance, with *RAG* offering actionable responses, and standing out as the most helpful in nearly half of the cases (46.5%).

Figure 5.5 provides visual support for qualitative insights into the distribution of helpful, comprehensive, and concise answers across the different sources. Focusing on helpfulness, Figure 5.2 illustrates that for 22 questions, all three sources provided answers that were considered helpful. *Human* answers account for the highest number of uniquely helpful answers for questions, with 5 exclusive cases, against 2 exclusive cases to *RAG* and 1 to *GPT*. The nearest source to the *Human* (in numbers) is the *RAG*, showing 10 answers for questions deemed as helpful. In contrast, *Human* and *GPT* have only one question where answers from both sources were deemed helpful. It is important

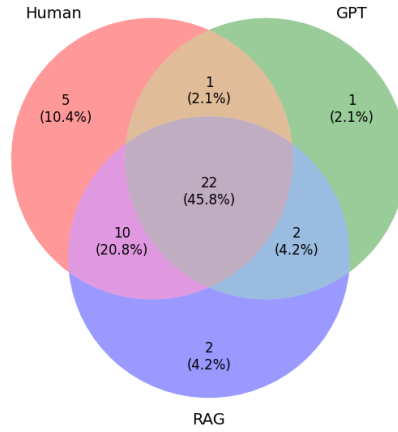


Figure 5.2: Helpful answers

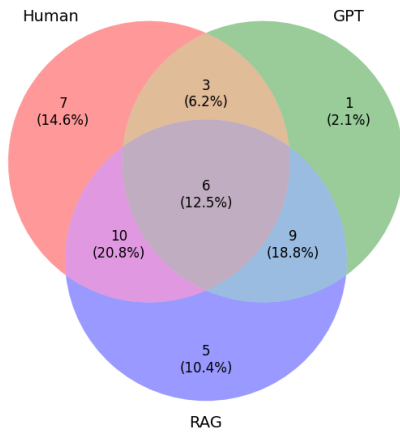


Figure 5.3: Comprehensive answers

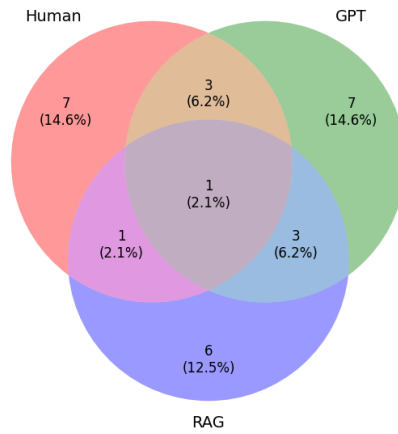


Figure 5.4: Concise answers

Figure 5.5: Number of answers and the occurrence of attributes.

to note that, despite the high number of helpful answers attributed to *Human*, *RAG* was the source with more answers rated as *the most helpful* by engineers, reinforcing the impact of retrieval augmentation in generating valuable responses.

Moreover, we observe that *RAG* provided helpful answers to only two questions that humans themselves did not provide helpful answers to, whereas *GPT* managed this for only one. This result suggests that, despite *RAG*'s tendency to perform better than *GPT* in general in terms of helpfulness, the challenge of surpassing humans in this aspect remains for both approaches.

5.5.1.2 Comprehensiveness

Concerning **comprehensiveness**, *RAG* answers were scored as comprehensive for 62.5% of questions. In contrast, *Human* and *GPT* answers were rated as comprehensive in 54.1% and 39.5% of questions, respectively. When

assessing which source provided the **most comprehensive** response, *RAG* was chosen in 55.8% of cases, followed by *GPT* answers in 23.2% and *Human* answers in 20.9%. These results may suggest the *RAG*'s capacity to retrieve relevant artifacts to construct responses with depth and completeness. In contrast, *GPT* answers without direct access to external references, relying solely on pre-trained knowledge, may have resulted in less detailed responses than *RAG*. Likewise, *Human* answers may have been constrained by time limitations or an assumption that the recipient had some background knowledge, leading to the lowest selection rate for the most comprehensive.

Figure 5.3 highlights the heterogeneity of sources considered as having comprehensive answers. *Human* was the source with more (7 unique cases) comprehensive answers; this may point to the human synthesis capacity. On the other hand, *RAG* follows closely with 5 unique cases. The overlap between *Human* and *RAG* (10 cases) suggests that this source can better approximate *Human* reasoning, in contrast to the overlap of *RAG* and *Human* (3 cases). The *RAG*'s performance (5 unique cases) contributes to the view that LLMs can answer technical questions in OSS. However, it is important to mention that *GPT* alone was comprehensive in only one answer, underscoring its limitations by relying solely on its pre-trained knowledge. The 6 cases in which all sources were equally comprehensive suggest that some questions inherently allow for completeness across different answering methods, at least for well-known software like Mozilla Firefox.

5.5.1.3 Conciseness

Conciseness is critical in real-time developer discussions, where long answers may lead to a waste of effort. *GPT* answers were considered concise in 29.1% of questions, while *Human* and *RAG* answers were rated concise in 25.0% and 22.9% of questions, respectively. Interestingly, this attribute was the most balanced between the three sources. When selecting the **most concise** answer, *GPT* was chosen in (35.7%) of the questions, followed by *Human* in (33.3%) and *RAG* in (30.9%). These results may suggest that no source was superior in terms of conciseness. This may suggest that, while participants generally did not find answers concise, they still chose the least verbose option. In this context, *GPT* could be perceived as slightly better than *Human* and the *RAG*, tending to generate direct answers that engineers found to be more concise without losing essential details.

Figure 5.4 shows the distribution of conciseness among the answer sources. As described in the last paragraph, conciseness demonstrates a bal-

anced distribution. This suggests that conciseness is not uniformly achievable across different answers for the same question. Notably, *Human* and *GPT* each had 7 unique, concise answers, indicating that both human expertise and language models can independently produce succinct answers. *RAG*, on the other hand, had 6 concise responses, maybe due to the amount of provided information, implying that retrieval augmentation does not necessarily lead to increased conciseness. In addition, there were 3 overlapping cases between *GPT* and *Human*, and 3 between *GPT* and *RAG*, highlighting instances in which conciseness emerged across different methods. Interestingly, in only 1 question, answers from all sources were rated as concise, reinforcing that achieving conciseness is highly dependent on the question and context.

5.5.1.4 Practice

Ultimately, the evaluators were asked to select the answer they would prefer to see in practice. *RAG* answers were preferred in 39.5%, followed closely by *Human* answers at 34.8%, and *GPT* alone at 25.8%. These percentages are relatively close, suggesting no single approach was overwhelmingly favored. However, it is worth noting that *RAG* had a nearly 15% and 5% over *GPT* and *Human*, respectively. Furthermore, it may suggest a preference for responses enriched with retrieval-augmented information and the need to better understand the factors influencing preferences. While *Human* answers were often assigned as helpful and comprehensive, *RAG*'s ability to integrate specific knowledge might have made it a more favorable option in some cases. The fact that human answers were frequently helpful but not always preferred to be seen in practice may suggest that evaluators valued structured, evidence-backed responses over informal or experience-based explanations. The lower preference for *GPT* might indicate that even concise responses could lack the necessary depth required for real-world applications without additional retrieval mechanisms.

5.5.2 Statistical Analysis of Attributes

We conducted two-sided *Fisher's Exact Tests* to statistically analyze the differences observed among the sources based on the contingency tables (See Table 5.3). In Table 5.6, we summarize the results from the statistical tests, including *p-values* and *odds ratios (OR)*. The *OR* quantifies the strength of association between two groups and an attribute. An *OR* greater than 1 indicates that the first group is more likely to exhibit the attribute, whereas an *OR* less than 1 suggests that the second group is more likely to exhibit it.

Table 5.6: Fisher’s test results per answer and attribute.

Comparison	p-value	Odds Ratio	Significance (< 0.05)
Helpfulness			
Human vs GPT	0.01656	3.215	Yes
Human vs RAG	0.80858	1.267	No
GPT vs RAG	0.05401	0.394	No
Comprehensiveness			
Human vs GPT	0.21959	1.804	No
Human vs RAG	0.53483	0.709	No
GPT vs RAG	0.04063	0.393	Yes
Conciseness			
Human vs GPT	0.81868	0.810	No
Human vs RAG	1.00000	1.121	No
GPT vs RAG	0.64240	1.385	No

5.5.2.1

Helpfulness

The results indicate a significant difference between *Human* and *GPT* responses regarding the helpfulness ($p = 0.016$, $OR = 3.215$), suggesting that *Human* answers were significantly more likely to be rated as helpful. However, no significant differences were observed between *Human* vs. *RAG* ($p = 0.808$, $OR = 1.267$), indicating that retrieval augmentation did not meaningfully affect helpfulness compared to human responses.

The comparison between *GPT* and *RAG* ($p = 0.054$, $OR = 0.394$) is a borderline case. The p-value is slightly above the conventional significance threshold of 0.05, meaning the evidence is not strong enough to reject the null hypothesis, but it suggests a possible trend. The OR indicates that *RAG* responses were more likely to be helpful compared to *GPT*. While this result is not statistically significant under a strict threshold, it does hint at a potential advantage of *RAG* answers that could become clearer with a larger sample size.

5.5.2.2

Comprehensiveness

There were no significant differences in comprehensiveness ratings between *Human* vs. *GPT* ($p = 0.219$, $OR = 1.804$) or *Human* vs. *RAG* ($p = 0.534$, $OR = 0.709$). However, a significant difference was observed between *GPT* and *RAG* ($p = 0.040$, $OR = 0.393$), indicating that *RAG* responses were significantly more likely to be comprehensive compared to *GPT* alone.

5.5.2.3

Conciseness

No significant differences were observed in conciseness across the tests, as shown by the *p-value* and OR. It is important to note that not finding a difference does not mean that the responses are equivalent in terms of conciseness.

5.5.2.4

Findings and Discussions

The study findings demonstrate that *RAG* notably enhances helpfulness and comprehensiveness compared to standard *GPT*, and performs comparably to *Human* answers across all evaluated attributes. No statistically significant differences were observed between *RAG* and *Human* answers in terms of helpfulness, comprehensiveness, or conciseness. This suggests that *RAG*-generated responses can match the overall quality of *Human* answers in developer support contexts.

5.5.3

Attributes Influencing Practical Preference

The results described in previous sections provide insights concerning the proportions of answers that experts perceive as helpful, comprehensive, and concise. However, we need to understand the characteristics influencing certain answers to be preferred in real-world scenarios. Thus, we conducted a correlation analysis to determine possible patterns. Figure 5.6 presents the correlation coefficients between the evaluated attributes and the selection of responses as preferred to see in practice.

The analysis indicates that *helpfulness* shows the strongest correlation (0.84) with *See in Practice*, pointing out that responses considered helpful are more likely to be perceived as applicable in real-world scenarios. This finding highlights the implication of providing helpful answers to address the problem effectively.

Next, *comprehensiveness* demonstrates a notable correlation (0.76) with the preference to *See in Practice*, indicating that answers with detailed and well-rounded explanations tend to be more valuable. The ability to integrate documentation and source code into answers probably contributes to this attribute, as it ensures that explanations are substantiated and contextually relevant.

Despite the dynamic context where the content analyzed was obtained, the *conciseness* attribute reached the weakest correlation (0.51) with *see in practice*, suggesting that while brevity is a desirable attribute, it is not a

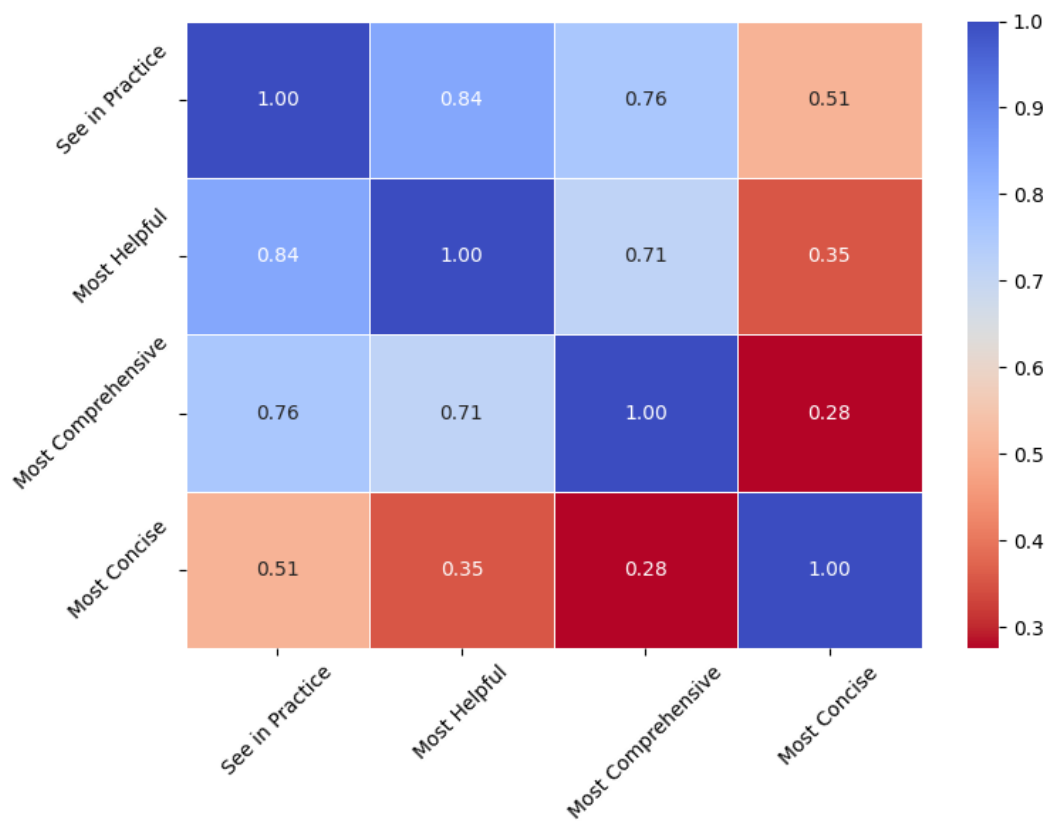


Figure 5.6: Correlation between answers attributes and their likelihood of being seen in practice.

primary concern. Concise responses may lack the necessary depth required for real-world application, potentially limiting their effectiveness despite being easier to read.

5.5.4 Evaluators' Perspectives

The evaluators provided detailed insight into the answers, highlighting strengths, weaknesses, and potential practical applications. Their assessments contribute to reaching a nuanced understanding of how *Human*, *GPT*, and *RAG* answers might perform in real-world interactions. In the following, we discuss the main findings.

5.5.4.1 Human Answers

Human answers were recognized for their correctness and practical relevance. Evaluators consistently highlighted that *Human* answers provided useful information, correctly capturing the intent of the questions and offering solutions grounded in real-world experience. Many responses included direct references to relevant resources, such as links and artifacts, which evaluators

found particularly useful (Question 1). Furthermore, we found that *Human* answers frequently selected as the best responses not only address the immediate technical question but also provide additional insights that could help developers solve related challenges.

Question 1
I would like to look at some WPT test results. How do I find a recent m-c landed version - or similar - in Treeherder?
Human's Answer: <p><code>https://wpt.fyi/results/?label=experimental&label=master&aligned</code> is likely easiest. Our results to the dashboard already come from treeherder But if you want logs, you could to to m-c in treeherder and just click on a recent job like <code>https://treeherder.mozilla.org/jobs?repo=mozilla-central&selectedTaskRun=IU6KZJ2uQ6e72CPphRZdGA.0</code>.</p>
Evaluator Comment: <p>I think that having direct links (as long as they are correct links) is always going to be more helpful than plain-word descriptions.</p>

Despite their benefits, *Human* answers sometimes lack technical depth or completeness. Some evaluators noted instances where *Human* responses assumed prior knowledge, making them less accessible, for instance, to newcomers. It is also possible that answers were written in a rush, leading to insufficient or less precise guidance. An evaluator explicitly identified this issue, saying that their own *Human* answer was rushed and that the *RAG* answer provided a more useful explanation (Question 2). This scenario emphasizes an important factor in developer interactions: while *Human* responses can be highly effective, time constraints and the informal nature of chat discussions may result in incomplete answers.

Question 2
Is there a helper function somewhere in the frontend to open a new browser window (<code>BROWSER_CHROME_URL</code>), and return a promise which resolves when <code>gBrowser</code> is available in that new window?
Human's Answer: <p><code>BrowserTestUtils</code> has some such thing but don't know the name off hand (on the phone sorry).</p>
Evaluator Comment: <p>[RAGAnswer] is more useful than [HumanAnswer] which was written on a rush (by me, lol). [GPTAnswer] is wrong, <code>OpenBrowserWindow</code> doesn't return a promise.</p>

In summary, *Human* answers seem to be appreciated by their accuracy and applicability. However, their too specific focus can sometimes result

in gaps, mainly when more detailed explanations or guidance are needed. According to our findings, one possible implication is that RAG responses—with their tendency to offer more extensive explanations—could effectively complement *Human* answers. While *Human* answers remain a gold standard, their overall effectiveness could be further enhanced by integrating more detailed support provided by RAG models.

5.5.4.2
GPT Answers

The answers generated by *GPT* generally exhibit positive attributes, particularly on conciseness, when compared to those from *RAG* and *Humans*. Evaluators noted that *GPT* answers were well-organized and easy to read, making them especially valuable for developers looking for quick explanations. For instance, when asked how to retrieve a specific version of the Firefox repository using Mercurial, *GPT* provided a clear and direct command without any unnecessary details (Question 3). Evaluators

Question 3
As someone who’s new to mercurial (coming from Git), how would I get a version of the Firefox repository, to build FF from source code, as it was (in the main/central branch) on a certain date?
<p>GPT’s Answer:</p> <p>To get a version of the Firefox repository from a specific date using Mercurial, you can use the following command: ‘hg update -d "YYYY-MM-DD"’. This will update your local repository to reflect the state of the central branch as it was on the specified date.</p>
<p>Evaluator Comment:</p> <p>[GPTAnswer] answers the question without adding the extra (probably unnecessary) information about how to clone the repo like [RAGAnswer] does. [HumanAnswer] is maybe more helpful from a general sense (if the question asker would rather use git than mercurial), but doesn’t have any details.</p>

appreciated this focused approach, especially in contrast to other answers that included additional but potentially irrelevant information. While *Human* responses sometimes offered broader insights, such as an alternative to Mercurial, *GPT* answer was often positively evaluated due to their precision and adherence.

GPT answers also revealed deficiencies. One primary concern was its tendency to deliver confident yet incorrect answers, which we can describe as plausible content with incorrect information (Question 4). This confidence in supplying answers could lead users to the wrong directions, especially in cases

where the subject matter is complex. We can also note that *GPT* can produce vague responses when a clear answer is unavailable. We might suppose that this issue could be more common in conditions without additional context, but further investigation is required.

Question 4
I was able to compile, but the long-time built Firefox is slower than the Firefox produced with artifact build? how can I enable optimization for performance on speedometer 2.1?
GPT's Answer: <p>To enable optimization for performance on Speedometer 2.1 in your Firefox build, you should ensure that you're compiling with optimization flags. Use <code>./mach build --enable-release</code> to build with optimizations for speed. Additionally, make sure that you are not using debug builds, as they are not optimized for performance.</p>
Evaluator comment: <p>The right answer is [HumanAnswer]. [GPTAnswer] just does something that doesn't exist. [RAGAnswer] almost gets it right, but doesn't <code>--enable-release</code> or anything like that. So you really want to look at the shippable mozconfigs in the tree :)</p>

GPT produces coherent responses, but its reliance on pre-trained knowledge without real-time retrieval could make it less reliable for domain-specific questions. Evaluators' comments suggest that *GPT* alone is insufficient for scenarios requiring highly accurate and project-specific guidance.

5.5.4.3 RAG Answers

RAG answers were generally well-evaluated, with the study participants frequently highlighting their technical accuracy. By leveraging relevant documentation and source code, *RAG* was able to provide answers aligned with Firefox's current artifacts. In several cases, evaluators preferred *RAG* answers because they included concrete details and illustrative examples without unnecessary elaboration (Questions 5 and 6).

Question 5
Can i convert NSString to NSString?
RAG's Answer: <p>Yes, you can convert an NSString (UTF-16) to an NSString (UTF-8) using the <code>NS_ConvertUTF16toUTF8</code> helper class. For example, you can do <code>NSString utf8String = NS_ConvertUTF16toUTF8(utf16String);</code> where <code>utf16String</code> is your NSString instance.</p>
Evaluator Comment:

[RAGAnswer] has the key function name and no other irrelevant information.

Question 6

Hi all, I'm working on making some changes to the new tab hover preview feature and I have come up against a strange issue. I am trying to set up a MutationObserver to listen to changes on other elements in the frontend (to decide if the tab preview should be disabled or not), but when I try to set it up (even with a no-op callback function), I get an `'UNEXPECTED_NS_ERROR_FAILURE'` in the console. There is a javascript traceback pointing to my call, but there doesn't seem to be any way to get a more descriptive error message. Has anyone here run into something like this before?

RAG's Answer:

It sounds like the `'UnexpectedNS_ERROR_FAILURE'` you're encountering might be due to the MutationObserver being set up on a node that isn't fully available or accessible at the time of the observer's initialization. Ensure that the target node exists and is fully loaded before setting up the observer. You can use `document.addEventListener('DOMContentLoaded', ...)` or `window.onload` to ensure the DOM is fully loaded before initializing the MutationObserver. Additionally, double-check that the node you're observing is not within a shadow DOM or another context that might restrict access.

Evaluator Comment:

This [RAGAnswer] gives a small code example of what might work to fix the issue and also points out other possible issues.

Despite the positive aspects of *RAG* answers, they were not flawless. While *RAG* was more specific than *GPT*, it occasionally missed key contextual details that *Human* answers captured. Furthermore, *RAG*'s reliance on retrieval sometimes led to overly detailed answers that, while accurate, could be overwhelming.

5.5.5

RAG for Developer Assistance

Results suggest that *RAG* has potential for real-world use in large-scale projects like Mozilla Firefox. By integrating project-specific knowledge, *RAG* could help developers answer questions requiring precise documentation references or technical details. An advantage is its ability to reduce the burden on senior developers, who often spend time addressing repetitive queries. Automating responses enables core maintainers to focus on high-impact development tasks rather than troubleshooting common issues.

Additionally, *RAG* can improve onboarding by retrieving and synthesizing up-to-date information from evolving project documentation, reducing the learning curve for new contributors. However, successful deployment requires

fine-tuning to ensure responses remain concise and contextually relevant. Prioritizing essential documentation while filtering out unnecessary details is critical for maintaining engagement in fast-paced discussions. While *RAG* cannot replace human expertise, it presents a promising approach to enhancing productivity, improving information accessibility, and reducing response time in developer communities.

5.6 Threats to Validity

We discuss threats to the study’s validity in what follows (Wohlin et al. 2012).

Construct and Internal Validity. Our study evaluates the effectiveness of Retrieval-Augmented Generation (*RAG*) in the context of developer assistance within the Firefox project. A key threat to construct validity relates to our choice of the evaluation metrics, namely, helpfulness, comprehensiveness, and conciseness. While these attributes are relevant for assessing response quality, they may not fully capture all dimensions of usefulness in real-world development scenarios. Additional factors, such as accuracy, actionability, and developer trust in AI-generated responses, could influence adoption and effectiveness. Another concern is the requirement for one-paragraph responses, which may have systematically favored concise over comprehensive answers and potentially biased our evaluation since conciseness was one of the assessment criteria; however, this requirement was necessary (Chaves et al. 2021) to match communication expectations in developer chat channels.

Another concern stems from the selection of the questions analyzed. Although we applied systematic filtering criteria to extract a representative set of developers’ questions, it is important to note that our dataset is inherently limited to Mozilla’s Matrix chat rooms. Therefore, the sampling process excluded other relevant sources of developers’ interaction, such as GitHub issues and mailing lists, which might vary in the characteristics of the questions and answers. Thus, we cannot generalize our findings to all sources of developers’ online interaction, even for the Firefox project.

The evaluation process may introduce bias, as all eight evaluators were Mozilla engineers. While their domain expertise ensures informed judgments, it also increases the risk of implicit preferences, particularly for *human* answers. To mitigate this, all responses were anonymized and randomized, and the sources were unrevealed. However, one evaluator recognized their prior answer, as noted in the qualitative feedback. Although this was an isolated case, it highlights a limitation of our design. Future studies could involve external

evaluators to mitigate this risk, although such an approach may introduce trade-offs related to domain expertise.

Our correlation analysis links preferred answers to those that are most helpful, comprehensive, or concise. We acknowledge that such distinctions may be less expressive when answers are similar in quality. Additionally, experts may have variability in their assessments, and we assessed agreement in four questions with overlapping evaluations. Our analysis focused on aggregate tendencies rather than individual-level modeling. Future work could adopt more sophisticated approaches, such as mixed-effects models, to better account for evaluators' variation.

Ultimately, the models' performance is dependent on the specific LLM version used at the time of the study. GPT-4o and its *RAG* variant are subject to continuous updates, which may affect their performance in future replications. It poses a reproducibility challenge, as different versions of the same model may yield significantly different results. While this is a known aspect in LLM research, we provide detailed methodological descriptions to aid in future replications. Besides, in the case of *GPT*, OpenAI also provides the previous LLM versions in its API to mitigate this threat. However, we cannot assure that these versions will always remain available, as the *GPT* models are proprietary.

Conclusion and External Validity. The primary concern in terms of the external validity of our study is the generalizability of our findings. While Firefox is a widely used OSS project, different projects may have varying levels of documentation, distinct communication styles, and different developer expectations regarding AI-generated responses. Thus, while our results demonstrate the potential of *RAG* in enhancing developer assistance, they may not be translated directly to other OSS communities without further validation.

Moreover, our study only considered responses from GPT-4o and GPT-4o enhanced with *RAG*. The effectiveness of alternative LLMs or different retrieval-augmented approaches remains an open question. Variations in training data, retrieval mechanisms, or fine-tuning strategies could lead to different performance outcomes. Future research should explore a broader range of models and techniques to determine whether similar improvements can be observed across different AI systems.

Finally, while our evaluation provides insights into response quality, real-world adoption of *RAG* depends on additional factors such as usability and ethical considerations. LLMs can produce hallucinated or misleading answers, which could introduce risks in high-stake development tasks. Future work

should explore long-term usability and developer trust to better understand how RAG-based assistance tools can be effectively integrated into OSS development practices.

5.7

Conclusion

This paper evaluates the effectiveness of Retrieval-Augmented Generation (RAG) for LLMs in addressing technical queries from open source developers within the Mozilla Firefox project. Through systematic comparison of responses from human experts, baseline GPT models, and RAG-enhanced systems, we assessed the relative performance of these approaches in providing helpful, comprehensive, and concise technical assistance.

Our findings indicate that RAG-enhanced responses achieve greater comprehensiveness than human responses while maintaining comparable levels of helpfulness. This indicates that RAG systems can effectively leverage project-specific documentation and source code to generate more detailed technical explanations without sacrificing quality or relevance.

The study highlights the potential of RAG-based approaches to transform developer assistance in open source environments. By automating responses to common technical questions, preserving high response quality, such systems can substantially reduce the workload burden on core maintainers. This has important implications for the sustainability and scalability of open source projects, particularly as they grow in complexity and community size.

Several directions emerge for future research. First, we plan to extend our evaluation framework to additional open source projects across different domains to assess the generalizability of our findings and identify domain-specific factors that influence RAG effectiveness. Second, we aim to develop more sophisticated retrieval strategies that can handle specialized requirements such as system dependability, security considerations, and performance constraints. Third, longitudinal studies are necessary to understand the long-term adoption patterns of developers and the role of LLM-assisted knowledge systems in OSS development workflows. Finally, investigating the integration of RAG systems with existing developer tools and platforms, as well as evaluating them with end users, represents promising research avenues.

6

Developers' Perceptions of Automated Detection and Resolution of Software Dependability Issues

This section describes a ongoing work to submit.

In Chapters 4 and 5, we introduced conversational agents designed to assist developers by answering technical questions in an open-source environment. While these studies demonstrated the feasibility of using large language models (LLMs) combined with augmentation strategies (RAG) for supporting collaborative software development, they focused solely on Q&A and did not directly intervene in code.

In this chapter, we address a more fundamental challenge: moving from interaction-level assistance to code-level improvements in dependability. As already mentioned, prior studies revealed that NFRs are consistently underemphasized in developer practice. In the Spring ecosystem, for instance, robustness was discussed far less than maintainability, security, or performance (Oliveira et al. 2024). Similarly, in Apache projects as described in Chapter 2, exception-related pull requests focused mainly on log messages. At the same time, deeper concerns such as error detection and propagation received little attention. These studies suggest that several dimensions of dependability are frequently underprioritized due to their complexity, low visibility, and lack of supporting tools.

While prior work has shown that LLMs can repair benchmark bugs (Sobania et al. 2025, Ramos et al. 2024), other studies demonstrate that multi-agent architectures can address exception safety by targeting exception handling paths (Zhang et al. 2025). However, these approaches are mostly evaluated in controlled benchmark settings, raising the threat that models may have already memorized solutions during training. Motivated by these gaps, we investigate whether a multi-agent pipeline can extend its role beyond answering questions to generating reliable code changes. Our suggested pipeline differs from the previous one by integrating multiple steps: detection, validation, repair, and repair evaluation for dependability issues, within the context of closed-source systems. In doing so, it aims not only to strengthen resilience and fault tolerance of software, but also to evaluate whether practitioners perceive the generated fixes.

We evaluate this approach on three closed-source python systems to examine whether it can generate fixes that practitioners perceive as relevant, effective, and trustworthy. We conducted a structured evaluation of 129 sampled files, during which the pipeline identified 149 potential issues and proposed corresponding fixes. From these, we selected 40 representative issue-fix pairs, each of which was reviewed independently by two experienced developers. Our evaluation design combined quantitative ratings, using Likert-scale questions, with qualitative feedback, resulting in a total of 80 assessments.

The results, while framed by the scope of our sample, were encouraging. Developers consistently rated the identified issues as relevant (mean = 4.47; Agree to Strong Agree) and the proposed fixes as effective (mean = 4.35; Agree to Strong Agree), and they expressed a general willingness to approve these fixes in code reviews (mean = 4.16; Agree to Strong Agree). These outcomes suggest that the pipeline does not merely generate superficial recommendations but is capable of surfacing meaningful and actionable improvements.

Our qualitative analysis revealed that the pipeline was particularly effective in addressing concerns related to integrity, reliability, and maintainability. Its limitations appeared in availability and context-dependent security issues, likely due to the absence of runtime information and the inherent complexity of security analysis. Importantly, disagreements among reviewers tended to stem from contextual assumptions or perceptions of redundancy, rather than from technical shortcomings in the proposed fixes.

Taken together, these findings highlight both the promise and the current limitations of conversational agents for enhancing software dependability. Our results show that even if the pipeline's suggestions are perceived as relevant and technically sound, the decision to adopt them must remain under human control. Developers play a critical role in judging whether a proposed fix aligns with system-specific constraints, architectural principles, and long-term maintenance goals. By considering not only detection accuracy but also developer perceptions and adoption potential, this study provides an initial but significant step toward understanding how a pipeline of multiple agents can support dependable software development.

This chapter details the findings of this evaluation, which forms a core contribution of this Ph.D. thesis. Specifically, it addresses the research question: *How do developers perceive a multi-agent architecture for detecting and fixing software dependability issues?*

6.1 Introduction

Ensuring software dependability, a non-functional requirement that encompasses attributes like reliability, security, integrity, and maintainability, is a critical part of ensuring the continued quality of any given software (Sobania et al. 2025, Ramos et al. 2024, Wu et al. 2024, Zhang et al. 2025). However, the increasing complexity of software systems makes the manual detection and resolution of dependability issues a challenging and resource-intensive endeavor (Földvári et al. 2021). Given that scenario, recent works have observed that developers have been neglecting some of these attributes, such as reliability (Correia et al. 2024, Oliveira et al. 2024, Oliveira et al. 2023), when maintaining their software.

The recent advent of Large Language Models (LLMs) has shown significant promise in automating various software development tasks, from code generation to automated repair (Sobania et al. 2025, Jiang et al. 2025). This has led to the exploration of new evaluation paradigms, such as “LLM-as-a-Judge” where models assess the quality and correctness of generated code, moving beyond traditional metrics that often fail to capture semantic correctness (Ren et al. 2020).

Despite these advancements, much of the existing works are limited to fundamental code generation and repair scenarios (Du et al. 2024, Zhou et al. 2024), which do not fully represent the multifaceted nature of dependability in complex, industrial software systems. Consequently, there is a limited understanding of how effective these automated approaches are in detecting and fixing genuine dependability issues in real-world contexts and, just as importantly, how professional developers perceive and value their outputs. Without human-centered evaluation, it is difficult to assess whether such tools can be meaningfully integrated into existing development workflows.

In this paper, we address this gap by proposing and evaluating a multi-agent LLM pipeline designed to automatically detect and resolve software dependability issues. Our approach employs a pipeline composed of four stages: Issue Detector, Issue Validator, Code Fixer, and Code Judger. This pipeline analyzes software repositories to identify problems and generate actionable fixes. To assess its practical relevance, we conducted a study involving eight experienced developers from a private research and development institute in Brazil. These developers evaluated 40 solutions generated by our pipeline for three closed-source, industrial Python systems currently in production.

Our quantitative results indicate a positive reception, with developers acknowledging the relevance of the identified problems (mean Likert score of

4.47 out of 5) and the effectiveness of the proposed fixes (4.35 out of 5). The qualitative analysis shows that the pipeline is particularly adept at uncovering issues related to integrity, reliability, and maintainability. Furthermore, our analysis of developer feedback reveals that while most suggestions were well-received, disagreements typically stemmed from differing assessments of issue severity or contextual misalignments, reinforcing that human oversight remains crucial for validating automated solutions.

In summary, the main contributions of this paper are:

- (i) the design and implementation of a novel, multi-agent pipeline that leverages multiple LLMs to automatically detect and resolve software dependability issues;
- (ii) a comprehensive, human-centered evaluation of the pipeline's effectiveness and relevance, conducted with professional developers on three real-world industrial systems;
- (iii) a mixed-methods analysis of developer perceptions, combining quantitative ratings and qualitative feedback to identify the factors influencing the acceptance of automatically generated fixes; and,
- (iv) a characterization of the dependability dimensions that the pipeline most effectively addresses, offering insights into the current strengths and limitations of automated code analysis tools.

6.2 Related Work

Large Language Models (LLMs) have recently been employed not only for code generation and repair, but also as judges to evaluate answers. Standard evaluation metrics, such as CodeBLEU, rely on human-written references and often do not capture semantic correctness (Ren et al. 2020). To overcome these limitations, researchers have explored the *LLM-as-a-Judge* paradigm, where models assess responses directly based on quality or correctness (Zheng et al. 2023). Previous works have studied prompting strategies, such as pairwise versus pointwise evaluation (Tripathi et al. 2025), or incorporated critiques to mitigate biases (McAleese et al. 2024). However, existing benchmarks are either small in scale or limited to basic code generation tasks, usually focused on *Mostly Basic Python Problems Dataset* (Google Research) or *HumanEval* (Google Research) datasets (Austin et al. 2021, Chen et al. 2021). Du et al. (Du et al. 2024), evaluated LLMs on code generation at the class level, highlighting the need for more realistic and complex assessments. Specialized

approaches, such as LLM4PatchCorrect (Zhou et al. 2024) and CriticGPT (McAleese et al. 2024), attempt to tailor models for code judging, but remain constrained in coverage and robustness. Jiang et al. (Jiang et al. 2025) substantially extends previous work by introducing CodeJudgeBench, a large-scale benchmark covering three fundamental coding tasks: code generation, code repair, and unit test generation. This work enabled a more comprehensive evaluation of LLM judges in various coding scenarios.

Other studies have investigated the use of LLMs for dependability problems. Using automated program repair (APR), Sobania et al. (Sobania et al. 2025) and Ramos et al. (Ramos et al. 2024) showed that ChatGPT repairs a significant share of benchmark bugs, while cautioning about generalization beyond standard test suites. Regarding Maintainability, Wu et al. (Wu et al. 2024) gather LLMs with tools (e.g., iSMELL) to improve code-smell *detection* and *refactoring*. For *reliability and robustness*, Zhang et al. (Zhang et al. 2025) propose *Seeker*, a multi-agent framework that targets exception safety by locating weak exception paths and suggesting repairs. In *security*, Zhou et al. (Zhou et al. 2024) survey rapid progress in vulnerability *detection* and *patching*, and Zibaeirad et al. (Zibaeirad et al. 2024) introduce VulnLLMEval to probe model limits on real-world repositories. These advances push dependability detection and fixing forward, although most evaluations relies on benchmarks rather than embedded in industrial workflows.

We address this gap by targeting real industrial systems and treating *dependability* as the core objective. Our agent pipeline goes beyond judging code: it *detects*, *validates*, and *fixes* dependability issues, with developers reviewing the outcomes. We also *employ an LLM-as-a-Judge* to automatically evaluate issue fix and quality, this approach is well-studied for general code correctness but remains uncommon in dependability pipelines (Zheng et al. 2023, Jiang et al. 2025). The approach shifts the focus from isolated benchmarks to development practice. In particular, because we do not evaluate on public benchmark datasets, we mitigate the threat of memorization or data-contamination (that is, models recalling benchmark' answers), highlighted by Ramos et al. (Ramos et al. 2024). The goal is not only to test LLM capability, but also to demonstrate the possibility of having a deployable pipeline that proactively detects and repairs dependability, such as through the automated creation of a pull request by a bot.

6.3 Study Design

In this section, we describe the study design. We begin by introducing the proposed pipeline designed to automatically detect and address dependability issues in software systems. Next, we outline our research questions that will guide the evaluation of the pipeline's effectiveness and relevance. We then explain the data collection process, the manual assessments performed by developers, and finally, the methods used for data analysis.

6.3.1 Architecture for Automatic Detection and Resolution of Dependability Issues

We designed and implemented a multi-agent pipeline that takes the path to a software repository as input and automatically analyzes it to detect and propose solutions for issues that may compromise software dependability. The overall architecture of the pipeline is depicted in Figure 6.2.

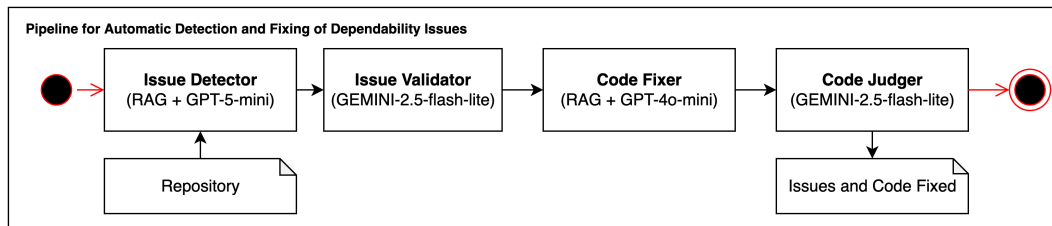


Figure 6.1: Pipeline for automatic detection and resolution of dependability issues.

The proposed pipeline, as depicted in Figure 6.2, consists of four interconnected agents, each relying on the output of the previous one. The agent identifies potential dependability issues in the source files. The second agent then validates these detected issues, filtering out false positives and unnecessary noise. Notably, we employed models from different companies for detection/validation and fix/judgment in an effort to mitigate bias. The third agent generates fixes in the target code. Finally, the last agent evaluates whether the proposed solutions effectively address the identified issues.

One of the primary challenges in developing a pipeline for detecting and proposing code changes is defining the prompts for each agent. This process involves several iterations of prompt engineering. We applied well-established strategies from the prompt engineering literature. These strategies included clearly defining the role of the model, specifying the task, and providing one-shot examples in the expected JSON output format (He et al. 2024, Kong et al. 2024, Shorten et al. 2024). These approaches offer several advantages. By

defining the role, we can focus the model's perspective, reducing irrelevant or overly creative responses. Stating the task clearly enhances clarity and ensures that the output remains reliable and focused. Additionally, by providing examples of the expected output, we minimize variability in the responses, which improves the model's precision and machine-readability of the results.

It is important to note that during our architectural design phase, GPT-5 was released, and we investigated the use of this new model in our study. This exploration highlighted the importance of employing effective prompting techniques. Unlike previous models, the GPT-5 API does not allow for strict output constraints or temperature tuning, which means there is less control at the parameter level. As a result, strategies such as role specification and providing format examples became essential for effectively guiding the responses. In the following subsections, we discuss each agent and its characteristics.

6.3.1.1 Issue Detector

The processing begins with a repository folder, which is the input to the analysis. The first step involves an agent we call *Issue Detector*, implemented as a combination of Retrieval-Augmented Generation (RAG) mechanism and the GPT-5-mini model. We used the Framework Cognita to implement our RAG mechanism, which we host locally (Local-RAG). Initially, we created a collection in the Cognira Framework. This collection basically consists of computing the embeddings for a set of documents (all the files in the analyzed repository) to input as context in queries to LLM models. Once the collection is created, Cognita enables the execution of queries to the models through an API, automatically forwarding the context retrieved from the collection. So, one file in the repository at a time, we performed queries (i.e., through the Local-Rag) to identify possible dependability issues. The output of this process is a set of potential dependability issues with the issue title, description, file, impact and the affected source code.

The Frame 6.3.1.1 shows the structure used for the prompt at this stage, the complete prompt can be found in the supplementary material (Correia et al. 2025). In our experiments with various model configurations, we discovered that GPT-5-mini, although slower in response than GPT-4o-mini, consistently yielded more accurate and comprehensive detections. The extra time taken during the detection step was justified by the improvements in both the quality and coverage of identified issues.

Issue Detector Prompt

Role You are a senior Python developer specialized in highly dependable systems [...]

Task Perform a thorough code review of the code file below, focused on detect dependability issues [...]

File: {file content}

Output: Return a JSON with:

```
{\,"title": "Handle missing config file",
  "location": "settings.py",
  "issue": "Fails if file absent ...",
  "impact": "Startup crash",\}
```

Context {retrieved content}

6.3.1.2

Issue Validator

The second stage in our pipeline is the *Issue Validator*, which is powered by Gemini-2.5-Flash-Lite. We chose a model from a different organization to avoid the possible biases in conducting detection and validation with models coming from the same design. The *Issue Validator* received as input the issue title, description, location, and affected code. The Frame 6.3.1.2 presents the prompt structure delivered to the model. We asked the model to evaluate whether an issue was truly related to dependability. The role of the *Issue Validator* was to check the *Issue Detector*'s findings and filter out false positives. By doing so, it ensures that only issues with strong evidence and clear dependability implications progress to the fixing stage.

Issue Validator Prompt

Role You are a senior Python developer specialized in highly dependable systems. [...]

Task Critically evaluate whether a reported issue is a valid dependability issue. Reject style-only concerns [...] or negligible impacts. [...]

Reported Issue: title, description, location and affected code.

Output: Return a JSON with:

```
{"description": "This issue is a valid...",
  "classification": "Confirm"}
```

6.3.1.3

Code Fixer

After undergoing double validation by the *Issue Validator*, the identified dependability issues were passed on to the Code Fixer. The information was structured as follows: issue title, description, and affected code. Utilizing our Local-RAG and the GPT-4o-mini model, we tasked the agent with generating a solution for the specified problem in the affected code. The prompt design for this stage is illustrated in Frame 6.3.1.3. We evaluated various configurations, including GPT-5-mini, and determined that GPT-4o-mini provided the best balance between latency and quality for this phase. The objective of the Code Fixer is to minimize "creative" modifications and to leverage repository context to maintain coding conventions while enhancing reliability, integrity, and maintainability.

Code Fixer Prompt

Role

You are a senior Python developer specialized in highly dependable systems [...]

Task

Think critically, step by step: evaluate the provided dependability issue and [...] fix the code [...]

Reported Issue:

title, description and affected code.

Output:

A JSON with two fields [...]

```
{ "description": "The create method ...",  
  "fixed_code": "def create(self,...)"
```

Context

{retrieved content}

6.3.1.4

Code Judger

The final stage, the *Code Judger* was also powered by Gemini-2.5-Flash-Lite. At this point, we already have a candidate solution for the identified issues, so we instruct the agent to assess whether this candidate adequately addresses the problems. The judger evaluates the issue title, description, the affected code, and the proposed fix, ultimately issuing a final verdict of either “Confirm” or “Reject,” the solution, accompanied by a brief rationale that highlights specific changes made to the code. A condensed version of the judger’s instructions can be found in Prompt 6.3.1.4. Finally, any issue that

successfully passes through all four stages is considered a recommendation for improving the system's dependability.

Final Judger Prompt

Role You are a senior Python developer specialized in highly dependable systems. [...]

Task Critically evaluate step by step whether the reported issue was completely fixed [...]

Reported Issue: title, description and affected code.

Proposed Solution: fixed code.

Output: Return a JSON with:

```
{"description": "This issue is a valid...",
"classification": "Confirm"}
```

6.3.2 Research Questions

To evaluate the proposed pipeline, we defined two research questions that address both the technical effectiveness of our pipeline and its practical relevance from the perspective of developers.

RQ₁: How effective is a multi-agent LLM pipeline at improving software dependability?

In this first research question, we aim to explore the effectiveness of our multi-agent pipeline in detecting dependability issues. We measure effectiveness in two dimensions: the extent to which the detected issues cover the main dependability aspects, and how experienced developers rate these detected issues. First, we analyze issues' descriptions generated by the pipeline and categorize them according to the nature of the dependability concerns. Each issue is mapped to attributes such as reliability, maintainability, confidentiality, safety, availability, and integrity. This analysis provides a clearer understanding of the pipeline's detection capabilities, highlighting both its strengths and limitations. Next, we assess effectiveness from the perspective of experienced developers in three dimensions: relevance, adequacy of the proposed solution, and acceptance of the suggested code changes. Through this evaluation, we examine whether the issues highlighted by the pipeline are perceived as contextually relevant, whether the proposed fixes are seen as effective resolutions to the problems, and whether the resulting code would be acceptable in a real review process.

RQ₂: How do developers assess the quality and relevance of issues and fixes generated by the pipeline?

We pose this question to identify the technical factors that either strengthen or weaken the mergeability and actionability of the pipeline's outputs in practice. Our goal is to understand developers' perceptions of these outputs. While structured evaluations offer a quantitative perspective, they often do not capture practitioners' views on the usefulness, adequacy, and trustworthiness of identified issues and proposed solutions. By collecting qualitative feedback from developers who maintain code, we can gain insights into whether the pipeline's contributions align with real-world development practices. This approach can also highlight situations in which our pipeline's outputs are more likely to be integrated into existing workflows.

6.3.3 Data Collection

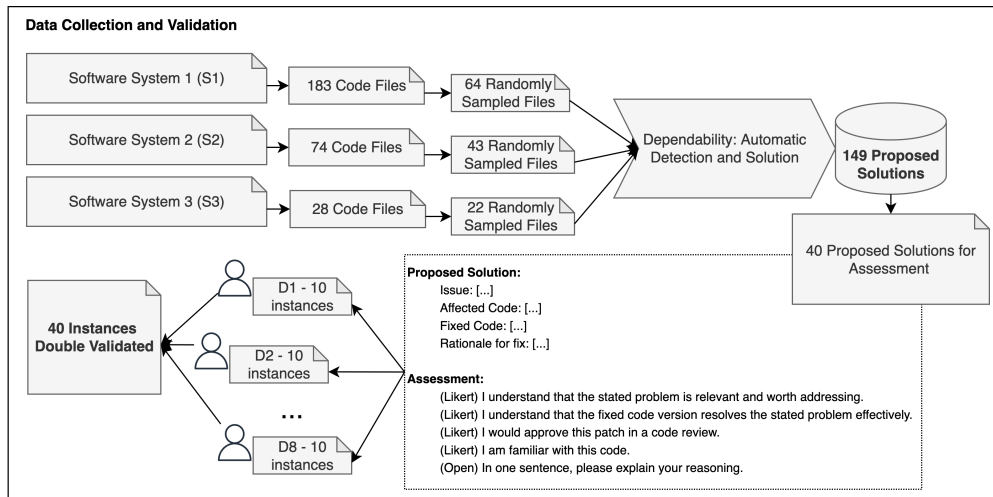


Figure 6.2: Data collection and validation.

Our data collection process begins with the identification of target projects for dependability analysis. We prioritize projects that allow us to validate dependability issues through expert code evaluation. To accomplish this, we collaborate with a private research and development institute based in Brazil. We have identified and established three different software systems in this context. All of these systems are written in Python and are currently in production for various government institutions in Brazil. Additional information about these systems can be found in Table 6.1.

We have a total of 285 code files available for analysis: 183 files from system S_1 , 74 files from system S_2 , and 28 files from system S_3 . To ensure our analysis was both feasible and representative, we applied random sampling

Table 6.1: Characterization of Target Systems by Type and Domain.

System	Type	Domain
S_1	REST/API	Back-end for an Advanced Process Control Tool in industry.
S_2	REST/API	Back-end for a Process Improvement Tool integrated with a Government System.
S_3	REST/API	Back-end for a Conversational Agent-based Tool integrated with a Government System.

to select a subset of files from each system. The sample size (n) for each system was calculated using a 95% confidence level, which allowed us to obtain a statistically valid subset of files. We sampled 64 files from S_1 , which is approximately 35% of its total files. From S_2 , we selected 43 files, representing about 58% of its total. Since S_3 was smaller, we considered all 22 files (78.6%). Altogether, this process resulted in a sample of 129 files, nearly half (45%) of all available files across the systems. Table 6.2 summarizes the data aforementioned.

Table 6.2: Overview of Issue Distribution by System and Pipeline Stage.

System	Files	Sampled Files	Issue Detector (# issues detected)	Issue Validator (# issues validated)	Code Fixer (# issues fixed)	Code Judge (# issues validated)
S_1	183	64	107	97	97	70
S_2	74	43	82	79	79	54
S_3	28	22	41	39	39	25

The next step was to apply our pipeline to the sampled files. For S_1 , we collected 70 issues across 21 unique files, which corresponds to approximately 33% of the sampled files. In sample S_2 , the system identified 54 issues in 15 files (35% of the sampled files), while in sample S_3 , it reported 25 issues across eight files (36% of the sampled files). Overall, this stage resulted in 149 issues with proposed solutions.

For our data collection purposes, the pipeline generates a structured spreadsheet where each row captures the complete details of an analyzed issue. The columns include: *title*, *location*, *affected code*, *description*, *impact*, *label*, *validator justification*, *validator classification*, *fixed code*, *fix description*, *judge justification*, and *judge classification*. Collectively, these parameters form a detailed record that can, for example, be used to automatically generate pull requests in the source repository, providing developers with actionable, evidence-based suggestions to improve reliability, integrity, and maintainability.

Following, we validate the issues and solutions with the developers who worked on these systems. To avoid overwhelming them with the need to manually validate all solutions, we selected a subset of 40 solutions, which

constitutes approximately 27%. This selection aimed to strike a balance between comprehensive coverage and practicality, ensuring sufficient diversity while keeping the validation workload manageable.

6.3.4 Manual Validation Process

Each of these 40 proposed solutions was double-validated by independent developers. In practice, this meant that every solution was reviewed by two different developers, ensuring cross-checking and reducing individual bias. We engaged eight developers, D_1 to D_8 , with each one validating 10 solutions. In total, this process produced 80 individual evaluations (40 solutions \times 2 reviewers). Table 6.3 shows more details about the developers' background.

Table 6.3: Developers Background

Developer	Education Level	Experience (years)	Experience in Company	Languages
D1	Master's	7	4	Python, Javascript, Java, C#
D2	Master's	10	2	Python
D3	Undergraduate	5	3	Python, Javascript, C/C++
D4	Undergraduate	2	1	Python, Matlab, C, C++
D5	Undergraduate	6	4	C, C++, C#, Java, Python, SQL, JavaScript, TypeScript
D6	Undergraduate	1	1	Python
D7	Master's	15	5	Ruby, JavaScript, Python
D8	Undergraduate	7	1	Python, Javascript

We designed a structured evaluation form. For each proposed solution, developers were presented with:

1. **Contextual information** – including the description of the identified issue, the affected code fragment, the automatically generated fixed code, and a brief rationale explaining the fix.
2. **Closed questions** – in these four questions, we employed a Likert scale with five points, ranging from *Strongly Disagree (1)* to *Strongly Agree (5)*. The statements were:
 - “I understand that the stated problem is relevant and worth addressing.”
 - “I understand that the fixed code version resolves the stated problem effectively.”
 - “I would approve this patch in a code review.”
 - “I am familiar with the code and context of this solution.”

3. **An open-ended field** – developers were asked to justify their decision in **one sentence**. This qualitative input offered deeper insight into the reasoning behind their assessments.

Each question in the evaluation form was carefully designed to capture a specific dimension of the developers' assessment of the proposed fixes. The first statement, "*I understand that the stated problem is relevant and worth addressing*", aims to assess whether the evaluator agrees that the identified issue is meaningful and requires attention, thereby validating the significance of the problem detected by the pipeline. The second statement, "*I understand that the fixed code version resolves the stated problem effectively*", focuses on the correctness and adequacy of the proposed solution, prompting reviewers to consider whether the automated fix addresses the root cause without introducing regressions. The third statement, "*I would approve this patch in a code review*", serves as a holistic judgment, integrating multiple aspects such as problem relevance, fix adequacy, and code quality into a realistic scenario of peer review. Finally, the fourth statement, "I am familiar with the code and context of this solution", was included to capture potential variation in confidence and contextual understanding, enabling us to interpret the other answers in light of the evaluator's familiarity with the underlying codebase. By integrating structured Likert-scale ratings with open-ended textual justifications, the validation process captured both quantifiable assessments and contextual insights into developer reasoning, enabling a more comprehensive understanding of their evaluation criteria.

6.3.5 Data Analysis

To answer our research questions, we conducted a multi-step analysis of the issues and fixes produced by the pipeline. We examined how effectively the pipeline identified and addressed dependability concerns and how developers assessed the quality and relevance of those outputs. We combined quantitative and qualitative data analysis to provide a broad view of the pipeline's technical and practical impact.

6.3.5.1 Quantitative Analysis

We started by converting the Likert scale evaluations into numerical scores ranging from 1 (Strongly Disagree) to 5 (Strongly Agree). For each of the 40 solutions, we aggregated the responses from the two developers by calculating their mean values. This resulted in a single representative score

for each solution across four evaluation dimensions: relevance of the problem, effectiveness of the fix, likelihood of approval in a code review, and familiarity with the code.

Using the aggregated scores, we calculated overall means for all solutions to establish the perceptions of problem significance, adequacy of the fix, acceptance in reviews, and contextual familiarity. Aiming at identifying patterns of agreement and disagreement between reviewers, we plotted the data and visually inspected any divergences. We reviewed cases where the two evaluators had notable discrepancies in their assessments, as these instances indicate solutions that received mixed judgments regarding their dependability or appropriateness.

6.3.5.2

Qualitative Analysis.

Initially, the issue descriptions were systematically analyzed using an open coding approach. Each description was carefully examined, and relevant codes were assigned to capture the nature and impact of the identified problem. Based on these codes, we further mapped each issue to one dependability dimension, such as integrity, reliability, maintainability, or security. In this way, we provide a structured view of the types of concerns addressed by the pipeline. This process enabled us to assess not only the technical relevance of the detected issues but also the range of dependability attributes covered. To further understand the pipeline's practical scope, we computed the frequency of issues associated with each dimension, offering insight into which areas of dependability the pipeline is most effective at uncovering, and identifying potential gaps in its detection capabilities.

Additionally, the open-ended justifications provided by developers were systematically analyzed also using an open coding approach. Each comment was coded to capture the underlying rationale supporting or rejecting the pipeline's proposed issue or fix. These codes were then grouped into broader categories through axial coding, distinguishing between positive rationales (supporting the relevance and adequacy of the solution) and negative rationales (highlighting reasons for rejection or criticism). Positive codes encompassed, for example, explicit mentions of improved validation, strengthened security, or reliable error handling. Negative codes often reflected concerns about incomplete fixes, low-severity issues, or misalignment with project context. By combining this categorical analysis with the quantitative Likert-scale ratings, we obtained a more comprehensive view of developer evaluations, balancing measurable agreement levels with nuanced reasoning behind acceptance or

rejection.

6.4

Results and Discussions

The evaluation of the pipeline to address dependability issues produced positive results. In the three closed-source systems analyzed, the developers recognized the relevance of the identified problems and the effectiveness of the proposed solutions. The experts mainly agreed on the recommendations as clear improvements in security, data integrity, and system reliability. When disagreements occurred, they were more about contextual assumptions than technical validity. In general, our findings suggest that the pipeline offers valuable recommendations based on dependability, although human oversight and contextual knowledge are still crucial.

6.4.1

Effectiveness in Resolving Dependability Issues

Improving software dependability is a complex challenge that involves not only identifying critical issues but also generating technically sound and context-aware solutions. Our first research question (RQ_1) investigates whether a multi-agent pipeline can effectively contribute to this goal by automatically identifying problems that impact key dependability attributes and proposing feasible solutions.

To address RQ_1 , we conducted a qualitative analysis of the pipeline's output to understand the dimensions of dependability associated with the detected issues. As detailed in the Study Design section (Section 6.3), we employed an open coding process on the textual descriptions of the identified issues, followed by a categorization process that linked each issue to a specific dependability dimension (e.g., reliability, maintainability). The results of this analysis are presented in Table 6.4, which displays the distribution of identified issues across various dependability dimensions, the number of occurrences out of 40 issues, and examples from the coding process.

Table 6.4: Frequency of Detected Dependability Issues

Category	Issues	Relevant Codes
Integrity	13	missing serializer-level validation; nullable constraint mismatch; conflicting model constraints; ...
Reliability	10	KeyError due to missing request context; brittle dependency on DRF runtime; no rollback on DB failure; ...
Maintainability	8	missing language localization in exceptions; undesirable side effects on import; unhandled log level values; ...
Security	5	unauthenticated user assignment risk; refresh token exposed in API; unsafe filename interpolation; ...
Confidentiality	3	sensitive user fields exposed in nested serializer; log exposes user email; PII risk in API response; ...
Availability	1	missing DB connection closure; potential connection leak; resource exhaustion over time; ...

The most frequently represented dependability dimension was *integrity*, with 13 detected issues, followed by *reliability*, with 10. This distribution

reflects the pipeline's strength in identifying problems related to data consistency, input validation, and predictable system behavior—concerns that are often explicitly manifested in the code. Typical examples include missing serializer-level checks, constraint mismatches, and improper error handling. *Maintainability* accounted for 8 issues, such as unlocalized exception messages, log inconsistencies, and side effects, which are generally detectable through structural or syntactic patterns. Although less common, the pipeline also flagged *security* (5 issues) and *confidentiality* (3 issues) problems, including exposed tokens and Personal Identification Information leaks. *Availability* was the least represented dimension, with only one issue identified.

The results indicate that the pipeline has the potential to detect issues that significantly impact all aspects of dependability, although it identifies fewer problems in some categories. This lower frequency is expected, as concerns such as security often require contextual understanding that static code representations may not fully capture. Additionally, the absence of availability issues likely reflects the inherent limitations of static analysis in identifying problems that generally arise under dynamic runtime conditions, such as resource leaks or concurrency bottlenecks. Overall, the findings show that the pipeline effectively addresses a wide range of dependability concerns, especially in the areas of integrity, reliability, and maintainability. However, there are opportunities to improve its coverage of more contextual or runtime dimensions, such as availability and security.

To further address RQ_1 , we conducted human-centered evaluations in which expert developers reviewed the pipeline's outputs. Table 6.5 presents the average Likert-scale ratings given by these evaluators on four dimensions: relevance of the issue, effectiveness of the fix, approval of the patch, and familiarity with the code. Overall, the results indicate a generally positive reception, suggesting that the pipeline is capable of producing outputs that developers find meaningful, accurate, and worthy of review.

Table 6.5: Mean Ratings and Likert Scale Interpretation

Question	Mean	Likert Interpretation
I understand that the stated problem is relevant and worth addressing.	4.47	Agree to Strongly Agree
I understand that the fixed code version resolves the stated problem effectively.	4.35	Agree to Strongly Agree
I would approve this patch in a code review.	4.16	Agree
I am familiar with this code.	4.00	Agree

We achieved the highest rating on the relevance of the issue dimension, with a mean score of 4.47. This value reflects a response trend between agree and strongly agree for the statement: “*I understand that the stated problem is relevant and worth addressing*”. As illustrated in the corresponding scatter plot (Figure 6.3), individual ratings were consistently high across

evaluators. This result suggests that developers systematically recognized the issues identified by the pipeline as meaningful and worthy of attention across the evaluated systems. These findings highlight the ability of the pipeline to detect dependability concerns that align with professional judgment and established notions of relevance in real-world software engineering contexts.

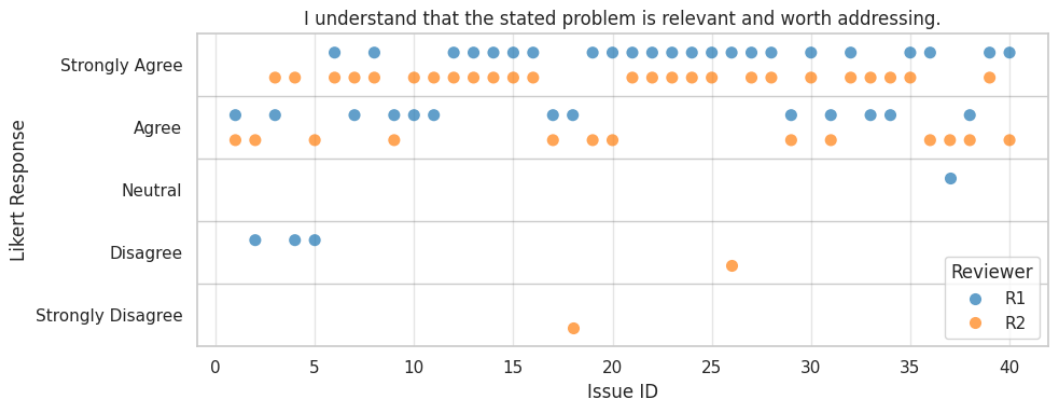


Figure 6.3: Relevance Scores for all Issues.

The second-highest mean score, 4.35, corresponds to the effectiveness of the fix dimension. This indicates a consistent trend toward agreement or strong agreement with the statement: "*I understand that the version of the fixed code effectively solves the stated problem*". As shown in the scatter plot for this dimension (Figure 6.4), ratings were generally favorable across reviewers. These results suggest that the proposed fixes were perceived not only as technically adequate but also contextually appropriate within the evaluated systems, reflecting the pipeline’s capacity to generate targeted and meaningful improvements.



positive range. This dimension, depicted in the corresponding scatter plot (Figure 6.5), shows that most evaluators expressed agreement, indicating a general willingness to endorse the fixes through standard review practices. These results reflect not only confidence in the technical correctness of the generated code but also its practical acceptability from the perspective of experienced developers engaged in real-world development workflows.

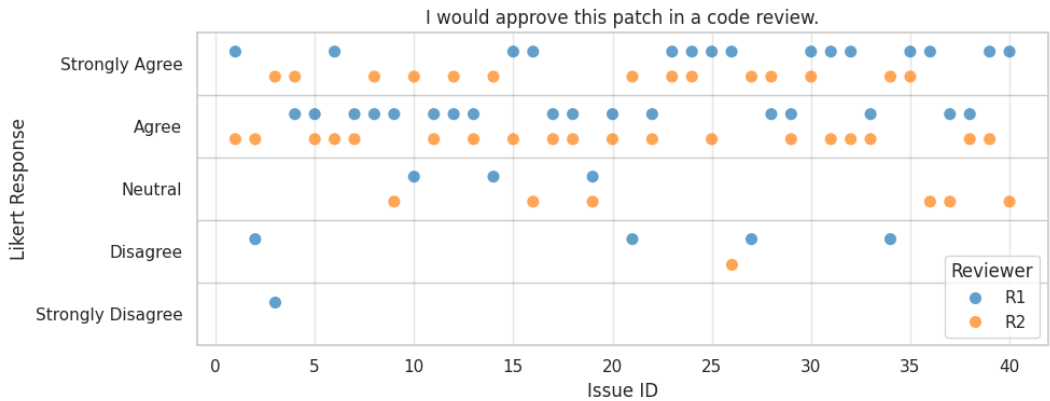


Figure 6.5: Approval Scores for all Issues.

Finally, as a control dimension, we assessed the familiarity of developers with the specific code under review. This dimension received a mean score of 4.00, which also falls within the agree range. The scatter plot for this dimension (Figure 6.6) shows slightly more variation among individual ratings, likely due to the fact that, although all evaluators were actively involved in the overall project, not all had contributed to the specific modules under review. Nonetheless, the overall score suggests a sufficient level of confidence in understanding the code, thereby reinforcing the credibility and contextual grounding of the evaluators' assessments across the other dimensions.

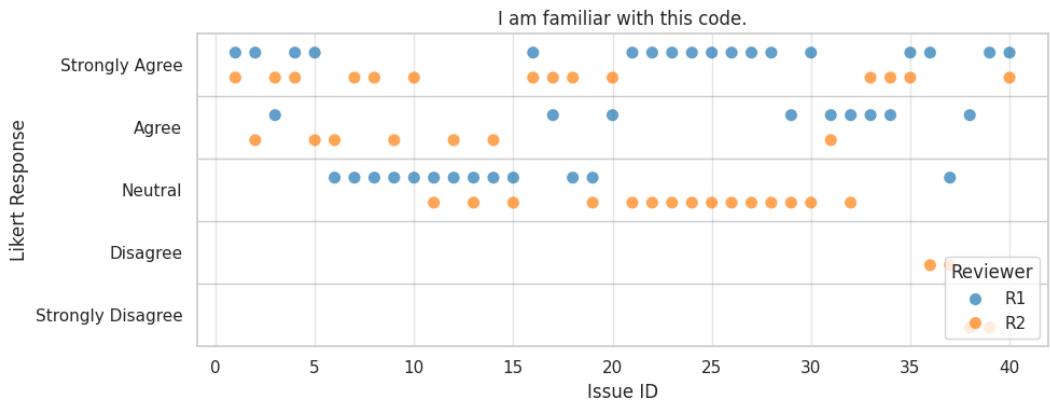


Figure 6.6: Familiarity with the Evaluated Code Scores for all Issues.

The findings indicate that the pipeline generates valuable outputs: developers consistently rated the flagged issues as relevant, the fixes as effective,

and the patches as reviewable. However, these ratings mainly reflect the perceived quality of the identified items, rather than the overall coverage of the dependability space. In other words, positive ratings do not determine which types of concerns the pipeline is most or least sensitive to, nor do they indicate its recall across various dimensions.

To address this limitation, we analyze the pipeline's detection profile through our coding analysis (see Table 6.4). This analysis reveals that the pipeline is more effective at detecting code-local integrity, reliability, and maintainability issues, while it shows comparatively weaker coverage of context- and runtime-dependent categories, such as availability and security. This distinction highlights the strengths and limitations of the pipeline and suggests opportunities for expanding its coverage to include more contextual and dynamic behaviors.

6.4.2 Quality of Generated Fixes

In the second research question (RQ_2), we aimed to understand how expert developers qualitatively assess the fixes generated by the pipeline. While previous analyses primarily used structured Likert-scale ratings, this section delves into the open feedback provided by evaluators, offering deeper insight into their perceptions of the proposed solutions. Table 6.6 presents the resulting categories, organized by whether the comments supported (Positive) or rejected (Negative) the proposed solutions, along with descriptions and examples of representative codes.

Table 6.6: Open Coding Categories Derived from Expert Feedback on Proposed Solutions

Suggestion Acceptance	Category	Description	Representative Codes
Positive	Fix Security and Privacy	Strengthens confidentiality, prevents leaks, and improves overall system security.	critical security flaw; removes PII from logs; fix ensures password complexity
Positive	Fix Validation and Data Integrity	Ensures input correctness, schema consistency, and clearer error feedback.	serializer validation improves feedback; data integrity risk; PositiveIntegerField enforces constraints
Positive	Fix Effectiveness and Reliability	Delivers efficient fixes that solve problems and enhance reliability.	simple and effective fix; correct rollback usage; improved observability
Negative	Incomplete or Misaligned Fixes	Represents fixes seen as partial, generic, or contextually unsuitable.	incomplete solution; generic fix criticized; fix lacks context
Negative	Low Severity Issue	Covers low-priority concerns or fixes adding little practical benefit.	minor issue; not serious issue; issue unlikely due to current design
Positive	Bug Recognition and Relevance	Highlights expert agreement on the existence and relevance of bugs.	valid issue acknowledged; bug acknowledged; context-aware bug
Negative	Uncertain Parameter Validation	Questions the necessity or redundancy of certain validation checks.	manual validation needed in some cases; parameter assumed present; unnecessary extra checks
Positive	Fix Approval	Indicates expert acceptance of fixes, sometimes with conditions.	approval without reservations; approval with suggestion; testing needed before approval
Negative	Security and Robustness Concerns	Points to lingering risks, leaks, or added maintenance burden.	exception message removed; requires future maintenance

6.4.2.1 Agreement on Experts Evaluations

In several cases, experts consistently recognized both the relevance of the identified issues and the adequacy of the proposed fixes. These instances illus-

trate situations where the pipeline's suggestions aligned well with established development practices and expert expectations.

A representative example was the case of the soft delete mechanism (Issue 40). The model contained a *deleted_at* field, which implied an intended strategy for soft deletion, but lacked the necessary supporting methods to enforce it. The default `delete()` operation still triggered hard deletion, creating the risk of inconsistent data handling if developers alternated between calling `.delete()` and manually setting *deleted_at*. Experts agreed that this represented a relevant dependability issue, as it could compromise data retention policies; however, they commented that it could impact for the systems queries to introduce inconsistencies in the state of the database.

The proposed fix introduced a *soft_delete()* method, replacing the default *delete()* to prevent hard deletions, and added a *DocumentQuerySet* with a helper method to filter active records. Experts consistently evaluated this fix as effective and aligned with expected practices in system. The open coding categories most associated with this consensus were Fix Validation and Data Integrity (ensuring consistency in how deletions are applied) and Fix Effectiveness and Reliability (a solution that directly addressed the identified risk and enforced consistent usage).

This pattern of agreement highlights that the pipeline was particularly successful in cases where the identified problem corresponded to well-understood dependability concerns (e.g., consistency, data integrity, and recoverability) and where the proposed fix provided a clear, contextually appropriate mechanism. In such cases, expert evaluations converged, reinforcing the potential of automated pipelines to deliver dependable fixes that meet developers' expectations without ambiguity.

6.4.2.2

Disagreements on Experts Evaluations

Although most of the evaluations demonstrated convergence between experts in terms of the relevance of the detected issues and the suitability of the proposed solutions, several cases revealed disagreements. These divergences highlight how contextual knowledge, the design of each system, and differing interpretations of dependability requirements shape developers' judgments.

First, some disagreements arise from contextual misalignment. For example, in the multilingual exception handling case (Issue 2), one expert viewed the lack of structured serialization and bilingual support as a minor issue, categorizing the fix as a improvement in error messages. In contrast, the second expert argued that the solution failed to understand the original design

choice of using English as the default language, consequently introducing unnecessary complexity. This situation falls under the category of “Incomplete or Misaligned Fixes,” where the pipeline’s proposals, although technically valid, were deemed contextually unsuitable.

Second, disagreements were also linked to different severity assessments. For instance, in serializers, the access to the request attribute assumes that the attribute exists (Issue 3); one developer minimized the likelihood of failure in practice, considering it an edge case. The second insisted that explicitly checking for request presence is important for defensive programming. Similarly, the *deleted_at* field manager (Issue 5) was seen by one expert as unlikely to cause failures because the field was guaranteed in the current model. In contrast, the other considered the check a relevant safeguard. These cases align with the “Low Severity Issue” category, where the same flaw is acknowledged but weighted differently depending on the perceived risk.

Other disagreements derived from uncertainties in parameter validation. In Issue 18, one attribute is not assigned during object creation, leading to a problem because the attribute is read-only. One developer emphasized that leaving the attribute unset introduced inconsistencies, while another argued that the context was missing since the attribute was always populated upstream. Similarly, in Issue 28, related to batch ID validation, opinions diverged between those who valued stricter upfront checks and those who considered them redundant or misapplied. These instances illustrate the “Uncertain Parameter Validation” category, where experts questioned whether additional validations were necessary or beneficial.

Finally, disagreements were also tied to different interpretations of security and robustness concerns. In the refresh token exposure case (Issue 34), one evaluator endorsed the fix as a necessary security measure, whereas the other criticized it as incomplete, since removing the token from the payload did not fully resolve the underlying vulnerability. Comparable situations appeared in database-related fixes (Issues 22, 27, 28), where one evaluator focused on improved reliability through validation or resource management, while the other emphasized potential side-effects, such as premature connection pool closure or suppressed errors. These patterns correspond to the “Security and Robustness Concerns” and “Incomplete or Misaligned Fixes” categories.

Overall, the disagreements show that expert evaluations were not simply binary judgments of correctness but were mediated by trade-offs among security, robustness, contextual relevance, and practical severity. The open coding results (Table 6.6) capture this diversity: positive categories (e.g., Fix Validation and Data Integrity, Fix Effectiveness and Reliability) often clashed

with negative categories (e.g., Incomplete or Misaligned Fixes, Low Severity Issue), depending on how experts weighed contextual constraints and long-term dependability. These findings underscore that the acceptance of automated fixes depends not only on their technical soundness but also on how they align with existing practices, assumptions, and risk perceptions in the target systems.

6.5

Threats to Validity

This section discusses potential threats to the validity of our study and the measures we took to mitigate them.

6.5.1

Internal Validity

Our random sampling strategy for selecting files from each system may introduce bias if the sampled files are not representative of the overall codebase characteristics. To mitigate this threat, we used statistical sampling with a 95% confidence level, ensuring a representative subset across all three systems. Additionally, the developers who evaluated the issues were all from the same organization and had varying levels of familiarity with the specific codebases under analysis. While developers actively work on the systems they evaluated, they were not necessarily reviewing code they had personally authored or maintained. This factor could introduce bias, particularly in how confidently they assessed the issues and fixes. To account for this, we included a familiarity dimension in our evaluation form as a control measure, allowing us to measure the extent to which each developer felt confident in their experience of the code under review.

6.5.2

External Validity

The study focused exclusively on Python systems, which may limit the generalizability of our findings to other programming languages. The results provide positive evidence of the pipeline's effectiveness within this context. However, additional validation is needed to determine if similar results are maintained across other technology stacks and application scenarios.

We validated 40 issues out of 149 issues (27%). This dataset may not capture the full spectrum of issues' characteristics. While this sample size was chosen to balance comprehensiveness with practical constraints, it may restrict our findings. Also, all evaluated systems came from a single research

and development institute in Brazil. The organizational culture, development practices, and quality standards may have influenced developer evaluations in ways that may not apply to other organizations or regions.

6.5.3

Construct Validity

The operational definition of dependability may not align with how other researchers or practitioners conceptualize this construct. We addressed this by grounding our definition in established literature and mapping detected issues to well-established dependability dimensions (reliability, security, integrity, maintainability, availability, confidentiality).

The Likert-scale questions we used to assess developer perceptions may not fully capture the nuanced judgments that developers make in real-world code review scenarios. We complemented these quantitative measures with open-ended qualitative feedback to provide richer insights into developer reasoning.

Moreover, our open coding approach for categorizing issues into dependability dimensions may have introduced subjectivity. We mitigated this by having multiple researchers review the coding scheme and ensuring consistency in the classification process.

6.5.4

Conclusion Validity

With only 40 evaluated solutions and 8 developers, our study may have limited statistical power to detect small but meaningful differences in developer perceptions. We addressed this by focusing on effective sizes and confidence intervals instead of relying on statistical tests.

Finally, we conducted multiple analyses across different dimensions and evaluation criteria, which increases the risk of Type I errors. We acknowledge this limitation and interpret our findings conservatively, focusing on patterns that are consistent across multiple dimensions.

6.6

Conclusion

This paper presented a comprehensive evaluation of a multi-agent LLM pipeline designed to automatically detect and resolve software dependability issues in industrial systems. Through a study involving eight experienced developers from a Brazilian research and development institute, we demonstrated that automated approaches can effectively identify meaningful dependability

concerns and generate actionable solutions that align with professional development practices.

Our evaluation revealed consistently positive developer perceptions across all evaluation dimensions. Developers strongly agreed that the identified problems were relevant and worth addressing (mean score of 4.47 out of 5), and they recognized the effectiveness of the proposed fixes (4.35 out of 5). The high approval rating for code review scenarios (4.16 out of 5) validates the practical applicability of the pipeline's outputs in real-world development workflows. The qualitative analysis showed that the pipeline is particularly effective at detecting problems related to *integrity* (13 issues), *reliability* (10 issues), and *maintainability* (8 issues), while also identifying *security* (5 issues) and *confidentiality* (3 issues) concerns.

While most evaluations showed agreement, disagreements came from four main areas: incomplete fixes, low severity issues, uncertain parameter validation, and security concerns. These disagreements demonstrate that human oversight and domain knowledge are essential for validating automated solutions. Our work makes three key contributions: first, we proved that a multi-agent LLM pipeline can find and fix real dependability issues in industrial systems; second, we showed how professional developers actually evaluate automated outputs through our study with 8 developers; and third, we identified that the pipeline works best for integrity (13 issues), reliability (10 issues), and maintainability (8 issues) problems.

The practical implications suggest that automated dependability analysis tools can be meaningfully integrated into existing development workflows, particularly for identifying issues related to data integrity, system reliability, and code maintainability. However, the importance of human oversight remains important, as contextual knowledge and domain expertise are essential for validating automated suggestions. Future work should explore the pipeline's effectiveness across diverse technology stacks and application domains, as well as investigate methods to improve detection of availability-related issues and enhance contextual understanding for more nuanced dependability concerns.

Conclusions and Future Work

7.1

Motivation

In modern software ecosystems, dependability is a fundamental quality attribute that includes availability, resilience in the presence of faults, and assurance for security. Exception handling mechanisms serve as the point where these attributes are enacted at the code level and faults are detected and addressed. Exception policies and their implementation directly shape the reliability/availability (graceful degradation and recovery), integrity (state rollback and resource cleanup), confidentiality (cleaning of error messages and stack traces), and maintainability (clarity of propagation paths and handler structure). However, achieving these properties is difficult in practice: incomplete documentation, contributor turnover, architectural drift, and missing design rationales frequently produce fragile exceptional paths and latent failures. Traditional techniques such as static analysis, model checking, and formal verification are valuable but often miss the socio-technical realities of evolving collaborative projects, precisely where exception-related decisions are negotiated between contributors.

Recent advances in LLMs present new opportunities to enhance developer assistance. However, their effectiveness relies heavily on having a rich project context, which includes requirements, coding standards, architectural rationale, and the current state of the codebase. With proper context, LLMs can provide tailored suggestions that boost developer productivity and improve code quality. This thesis uses exceptions as a framework to explore dependability. First, we characterize how exception-related collaborations occur in modern OSS, examining what developers change, why they make those changes, and the associated trade-offs. We then leverage these insights to design and evaluate LLM-based assistance that evolves from simple question-answering to implementing concrete modifications. These modifications include improving exception messages and propagation, fixing cleanup and rollback mechanisms, and adding tests for exceptional behavior. Finally, we generalize this approach into a multi-agent pipeline that detects and repairs broader dependability is-

sues, including, but not limited to, exception handling.

7.2

Thesis Contributions

This thesis delivers three key contributions, which are discussed in detail in the following sections: (C1) a systematic account of exception-related PRs in OSS, revealing their proactive and multifaceted nature; (C2) an evaluation of project-aware conversational agents like DevMentorAI, showing how grounding in project artifacts improves answer quality; and (C3) a practical multi-agent pipeline for detecting and fixing dependability issues in code, demonstrating the potential of LLM-based automation for robust software development. The Figure 7.1, maps how the research questions (RQs) led to findings, motivated further investigations, and ultimately shaped the contributions.

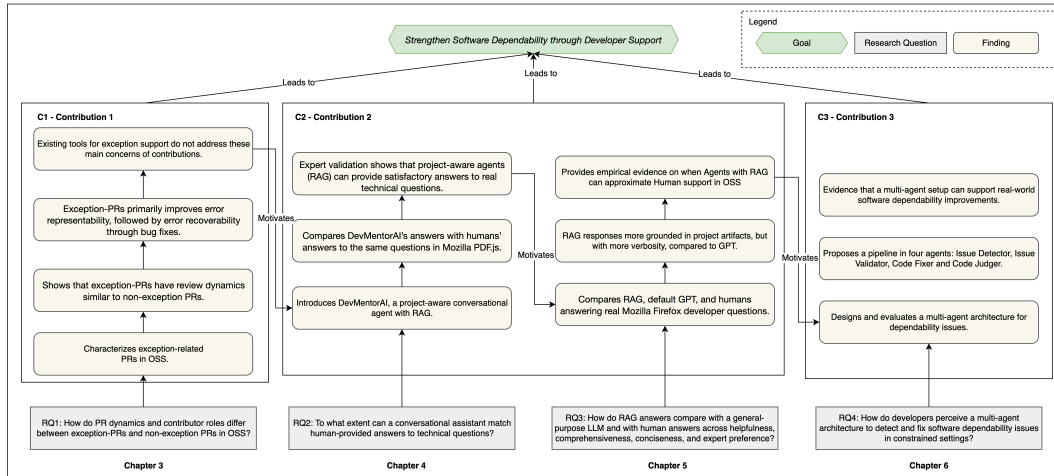


Figure 7.1: Flow of research questions, findings, and contributions leading to the thesis major goal of strengthening software dependability through developer support.

7.2.1

C1. Reframing Exception Handling as Routine (Chapter 3)

The first part of this thesis examines exception-handling in PRs to uncover how developers manage exceptions in OSS. By analyzing PR dynamics and manually characterizing exception-related changes, this study provides empirical evidence that frames exception handling as a routine and collaborative activity. This understanding offers a foundation for improving dependability practices through empirical awareness.

Approach. We analyzed 988 PRs from three Apache Java projects and manually inspected a stratified subset to characterize exception-related work.

Findings. Exception PRs are often proactive improvements (55.12%) rather than fixes (40.55%). The most frequently modified aspect was *external error representability* (42.52%), followed by *effective error handling* (36.22%). Review dynamics were comparable to non-exception PRs, challenging the assumption that specialized “exception engineers” dominate this space. Moreover, developers commonly touched multiple exception aspects in one PR, reflecting the interdependence of representability, detection, recovery, and propagation. Exception handling is not a niche; it is routine, reviewable work with strong ties to usability (messages) and maintainability—fertile ground for developer assistance.

Implications. This study documents exception handling as a routine, collaborative part of software development rather than a specialized task. However, it also raises a significant concern: the aspects of exceptions that developers most frequently address, such as error messages and handler logic are not well supported by current tools. Static analyzers and formal verification methods typically do not account for usability or maintainability concerns, while automated repair frameworks focus on generic bugs and neglect recovery practices. These findings provide a solid foundation for designing AI-driven solutions that better align with actual exception-handling work, improve robustness within standard workflows, and ultimately enhance software dependability.

C1: Exception-related PRs in OSS are routine, proactive, and multi-aspect, with review dynamics comparable, making exceptions a practical entry point for improving reliability and maintainability.

7.2.2

C2. Grounding LLMs in Project Artifacts Improves Answers (Chapters 4 and 5)

Building on the need for scalable developer assistance, the second contribution investigates how project-aware conversational agents, particularly those using retrieval-augmented generation (RAG), can support question answering in OSS environments. The findings compare LLMs, RAG-enhanced models, and human responses, uncovering how contextual grounding impacts relevance, completeness, and usability.

Approach. This contribution builds on two complementary studies. First, we integrated DevMentorAI into the PDF.js project to examine whether a project-specific RAG assistant could provide accurate and actionable answers to real developer questions. Second, we conducted a larger-scale comparison in the Firefox project, evaluating responses from RAG, GPT, and human contrib-

utors to 52 newcomer questions, with Mozilla engineers assessing helpfulness, comprehensiveness, and conciseness.

Findings. In PDF.js, DevMentorAI outperformed human answers in 8 of 14 questions, showing that project-aware assistants can match or exceed expert guidance in small-scale OSS settings. In the Mozilla Firefox study, RAG emerged as the *most helpful* source in 46.5% of questions and the *most comprehensive* in 55.8%, while humans and GPT were more often considered *most concise*. Importantly, RAG answers were also those engineers most preferred to see in practice (39.5%).

Implications Together, these two studies demonstrate that grounding LLMs with project artifacts (RAG) consistently enhances accuracy and coverage across contexts—from targeted experiments in PDF.js to broader evaluations in Firefox. At the same time, they highlight the trade-off between completeness and conciseness, underscoring the need to adapt presentation styles to review and onboarding workflows.

C2: LLM with RAG result in answers for technical questions that are more accurate and preferred by experts than generic LLMs or Humans, with a known completeness vs. conciseness trade-off for onboarding and reviews.

7.2.3

C3. Multi-Agent LLMs Produce Review-Ready Fixes for Real-World Dependability Issues (Chapter 6)

Finally, the thesis presents a multi-agent architecture that leverages LLMs to detect, validate, and fix code-level dependability issues in industrial systems. This contribution connects the earlier empirical and evaluative insights with actionable tooling, showing how conversational AI can evolve from informational assistance to dependable code transformation.

Approach. We built a four-agent pipeline (Issue Detector, Issue Validator, Code Fixer, Code Judger) and evaluated its outputs on three industrial systems with eight expert developers.

Findings. Developers rated issue relevance (4.47/5), fix effectiveness (4.35/5), and code-review approval (4.16/5) positively, with strongest coverage across integrity, reliability, and maintainability concerns. Disagreements clustered around incomplete fixes, low severity, parameter validation, and security.

Implications. Multi-agent LLMs can propose actionable patches that developers are willing to review, provided the system is explainable, context-aligned, and embedded in human-in-the-loop workflows.

C3: A multi-agent pipeline able to detect, validate, and repair real dependability issues, producing review-ready fixes when paired with explainability and human oversight.

7.3

Synthesis and Lessons Learned

Exception handling is everyday work, but tools lag behind practice. Our analysis of Apache Java projects revealed that exception-PRs represent less than 2% of all contributions, a subtle proportion given the critical role of exception handling in ensuring software dependability. These contributions typically target two key areas. First, they aim to improve the representation of errors, whether through enhanced diagnostic messages for developers (internal representation) or more explicit user-facing messages (external representation). Second, they focus on strengthening exception-handling mechanisms to address potentially unhandled failures and improve system recoverability.

One plausible interpretation of this behavior is that developers initially adopt broad or generic exception handling strategies to quickly stabilize control flow. These early implementations serve as safety nets, especially when faced with tight deadlines or changing requirements. As systems mature and developers gain a clearer understanding of potential exception scenarios, they revisit and refine these handlers. This refinement will likely constitute the most significant contributions observed in exceptional code. Typically, this involves narrowing the scope of exception handling, adding specific exception types, improving associated error messages, and aligning recovery actions with the project's fault tolerance policies. This iterative refinement process, however, is poorly supported by existing tools. Most exception-focused techniques, such as static analyzers and automated repair frameworks, remain centered on detecting violations of control flow or correcting evident bugs, offering little support for incremental improvements to handler quality or message clarity.

This mismatch highlights a significant gap: to better support dependable development, tools must shift their focus toward the real-world exception-handling practices observed in collaborative projects. This includes detecting and assisting with representational improvements and nuanced handler refinements, rather than limiting intervention to syntactic or runtime correctness checks. Importantly, this does not diminish the relevance of existing tools—static analyzers and automated repair frameworks remain essential for identifying critical faults and enforcing baseline correctness. Rather, our

findings point to a complementary opportunity: expanding tool capabilities to encompass the incremental, usability-oriented exception work that developers routinely perform but that current solutions tend to overlook.

Grounding agents matters more than scale. A consistent finding across the multiple studies is that providing project-specific context through documentation, source code, and other artifacts enables models to generate significantly more valuable answers compared to general-purpose LLMs. Both the DevMentorAI assistant, discussed in Chapter 4, and the RAG-based agent evaluated in Mozilla Firefox, covered in Chapter 5, exhibited superior performance not only over generic models but also, in certain scenarios, over human responses.

Importantly, the most significant answers were not just fluent or grammatically correct; they were deeply grounded in the technical specifics of the project, including internal APIs, architectural conventions, and up-to-date links to relevant information. Unlike human respondents, who may omit important details, assume implicit knowledge, or be in a rush, the RAG-based model consistently provides complete, polite, and well-structured answers.

Additionally, it has access to a broader and more stable knowledge base than generic LLMs, which rely solely on pretraining and may lack visibility into the latest project changes. This limitation is even more pronounced in closed-source environments, where general-purpose models do not have access to proprietary artifacts, internal documentation, or production code. These findings highlight that the effectiveness of assistants relies not only on linguistic ability or model scale but also on the strategic use of contextual grounding and retrieval tailored to the specific ecosystem of the project.

Verbosity is a challenging trade-off for agents. Although the conversational agents we explored produce answers with improved accuracy and coverage, they often produce verbose answers. This circumstance raises a concern: while detailed responses can enhance accuracy and completeness, they may hinder the practical use. This can be due to two scenarios: tasks are time-sensitive, or users prefer straightforward answers for personal reasons.

In our evaluations, developers consistently prioritized correctness but also requested more concise summaries and clearer structures. However, balancing these needs is particularly challenging. Excessive brevity can lead to responses that assume the reader has implicit knowledge or omit critical explanations, which can reintroduce issues commonly seen in human responses, such as lack of context or unstated assumptions. Future research should explore adaptive solutions, such as summarization techniques or user-controlled verbosity levels. These approaches could help retrieval-based agents strike a balance between

clarity and completeness without sacrificing trust or usability.

Human-in-the-loop is not optional. In our final study (Chapter 6), developers evaluated a multi-agent pipeline designed to detect and repair dependability issues in closed-source systems. While the proposed fixes were generally helpful and aligned with project standards, developers consistently highlighted the need for human oversight, especially when it came to architectural changes or those involving security components. A recurring concern was the risk of incorrect modifications leading to unintended consequences. Specific worries included breaking inter-component contracts, introducing redundant logic or cause side effects. These observations uncover a significant boundary of this kind of solution: some fixes require deeper architectural reasoning and negotiation with other stakeholders, which appear to go beyond the capabilities of fully automated solutions.

This limitation is particularly relevant in closed-source environments, where review cycles are often shorter and less collaborative than in open-source projects. In these contexts, developers may quickly approve and merge automated patches without thoroughly assessing their broader implications, which can inadvertently allow subtle but impactful faults to spread. These findings indicate that autonomous systems designed to support dependability must prioritize human supervision as a fundamental design principle. This approach is essential not only for ensuring technical correctness but also for addressing potential organizational blind spots that may arise during rapid code review processes.

7.4

Future Work and Final Remarks

This Thesis opens multiple promising directions for advancing dependable software development through intelligent, context-aware agents. While our studies demonstrate the feasibility and potential LLMs and RAG in supporting exception handling and other dependability-related tasks, there are several directions to improvement. Addressing these gaps will be crucial for transitioning from proof-of-concept pipelines to production-ready, developer-trusted systems.

A key area for enhancement is the capture of richer architectural information during issue detection and repair. The current pipeline heavily relies on localized code context and static artifact retrieval. However, many dependability concerns, particularly in exception handling, are architectural in nature. Issues such as cross-layer exception propagation, redundancy in error checks, and misaligned recovery responsibilities cannot be resolved without a com-

prehensive understanding of system-wide design decisions. Future iterations of the pipeline should incorporate architectural information, such as dependency graphs.

Equally important is the transformation of the pipeline from a static suggestion engine into an evidence-seeking collaborator. Instead of merely proposing a fix, agents should articulate their reasoning, cite supporting examples from the codebase or historical pull requests, and express confidence or uncertainty as appropriate. This shift would enable developers to better prioritize suggestions based not only on severity or scope but also on the quality and reliability of supporting evidence, thereby facilitating informed decision-making.

Additionally, integrating runtime context is a critical step forward. Many dependability issues only manifest under specific operational conditions. By incorporating telemetry data, log traces, test coverage information, and property-based test results, agents could uncover problems that are invisible in static code and generate fixes that are more aligned with real system behavior. Techniques such as fuzzing, chaos engineering, or fault injection could complement the existing static pipeline, allowing for broader detection and validation of potential failure modes.

The current pipeline also lacks mechanisms for learning from developer feedback. If a developer rejects a proposed issue or patch, the system does not adapt or attempt to refine its suggestions. Future work should focus on establishing feedback loops that enable the pipeline to learn from both explicit reviewer decisions (e.g., “rejected due to low severity” or “redundant with existing checks”) and implicit outcomes (e.g., reverts, edits, approvals). Reviewer comments, discussion threads, and post-review changes could all serve as valuable supervisory signals to refine prompts, fix templates, and retrieval strategies over time.

To enhance integration with developer workflows, we propose exploring review processes embedded within the pull request workflow. Instead of delivering monolithic suggestions, agents could open pull requests with inline comments. This approach reduces the review fatigue and clarify intentions.

In summary, transitioning from promising prototypes to practical, developer-trusted tools requires broadening the contextual scope, deepening runtime awareness, embracing feedback-aware learning, integrating seamlessly into workflows, validating across ecosystems, and rigorously measuring real-world outcomes. These directions not only extend the pipeline proposed in this Thesis but also contribute to a broader agenda for developing robust, intelligent, and dependable support systems for developers.

Bibliography

- [Achiam *et al.* 2023] ACHIAM, J.; ADLER, S.; AGARWAL, S.; AHMAD, L.; AKKAYA, I.; ALEMAN, F. L.; ALMEIDA, D.; ALTENSCHMIDT, J.; ALTMAN, S.; ANADKAT, S. ; OTHERS. **Gpt-4 technical report**. arXiv preprint arXiv:2303.08774, 2023.
- [Adejumo *et al.* 2024] ADEJUMO, E. K.; JOHNSON, B.. **Towards leveraging LLMs for reducing open source onboarding information overload**. In: PROCEEDINGS OF THE 39TH IEEE/ACM INTERNATIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGINEERING, p. 2210–2214, 2024.
- [Asaduzzaman *et al.* 2016] ASADUZZAMAN, M.; AHASANUZZAMAN, M.; ROY, C. K. ; SCHNEIDER, K. A.. **How developers use exception handling in java?** In: 13TH INTERNATIONAL CONFERENCE ON MINING SOFTWARE REPOSITORIES, p. 516–519, 2016.
- [Austin *et al.* 2021] AUSTIN, J.; ODENA, A.; NYE, M.; BOSMA, M.; MICHALEWSKI, H.; DOHAN, D.; JIANG, E.; CAI, C.; TERRY, M.; LE, Q. ; SUTTON, C.. **Program synthesis with large language models**. arXiv preprint arXiv:2108.07732, 2021.
- [Avizienis *et al.* 2004] AVIZIENIS, A.; LAPRIE, J.-C.; RANDELL, B. ; LANDWEHR, C.. **Basic concepts and taxonomy of dependable and secure computing**. IEEE Transactions on Dependable and Secure Computing, 1(1):11–33, 2004.
- [Azanza *et al.* 2024] AZANZA, M.; PEREIRA, J.; IRASTORZA, A. ; GALDOS, A.. **Can LLMs facilitate onboarding software developers? an ongoing industrial case study**. In: 2024 36TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING EDUCATION AND TRAINING (CSEE&T), p. 1–6. IEEE, 2024.
- [Azaria *et al.* 2024] AZARIA, A.; AZOULAY, R. ; RECHES, S.. **ChatGPT is a Remarkable Tool—For Experts**. Data Intelligence, 6(1):240–296, 02 2024.

- [Bairi et al. 2024] BAIRI, R.; SONWANE, A.; KANADE, A.; C, V. D.; IYER, A.; PARTHASARATHY, S.; RAJAMANI, S.; ASHOK, B. ; SHET, S.. **Codeplan: Repository-level coding using LLMs and planning**. Proceedings of the ACM on Software Engineering, 1(FSE):675–698, 2024.
- [Balfroid et al. 2024] BALFROID, M.; VANDEROSE, B. ; DEVROEY, X.. **Towards llm-generated code tours for onboarding**. In: WORKSHOP ON NL-BASED SOFTWARE ENGINEERING (NLBSE'24), 2024.
- [Barbosa and Garcia 2018] BARBOSA, E. A.; GARCIA, A.. **Global-aware recommendations for repairing violations in exception handling**. IEEE Trans. Software Eng., 44(9):855–873, 2018.
- [Barbosa and Garcia 2018] BARBOSA, E. A.; GARCIA, A.. **Global-aware recommendations for repairing violations in exception handling**. In: Chaudron, M.; Crnkovic, I.; Chechik, M. ; Harman, M., editors, 40TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, p. 858. ACM, 2018.
- [Barbosa et al. 2012] BARBOSA, E. A.; GARCIA, A. ; MEZINI, M.. **Heuristic strategies for recommendation of exception handling code**. In: 26TH BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING, p. 171–180. IEEE Computer Society, 2012.
- [Barbosa et al. 2012] BARBOSA, E. A.; GARCIA, A. F. ; MEZINI, M.. **A recommendation system for exception handling code**. In: 5TH INTERNATIONAL WORKSHOP ON EXCEPTION HANDLING, p. 52–54. IEEE, 2012.
- [Barbosa et al. 2014] BARBOSA, E. A.; GARCIA, A. F. ; BARBOSA, S. D. J.. **Categorizing faults in exception handling: A study of open source projects**. In: 2014 BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING, p. 11–20. IEEE Computer Society, 2014.
- [Barbosa et al. 2016] BARBOSA, E. A.; GARCIA, A. F.; ROBILLARD, M. P. ; JAKOBUS, B.. **Enforcing exception handling policies with a domain-specific language**. IEEE Trans. Software Eng., 42(6):559–584, 2016.
- [Bender et al. 2021] BENDER, E. M.; GEBRU, T.; MCMILLAN-MAJOR, A. ; SHMITCHELL, S.. **On the dangers of stochastic parrots: Can language models be too big?** In: PROCEEDINGS OF THE 2021

ACM CONFERENCE ON FAIRNESS, ACCOUNTABILITY, AND TRANSPARENCY, p. 610–623, 2021.

[Bergquist 2020] BERGQUIST, M.. **Open source software development as gift culture: Work and identity formation in an internet community**. In: NEW TECHNOLOGIES AT WORK, p. 223–241. Routledge, 2020.

[Black 1982] BLACK, A. P.. **Exception handling: The case against**. PhD thesis, University of Oxford, 1982.

[Bloch 2018] BLOCH, J.. **Effective Java**. Addison-Wesley, 3rd edition, 2018.

[Bommasani *et al.* 2021] BOMMASANI, R.; HUDSON, D. A.; ADELI, E.; ALTMAN, R.; ARORA, S.; VON ARX, S.; BERNSTEIN, M. S.; BOHG, J.; BOSSELUT, A.; BRUNSKILL, E. ; OTHERS. **On the opportunities and risks of foundation models**. arXiv preprint arXiv:2108.07258, 2021.

[Brisson *et al.* 2020] BRISSON, S.; NOEI, E. ; LYONS, K.. **We are family: Analyzing communication in github software repositories and their forks**. In: 2020 IEEE 27TH INTERNATIONAL CONFERENCE ON SOFTWARE ANALYSIS, EVOLUTION AND REENGINEERING (SANER), p. 59–69, 2020.

[Brown *et al.* 2020] BROWN, T.; MANN, B.; RYDER, N.; SUBBIAH, M.; KAPLAN, J. D.; DHARIWAL, P.; NEELAKANTAN, A.; SHYAM, P.; SASTRY, G.; ASKELL, A.; AGARWAL, S.; HERBERT-VOSS, A.; KRUEGER, G.; HENIGHAN, T.; CHILD, R.; RAMESH, A.; ZIEGLER, D.; WU, J.; WINTER, C.; HESSE, C.; CHEN, M.; SIGLER, E.; LITWIN, M.; GRAY, S.; CHESSE, B.; CLARK, J.; BERNER, C.; MCCANDLISH, S.; RADFORD, A.; SUTSKEVER, I. ; AMODEI, D.. **Language models are few-shot learners**. In: Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M. ; Lin, H., editors, ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, volumen 33, p. 1877–1901. Curran Associates, Inc., 2020.

[Buhr *et al.* 2000] BUHR, P. A.; MOK, W. R.. **Advanced exception handling mechanisms**. IEEE Transactions on Software Engineering, 26(9):820–836, 2000.

[Cabral and Marques 2011] CABRAL, B.; MARQUES, P.. **A transactional model for automatic exception handling**. Comput. Lang. Syst. Struct., 37(1):43–61, 2011.

- [Cacho *et al.* 2014] CACHO, N.; BARBOSA, E. A.; ARAUJO, J.; PRANTO, F.; GARCIA, A.; CESAR, T.; SOARES, E.; CASSIO, A.; FILIPE, T. ; GARCIA, I.. **How does exception handling behavior evolve? an exploratory study in java and c# applications.** In: IEEE INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE AND EVOLUTION, p. 31–40, 2014.
- [Cacho *et al.* 2014] CACHO, N.; CÉSAR, T.; FILIPE, T.; SOARES, E.; CASSIO, A.; SOUZA, R.; GARCIA, I.; BARBOSA, E. A. ; GARCIA, A.. **Trading robustness for maintainability: an empirical study of evolving c# programs.** In: 36TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, p. 584–595. ACM, 2014.
- [Canfora *et al.* 2012] CANFORA, G.; DI PENTA, M.; OLIVETO, R. ; PANICHELLA, S.. **Who is going to mentor newcomers in open source projects?** In: PROCEEDINGS OF THE ACM SIGSOFT 20TH INTERNATIONAL SYMPOSIUM ON THE FOUNDATIONS OF SOFTWARE ENGINEERING, p. 1–11, 2012.
- [Chase *et al.* 2022] CHASE, H.. **Langchain.** <https://github.com/langchain-ai/langchain>, 2022. Accessed: April 3, 2024.
- [Chaves *et al.* 2021] CHAVES, A. P.; GEROSA, M. A.. **The impact of chatbot linguistic register on user perceptions: a replication study.** In: INTERNATIONAL WORKSHOP ON CHATBOT RESEARCH AND DESIGN, p. 143–159. Springer, 2021.
- [Chen *et al.* 2021] CHEN, M.; TWOREK, J.; JUN, H. ; OTHERS. **Evaluating large language models trained on code.** arXiv preprint arXiv:2107.03374, 2021.
- [Chen *et al.* 2019] CHEN, H.; DOU, W.; JIANG, Y. ; QIN, F.. **Understanding exception-related bugs in large-scale cloud systems.** In: 2019 34TH IEEE/ACM INTERNATIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGINEERING (ASE), p. 339–351, 2019.
- [Chowdhery *et al.* 2022] CHOWDHERY, A.; NARANG, S.; DEVLIN, J. ; OTHERS. **Palm: Scaling language modeling with pathways.** In: PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON MACHINE LEARNING, p. 3878–3891, 2022.

- [Chung et al. 2012] CHUNG, L.; NIXON, B. A.; YU, E. ; MYLOPOULOS, J.. **Non-functional requirements in software engineering**, volumen 5. Springer Science & Business Media, 2012.
- [Cognita] COGNITA. **Cognita**. <https://github.com/truefoundry/cognita>. (Accessed on 07/15/2025).
- [Cornu et al. 2015] CORNU, B.; SEINTURIER, L. ; MONPERRUS, M.. **Exception handling analysis and transformation using fault injection: Study of resilience against unanticipated exceptions**. *Inf. Softw. Technol.*, 57:66–76, 2015.
- [Correia et al. 2024] CORREIA, J. A.; NICHOLSON, M. C.; COUTINHO, D.; BARBOSA, C.; CASTELLUCCIO, M.; GEROSA, M.; GARCIA, A. ; STEINMACHER, I.. **Unveiling the potential of a conversational agent in developer support: Insights from mozilla’s pdf.js project**. In: PROCEEDINGS OF THE 1ST ACM INTERNATIONAL CONFERENCE ON AI-POWERED SOFTWARE, Alware 2024, p. 10–18, New York, NY, USA, 2024. Association for Computing Machinery.
- [Correia et al. 2024] CORREIA, J.; COUTINHO, D.; GARCIA, A.; DE MELLO, R.; BARBOSA, C.; OLIVEIRA, A.; ASSUNÇÃO, W. K. G.; PEREIRA, J. A.; STEINMACHER, I.; GEROSA, M.; SOUZA, J. ; ARRIEL, J.. **On the investigation of exception pull request characteristics: Exploring the apache ecosystem**. In: 2024 IEEE INTERNATIONAL CONFERENCE ON SOURCE CODE ANALYSIS AND MANIPULATION (SCAM), p. 143–153, 2024.
- [Correia et al. 2024] CORREIA, J.; COUTINHO, D.; GARCIA, A.; DE MELLO, R.; BARBOSA, C.; OLIVEIRA, A.; ASSUNÇÃO, W. K. G.; PEREIRA, J. A.; STEINMACHER, I.; GEROSA, M.; SOUZA, J. ; ARRIEL, J.. **Chapter 3 - replication package**. 2024. Accessed on: 2024-06-23.
- [Correia et al. 2024] CORREIA, J.; COUTINHO, D.; GARCIA, A.; DE MELLO, R.; BARBOSA, C.; OLIVEIRA, A.; ASSUNÇÃO, W. K. G.; PEREIRA, J. A.; STEINMACHER, I.; GEROSA, M.; SOUZA, J. ; ARRIEL, J.. **On the investigation of exception pull request characteristics: Exploring the apache ecosystem**. In: 2024 IEEE INTERNATIONAL CONFERENCE ON SOURCE CODE ANALYSIS AND MANIPULATION (SCAM), p. 143–153, 2024.
- [Correia et al. 2025] CORREIA, J.; COUTINHO, D.; CASTELLUCCIO, M.; BARBOSA, C.; STEINMACHER, I.; GEROSA, M.; GARCIA, A.; DE MELLO, R.

- ; SARMA, A.. **Chapter 5 - replication package**. https://osf.io/9fjpm/?view_only=eb07b25a6b194c0fa9f771aed145786a, 2025.
- [Correia et al. 2025] CORREIA. **Chapter 6 - replication package**. https://osf.io/ysftv/overview?view_only=5d3ee534fe7e4bddb25ae3325a43951d, 2025.
- [Dabbish et al. 2012] DABBISH, L.; STUART, C.; TSAY, J. ; HERBSLEB, J.. **Social coding in github: transparency and collaboration in an open software repository**. In: PROCEEDINGS OF THE ACM 2012 CONFERENCE ON COMPUTER SUPPORTED COOPERATIVE WORK, CSCW '12, p. 1277–1286, New York, NY, USA, 2012. Association for Computing Machinery.
- [Dagenais et al. 2010] DAGENAIS, B.; OSSHER, H.; BELLAMY, R. K.; ROBIL-LARD, M. P. ; DE VRIES, J.. **Moving into a new software project landscape**. In: PROCEEDINGS OF THE 32ND ACM/IEEE INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE), p. 275–284. ACM, 2010.
- [Dakhel et al. 2023] MORADI DAKHEL, A.; MAJDINASAB, V.; NIKANJAM, A.; KHOMH, F.; DESMARAIS, M. C. ; JIANG, Z. M. J.. **Github copilot ai pair programmer: Asset or liability?** Journal of Systems and Software, 203:111734, 2023.
- [Devlin et al. 2019] DEVLIN, J.; CHANG, M.-W.; LEE, K. ; TOUTANOVA, K.. **Bert: Pre-training of deep bidirectional transformers for language understanding**. In: PROCEEDINGS OF NAACL-HLT, p. 4171–4186, 2019.
- [Discord] DISCORD. **Discord**. <https://discord.com/>. (Accessed on 05/14/2024).
- [Dodge 2008 (Mann-Whitney Test)] DODGE, Y.. **The concise encyclopedia of statistics: Mann–Whitney Test**, p. 327–329. Springer Science & Business Media, 2008.
- [Du et al. 2024] DU, X.; LIU, M.; WANG, K.; WANG, H.; LIU, J.; CHEN, Y.; FENG, J.; SHA, C.; PENG, X. ; LOU, Y.. **Evaluating large language models in class-level code generation**. In: PROCEEDINGS OF THE 2024 ACM/IEEE 44TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE 2024), p. 982–994, Lisbon, Portugal, 2024. IEEE Computer Society.

- [Ebert *et al.* 2015] EBERT, F.; CASTOR, F. ; SEREBRENIK, A.. **An exploratory study on exception handling bugs in java programs.** *Journal of Systems and Software*, 106:82–101, 2015.
- [Ebert *et al.* 2020] EBERT, F.; CASTOR, F. ; SEREBRENIK, A.. **A reflection on “an exploratory study on exception handling bugs in java programs”.** In: 2020 IEEE 27TH INTERNATIONAL CONFERENCE ON SOFTWARE ANALYSIS, EVOLUTION AND REENGINEERING (SANER), p. 552–556, 2020.
- [Fagerholm *et al.* 2014] FAGERHOLM, F.; SANCHEZ GUINEA, A.; BORENSTEIN, J. ; MÜNCH, J.. **Onboarding in open source projects.** *IEEE Software*, 31(6):54–61, 2014.
- [Fan *et al.* 2023] FAN, A.; GOKKAYA, B.; HARMAN, M.; LYUBARSKIY, M.; SENGUPTA, S.; YOO, S. ; ZHANG, J. M.. **Large language models for software engineering: Survey and open problems.** In: 2023 IEEE/ACM INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING: FUTURE OF SOFTWARE ENGINEERING (ICSE-FOSE), p. 31–53. IEEE, 2023.
- [Feng *et al.* 2025] FENG, Z.; STEINMACHER, I.; GEROSA, M.; MENEZES, T.; SEREBRENIK, A.; MILEWICZ, R. ; SARMA, A.. **The multifaceted nature of mentoring in OSS: strategies, qualities, and ideal outcomes.** In: 2025 IEEE/ACM 18TH INTERNATIONAL CONFERENCE ON COOPERATIVE AND HUMAN ASPECTS OF SOFTWARE ENGINEERING (CHASE), p. 203–214. IEEE, 2025.
- [Feng *et al.* 2022] FENG, Z.; CHATTERJEE, A.; SARMA, A. ; AHMED, I.. **A case study of implicit mentoring, its prevalence, and impact in apache.** In: PROCEEDINGS OF THE 30TH ACM JOINT EUROPEAN SOFTWARE ENGINEERING CONFERENCE AND SYMPOSIUM ON THE FOUNDATIONS OF SOFTWARE ENGINEERING, ESEC/FSE 2022, p. 797–809, New York, NY, USA, 2022. Association for Computing Machinery.
- [Finsås *et al.* 2024] FINSÅS, M.; MAKSIM, J.. **Optimizing rag systems for technical support with llm-based relevance feedback and multi-agent patterns.** Master’s thesis, NTNU, 2024.
- [Földvári *et al.* 2021] FÖLDVÁRI, A.; PATARICZA, A.. **Semi-automated model extraction from observations for dependability analysis.** In: 2021 IEEE INTERNATIONAL SYMPOSIUM ON SOFTWARE RELIABILITY ENGINEERING WORKSHOPS (ISSREW), p. 1–6. IEEE, 2021.

- [Fronchetti *et al.* 2023] FRONCHETTI, F.; SHEPHERD, D. C.; WIESE, I.; TREUDE, C.; GEROSA, M. A. ; STEINMACHER, I.. **Do contributing files provide information about oss newcomers' onboarding barriers?** In: PROCEEDINGS OF THE 31ST ACM JOINT EUROPEAN SOFTWARE ENGINEERING CONFERENCE AND SYMPOSIUM ON THE FOUNDATIONS OF SOFTWARE ENGINEERING, ESEC/FSE 2023, p. 16–28, New York, NY, USA, 2023. Association for Computing Machinery.
- [Garcia *et al.* 2001] GARCIA, A. F.; RUBIRA, C. M.; ROMANOVSKY, A. ; XU, J.. **A comparative study of exception handling mechanisms for building dependable object-oriented software.** *Journal of Systems and Software*, 59(2):197–222, 2001.
- [Gemini *et al.* 2023] TEAM, G.; ANIL, R.; BORGEAUD, S.; WU, Y.; ALAYRAC, J.-B.; YU, J.; SORICUT, R.; SCHALKWYK, J.; DAI, A. M.; HAUTH, A. ; OTHERS. **Gemini: a family of highly capable multimodal models.** arXiv preprint arXiv:2312.11805, 2023.
- [Google Research] GOOGLE RESEARCH. **MBPP: Mostly basic python problems (dataset and tasks).** <https://github.com/google-research/google-research/tree/master/mbpp>. Accessed: 2025-09-16.
- [Guizani *et al.* 2025] GUIZANI, M.; FENG, Z.; GARCIA, E. J. A.; KIMURA, K.; MUELLER, D.; DÍAZ, L. C.; SEREBRENIK, A. ; SARMA, A.. **Community tapestry: An actionable tool to track turnover and diversity in oss.** *Information and Software Technology*, 188:107871, 2025.
- [Guo *et al.* 2024] GUO, Q.; CAO, J.; XIE, X.; LIU, S.; LI, X.; CHEN, B. ; PENG, X.. **Exploring the potential of ChatGPT in automated code refinement: An empirical study.** In: PROCEEDINGS OF THE 46TH IEEE/ACM INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, p. 1–13, 2024.
- [Hagen *et al.* 2000] HAGEN, C.; ALONSO, G.. **Exception handling in workflow management systems.** *IEEE Transactions on software engineering*, 26(10):943–958, 2000.
- [Hao *et al.* 2025] YU, H.; GAN, A.; ZHANG, K.; TONG, S.; LIU, Q. ; LIU, Z.. **Evaluation of retrieval-augmented generation: A survey.** In: BIG DATA, p. 102–120, Singapore, 2025. Springer Nature Singapore.

- [He et al. 2024] HE, J.; RUNGTA, M.; KOLECZEK, D.; SEKHON, A.; WANG, F. X. ; HASAN, S.. Does prompt formatting have any impact on llm performance?, 2024.
- [IEEE Std. 928-2024] Ieee standard for measures of the software aspects of dependability. IEEE Std 982-2024 (Revision of IEEE Std 982-2005), p. 1–70, 2024.
- [ISO/IEC/IEEE 24765:2017] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. ISO/IEC/IEEE International Standard - Systems and software engineering–Vocabulary. ISO/IEC/IEEE 24765:2017(E), p. 1–541, 2017.
- [Ibtasham et al. 2025] IBTASHAM, M. S.; BASHIR, S.; ABBAS, M.; HAIDER, Z.; SAADATMAND, M. ; CICHETTI, A.. Reqrag: Enhancing software release management through retrieval-augmented llms: An industrial study. 2025.
- [Jakobus et al. 2015] JAKOBUS, B.; BARBOSA, E. A.; GARCIA, A. ; PEREIRA DE LUCENA, C. J.. Contrasting exception handling code across languages: An experience report involving 50 open source projects. In: 26TH INTERNATIONAL SYMPOSIUM ON SOFTWARE RELIABILITY ENGINEERING (ISSRE), p. 183–193, 2015.
- [Ji et al. 2023] JI, Z.; LEE, N.; XU, R. ; OTHERS. Survey of hallucination in natural language generation. ACM Computing Surveys, 55(12):1–38, 2023.
- [Jiang et al. 2025] JIANG, H.; CHEN, Y.; CAO, Y.; LEE, H.-Y. ; TAN, R. T.. Codejudgebench: Benchmarking llm-as-a-judge for coding tasks. arXiv preprint arXiv:2507.10535, 2025.
- [Käfer et al. 2018] KÄFER, V.; GRAZIOTIN, D.; BOGICEVIC, I.; WAGNER, S. ; RAMADANI, J.. Communication in open-source projects-end of the e-mail era? In: PROCEEDINGS OF THE 40TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING: COMPANION PROCEEDINGS, ICSE '18, p. 242–243, New York, NY, USA, 2018. Association for Computing Machinery.
- [Kechagia et al. 2014] KECHAGIA, M.; SPINELLIS, D.. Undocumented and unchecked: Exceptions that spell trouble. 05 2014.

- [Khojah et al. 2024] KHOJAH, R.; MOHAMAD, M.; LEITNER, P. ; DE OLIVEIRA NETO, F. G.. **Beyond code generation: An observational study of chatgpt usage in software engineering practice**. *Proceedings of the ACM on Software Engineering*, 1(FSE):1819–1840, 2024.
- [Knight 2012] KNIGHT, J. C.. **Fundamentals of Dependable Computing for Software Engineers**. CRC Press, 2012.
- [Kokinda et al. 2023] KOKINDA, E.; MOSTER, M.; DOMINIC, J. ; RODEGHERO, P.. **Under the bridge: Trolling and the challenges of recruiting software developers for empirical research studies**. In: ICSE-NIER '23, p. 55–59, Melbourne, Australia, 2023. Association for Computing Machinery.
- [Kong et al. 2024] KONG, A.; ZHAO, S.; CHEN, H.; LI, Q.; QIN, Y.; SUN, R.; ZHOU, X.; WANG, E. ; DONG, X.. **Better zero-shot reasoning with role-play prompting**, 2024.
- [Kou et al. 2023] KOU, B.; CHEN, M. ; ZHANG, T.. **Automated summarization of Stack Overflow posts**. In: 2023 IEEE/ACM 45TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE), p. 1853–1865. IEEE, 2023.
- [Kramer et al. n.d.] KRAMER, B.; KORKMAZ, G.; CALDERON, J. B. S. ; ROBINS, C.. **International collaboration in open source software: A longitudinal network analysis of github**.
- [Laprie et al. 1992] LAPRIE, J. C.; AVIZIENIS, A. ; KOPETZ, H.. **Dependability: Basic Concepts and Terminology**. Springer-Verlag, Berlin, Heidelberg, 1992.
- [Laranjeiro et al. 2021] LARANJEIRO, N.; AGNELO, J. ; BERNARDINO, J.. **A systematic review on software robustness assessment**. *ACM Computing Surveys (CSUR)*, 54(4):1–65, 2021.
- [Lewis et al. 2020] LEWIS, P.; PEREZ, E.; PIKTUS, A.; PETRONI, F.; KARPUKHIN, V.; GOYAL, N.; KÜTTLER, H.; LEWIS, M.; YIH, W.-T.; ROCKTÄSCHEL, T.; RIEDEL, S. ; KIELA, D.. **Retrieval-augmented generation for knowledge-intensive nlp tasks**. In: Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M. ; Lin, H., editors, *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, volumen 33, p. 9459–9474. Curran Associates, Inc., 2020.

- [Lyu 2007] Lyu, M. R., editor. **Handbook of Software Reliability Engineering**. McGraw-Hill, 2007.
- [Mathews et al. 2024] MATHEWS, N. S.; NAGAPPAN, M.. **Test-driven development and LLM-based code generation**. In: PROCEEDINGS OF THE 39TH IEEE/ACM INTERNATIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGINEERING, p. 1583–1594, 2024.
- [Matrix] MATRIX. **An open network for secure, decentralised communication**. <https://matrix.org/>. (Accessed on 07/15/2025).
- [McAleese et al. 2024] MCALEESE, N.; TRĘBACZ, M.. **Finding gpt-4’s mistakes with gpt-4**. url=<https://openai.com/index/finding-gpt4s-mistakes-with-gpt-4/>, 2024.
- [Mu et al. 2024] MU, F.; SHI, L.; WANG, S.; YU, Z.; ZHANG, B.; WANG, C.; LIU, S. ; WANG, Q.. **ClarifyGPT: A framework for enhancing LLM-based code generation via requirements clarification**. Proceedings of the ACM on Software Engineering, 1(FSE):2332–2354, 2024.
- [Mylavarapu et al. 2019] MYLAVARAPU, G.; THOMAS, J. P. ; VISWANATHAN, K. A.. **An automated big data accuracy assessment tool**. In: 2019 IEEE 4TH INTERNATIONAL CONFERENCE ON BIG DATA ANALYTICS (ICBDA), p. 193–197, 2019.
- [Nakshatri et al. 2016] NAKSHATRI, S.; HEGDE, M. ; THANDRA, S.. **Analysis of exception handling patterns in java projects: an empirical study**. In: 13TH INTERNATIONAL CONFERENCE ON MINING SOFTWARE REPOSITORIES, p. 500–503. ACM, 2016.
- [Nam et al. 2024] NAM, D.; MACVEAN, A.; HELLENDORRN, V.; VASILESCU, B. ; MYERS, B.. **Using an llm to help with code understanding**. In: PROCEEDINGS OF THE IEEE/ACM 46TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, p. 1–13, 2024.
- [Nguyen et al. 2024] NGUYEN, S.; BABE, H. M.; ZI, Y.; GUHA, A.; ANDERSON, C. J. ; FELDMAN, M. Q.. **How beginning programmers and code LLMs (mis) read each other**. In: PROCEEDINGS OF THE 2024 CHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, p. 1–26, 2024.
- [Nygard 2018] NYGARD, M. T.. **Release It!: Design and Deploy Production-Ready Software**. Pragmatic Bookshelf, 2nd edition, 2018.

- [Oliveira et al. 2023] OLIVEIRA, A.; CORREIA, J.; SOUSA, L.; ASSUNÇÃO, W. K. G.; COUTINHO, D.; GARCIA, A.; OIZUMI, W.; BARBOSA, C.; UCHÔA, A. ; PEREIRA, J. A.. **Don't forget the exception! : Considering robustness changes to identify design problems.** In: 2023 IEEE/ACM 20TH INTERNATIONAL CONFERENCE ON MINING SOFTWARE REPOSITORIES (MSR), p. 417–429, 2023.
- [Oliveira et al. 2024] OLIVEIRA, A.; CORREIA, J. A.; ASSUNÇÃO, W. K. G.; PEREIRA, J. A.; DE MELLO, R.; COUTINHO, D.; BARBOSA, C.; LIBÓRIO, P. ; GARCIA, A.. **Understanding developers' discussions and perceptions on non-functional requirements: The case of the spring ecosystem.** *Proc. ACM Softw. Eng.*, 1(FSE), July 2024.
- [Pandoc] PANDOC. **Pandoc: a universal document converter.** <https://pandoc.org/>. (Accessed on 05/14/2024).
- [Parametrix Solutions, Inc. 2025] PARAMETRIX SOLUTIONS, INC.. **CrowdStrike's impact on the fortune 500.** <https://www.parametrixinsurance.com/crowdstrike-outage-impact-on-the-fortune-500>, 2025. Accessed: 2025-09-19.
- [Petke et al. 2021] PETKE, J.; CLARK, D. ; LANGDON, W. B.. **Software robustness: a survey, a theory, and prospects.** In: 29TH ACM JOINT MEETING ON EUROPEAN SOFTWARE ENGINEERING CONFERENCE AND SYMPOSIUM ON THE FOUNDATIONS OF SOFTWARE ENGINEERING, p. 1475–1478, 2021.
- [Piantadosi et al. 2019] PIANTADOSI, V.; SCALABRINO, S. ; OLIVETO, R.. **Fixing of security vulnerabilities in open source projects: A case study of apache http server and apache tomcat.** In: 2019 12TH IEEE CONFERENCE ON SOFTWARE TESTING, VALIDATION AND VERIFICATION (ICST), p. 68–78, 2019.
- [Pinto et al. 2023] PINTO, G.; CARDOSO-PEREIRA, I.; MONTEIRO, D.; LUCENA, D.; SOUZA, A. ; GAMA, K.. **Large language models for education: Grading open-ended questions using chatgpt.** In: PROCEEDINGS OF THE XXXVII BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING, SBES '23, p. 293–302, New York, NY, USA, 2023. Association for Computing Machinery.

- [Pudari and Ernst 2023] PUDARI, R.; ERNST, N. A.. **From copilot to pilot: Towards ai supported software development**. arXiv preprint arXiv:2303.04142, 2023.
- [Qiu *et al.* 2010] QIU, X.; ZHANG, L. ; LIAN, X.. **Static analysis for java exception propagation structure**. In: 2010 IEEE INTERNATIONAL CONFERENCE ON PROGRESS IN INFORMATICS AND COMPUTING, volumen 2, p. 1040–1046, 2010.
- [Ramos *et al.* 2024] RAMOS, D.; MAMEDE, C.; JAIN, K.; CANELAS, P.; GAMBOA, C. ; GOUES, C. L.. **Are large language models memorizing bug benchmarks?**, 2025.
- [Ren *et al.* 2020] REN, S.; GUO, D. ; ET AL.. **Codebleu: a method for automatic evaluation of code synthesis**. In: PROCEEDINGS OF THE 2020 INTERNATIONAL CONFERENCE ON LEARNING REPRESENTATIONS (ICLR), 2020.
- [Replication Package 2024] **Chapter 4 - replication package**. <https://github.com/RESHAPELab/DevMentorAI-PDF.js-replication/>, 2024. Accessed: May 17, 2024.
- [Rey and Neuhäuser 2011] REY, D.; NEUHÄUSER, M.. **Wilcoxon-Signed-Rank Test**, p. 1658–1659. Springer Berlin Heidelberg, 2011.
- [Richards *et al.* 2024] RICHARDS, J.; WESSEL, M.. **What you need is what you get: Theory of mind for an LLM-based code understanding assistant**. In: 2024 IEEE INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE AND EVOLUTION (ICSME), p. 666–671, 2024.
- [Robillard *et al.* 2003] ROBILLARD, M. P.; MURPHY, G. C.. **Static analysis to support the evolution of exception structure in object-oriented systems**. ACM Transactions on Software Engineering and Methodology (TOSEM), 12(2):191–221, 2003.
- [Rocha *et al.* 2013] ROCHA, L. S.; ANDRADE, R. M. C. ; GARCIA, A. F.. **A method for model checking context-aware exception handling**. In: 27TH BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING, p. 59–68. IEEE Computer Society, 2013.
- [Ross *et al.* 2023] ROSS, S. I.; MARTINEZ, F.; HOUDE, S.; MULLER, M. ; WEISZ, J. D.. **The programmer’s assistant: Conversational interaction with a large language model for software development**.

- In: PROCEEDINGS OF THE 28TH INTERNATIONAL CONFERENCE ON INTELLIGENT USER INTERFACES, IUI '23, p. 491–514, New York, NY, USA, 2023. Association for Computing Machinery.
- [Sallou et al. 2023] SALLOU, J.; DURIEUX, T. ; PANICHELLA, A.. **Breaking the silence: the threats of using llms in software engineering**. arXiv preprint arXiv:2312.08055, 2023.
- [Santos et al. 2021] SANTOS, F.; WIESE, I.; TRINKENREICH, B.; STEIN-MACHER, I.; SARMA, A. ; GEROSA, M. A.. **Can I solve it? identifying APIs required to complete OSS tasks**. In: 2021 IEEE/ACM 18TH INTERNATIONAL CONFERENCE ON MINING SOFTWARE REPOSITORIES (MSR), p. 346–257. IEEE, 2021.
- [Sauvola et al. 2024] SAUVOLA, J.; TARKOMA, S.; KLEMETTINEN, M.; RIEKKI, J. ; DOERMANN, D.. **Future of software development with generative ai**. Automated Software Engineering, 31(1):26, 2024.
- [Sawadpong et al. 2012] SAWADPONG, P.; ALLEN, E. B. ; WILLIAMS, B. J.. **Exception handling defects: An empirical study**. In: 14TH INTERNATIONAL SYMPOSIUM ON HIGH-ASSURANCE SYSTEMS ENGINEERING, p. 90–97, 2012.
- [Shah et al. 2008] SHAH, H.; GÖRG, C. ; HARROLD, M. J.. **Why do developers neglect exception handling?** In: 4TH INTERNATIONAL WORKSHOP ON EXCEPTION HANDLING, p. 62–68. ACM, 2008.
- [Shah et al. 2010] SHAH, H.; GORG, C. ; HARROLD, M. J.. **Understanding exception handling: Viewpoints of novices and experts**. IEEE Transactions on Software Engineering, 36(2):150–161, 2010.
- [Shahrokni et al. 2013] SHAHROKNI, A.; FELDT, R.. **A systematic review of software robustness**. Information and Software Technology, 55(1):1–17, 2013.
- [Shorten et al. 2024] SHORTEN, C.; PIERSE, C.; SMITH, T. B.; CARDENAS, E.; SHARMA, A.; TRENGROVE, J. ; VAN LUIJT, B.. **Structuredrag: Json response formatting with large language models**, 2024.
- [Sobania et al. 2025] SOBANIA, D.; BRIESCH, M.; HANNA, C. ; PETKE, J.. **An analysis of the automatic bug fixing performance of chatgpt**. p. 23–30, May 2023.

- [Stack Overflow 2023] OVERFLOW, S.. **Stack overflow developer survey 2023**. <https://survey.stackoverflow.co/2023/#ai>. (Accessed on 04/04/2024).
- [Steinmacher et al. 2012] STEINMACHER, I.; WIESE, I. S. ; GEROSA, M. A.. **Recommending mentors to software project newcomers**. In: 2012 THIRD INTERNATIONAL WORKSHOP ON RECOMMENDATION SYSTEMS FOR SOFTWARE ENGINEERING (RSSE), p. 63–67. IEEE, 2012.
- [Steinmacher et al. 2014] STEINMACHER, I.; SILVA, M. A. G. ; GEROSA, M. A.. **Barriers faced by newcomers to open source projects: A systematic review**. In: Corral, L.; Sillitti, A.; Succi, G.; Vlasenko, J. ; Wasserman, A. I., editors, OPEN SOURCE SOFTWARE: MOBILE OPEN SOURCE TECHNOLOGIES, p. 153–163, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
- [Steinmacher et al. 2015] STEINMACHER, I.; CONTE, T.; GEROSA, M. A. ; REDMILES, D.. **Social barriers faced by newcomers placing their first contribution in open source software projects**. In: PROCEEDINGS OF THE 18TH ACM CONFERENCE ON COMPUTER SUPPORTED COOPERATIVE WORK & SOCIAL COMPUTING, CSCW '15, p. 1379–1392, New York, NY, USA, 2015. Association for Computing Machinery.
- [Steinmacher et al. 2016] STEINMACHER, I.; CONTE, T. U.; TREUDE, C. ; GEROSA, M. A.. **Overcoming open source project entry barriers with a portal for newcomers**. In: PROCEEDINGS OF THE 38TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, ICSE '16, p. 273–284, New York, NY, USA, 2016. Association for Computing Machinery.
- [Steinmacher et al. 2021] STEINMACHER, I.; BALALI, S.; TRINKENREICH, B.; GUIZANI, M.; IZQUIERDO-CORTAZAR, D.; CUEVAS ZAMBRANO, G. G.; GEROSA, M. A. ; SARMA, A.. **Being a mentor in open source projects**. *Journal of Internet Services and Applications*, 12:1–33, 2021.
- [Stenberg 2014] STENBERG, J.. **Exploring the hexagonal architecture**. *InfoQ* (Oct. 31, 2014). url: <https://www.infoq.com/news/2014/10/exploring-hexagonal-architecture/>(visited on 08/12/2019), 2014.

- [Strubell et al. 2019] STRUBELL, E.; GANESH, A. ; MCCALLUM, A.. **Energy and policy considerations for deep learning in nlp**. In: PROCEEDINGS OF THE 57TH ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS, p. 3645–3650, 2019.
- [Treude et al. 2011] TREUDE, C.; BARZILAY, O. ; STOREY, M.-A.. **How do programmers ask and answer questions on the web?** In: PROCEEDINGS OF THE 33RD INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, NIER TRACK), p. 804–807, 2011.
- [Tripathi et al. 2025] TRIPATHI, T.; WADHWA, M.; DURRETT, G. ; NIEKUM, S.. **Pairwise or pointwise? evaluating feedback protocols for bias in llm-based evaluation**, 2025.
- [Tsay et al. 2014] TSAY, J.; DABBISH, L. ; HERBSLEB, J.. **Let’s talk about it: evaluating contributions through discussion in github**. In: PROCEEDINGS OF THE 22ND ACM SIGSOFT INTERNATIONAL SYMPOSIUM ON FOUNDATIONS OF SOFTWARE ENGINEERING, FSE 2014, p. 144–154, New York, NY, USA, 2014. Association for Computing Machinery.
- [Uchôa et al. 2020] UCHÔA, A.; BARBOSA, C.; OIZUMI, W.; BLENILIO, P.; LIMA, R.; GARCIA, A. ; BEZERRA, C.. **How does modern code review impact software design degradation? an in-depth empirical study**. In: 2020 IEEE INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE AND EVOLUTION (ICSME), p. 511–522, 2020.
- [Vaswani et al. 2017] VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, L. U. ; POLOSUKHIN, I.. **Attention is all you need**. In: Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S. ; Garnett, R., editors, ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, volumen 30. Curran Associates, Inc., 2017.
- [Viviani et al. 2018] VIVIANI, G.; JANIK-JONES, C.; FAMELIS, M.; XIA, X. ; MURPHY, G. C.. **What design topics do developers discuss?** In: 26TH CONFERENCE ON PROGRAM COMPREHENSION, p. 328–331. ACM, 2018.
- [Viviani et al. 2021] VIVIANI, G.; FAMELIS, M.; XIA, X.; JANIK-JONES, C. ; MURPHY, G. C.. **Locating latent design information in developer discussions: A study on pull requests**. IEEE Transactions on Software Engineering, 47(7):1402–1413, 2021.

- [Wagner et al. 2020] WAGNER, S.; MENDEZ, D.; FELDERER, M.; GRAZIOTIN, D. ; KALINOWSKI, M.. **Challenges in Survey Research**, p. 93–125. Springer International Publishing, Cham, 2020.
- [Wang et al. 2024] WANG, W.; NING, H.; ZHANG, G.; LIU, L. ; WANG, Y.. **Rocks coding, not development—a human-centric, experimental evaluation of llm-supported se tasks**. arXiv preprint arXiv:2402.05650, 2024.
- [Wei et al. 2022] WEI, J.; TAY, Y.; BOMMASANI, R. ; OTHERS. **Emergent abilities of large language models**. arXiv preprint arXiv:2206.07682, 2022.
- [Wohlin et al. 2012] WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M.; REGNELL, B. ; WESSLÉN, A.. **Experimentation in Software Engineering**. Springer Science & Business Media, 1st edition, 2012.
- [Wu et al. 2024] WU, D.; MU, F.; SHI, L.; GUO, Z.; LIU, K.; ZHUANG, W.; ZHONG, Y. ; ZHANG, L.. **ismell: Assembling llms with expert toolsets for code smell detection and refactoring**. In: PROCEEDINGS OF THE 39TH IEEE/ACM INTERNATIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGINEERING (ASE). ACM, 2024.
- [Xiao et al. 2022] XIAO, W.; HE, H.; XU, W.; TAN, X.; DONG, J. ; ZHOU, M.. **Recommending good first issues in GitHub OSS projects**. In: PROCEEDINGS OF THE 44TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, p. 1830–1842, 2022.
- [Xu et al. 2024] XU, Z.; CRUZ, M. J.; GUEVARA, M.; WANG, T.; DESHPANDE, M.; WANG, X. ; LI, Z.. **Retrieval-augmented generation with knowledge graphs for customer service question answering**. Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2024.
- [Yan et al. 2023] YAN, A.; ZHONG, H.; SONG, D. ; JIA, L.. **How do programmers fix bugs as workarounds? an empirical study on apache projects**. Empirical Softw. Engg., 28(4), June 2023.
- [Zaiontz 2024 (Cliff’s Delta)] ZAIONTZ, C.. **Cliff’s delta**, 2024. Accessed: 2024-06-18.
- [Zhang et al. 2025] ZHANG, X.; CHEN, Y.; YUAN, Y. ; HUANG, M.. **Towards exception safety code generation with intermediate representation agents framework**, 2025.

- [Zheng et al. 2023] ZHENG, L.; CHIANG, W.-L.; SHENG, Y.; ZHUANG, S.; WU, Z.; ZHUANG, Y.; LIN, Z.; LI, Z.; LI, D.; XING, E. P.; ZHANG, H.; GONZALEZ, J. E. ; STOICA, I.. **Judging llm-as-a-judge with mt-bench and chatbot arena**. In: PROCEEDINGS OF THE 37TH INTERNATIONAL CONFERENCE ON NEURAL INFORMATION PROCESSING SYSTEMS, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.
- [Zhong and Mei 2018] ZHONG, H.; MEI, H.. **Mining repair model for exception-related bug**. *Journal of Systems and Software*, 141:16–31, 2018.
- [Zhou et al. 2023] ZHOU, K.; ZHENG, J.; WANG, H. ; OTHERS. **A comprehensive survey on pretrained foundation models: A history from bert to chatgpt**. *arXiv preprint arXiv:2302.09419*, 2023.
- [Zhou et al. 2024] ZHOU, X.; CAO, S.; SUN, X. ; LO, D.. **Large language model for vulnerability detection and repair: Literature review and the road ahead**, May 2025.
- [Zhou et al. 2024] ZHOU, X.; XU, B.; KIM, K.; HAN, D.; NGUYEN, H. H.; LE-CONG, T.; HE, J.; LE, B. ; LO, D.. **Leveraging large language model for automatic patch correctness assessment**. *IEEE Transactions on Software Engineering*, 50(11):2865–2883, 2024.
- [Zibaeirad et al. 2024] ZIBAEIRAD, A.; VIEIRA, M.. **Vulnllmeval: A framework for evaluating large language models in software vulnerability detection and patching**, 2024.

A

On the Investigation of Exception Pull Request Characteristics: Exploring the Apache Ecosystem

This appendix provides additional details regarding our first study (see Chapter 3). This study includes references to external links that provide access to the supplementary material presented in this section.

A.1

Full List of Exception-PRs Identified by the Heuristic in the Druid.

The spreadsheet below contains the full list of exception-PRs found by the heuristic in Druid.

PR	SYSTEM	URL
14355	apache/druid	https://github.com/apache/druid/pull/14355
14271	apache/druid	https://github.com/apache/druid/pull/14271
14209	apache/druid	https://github.com/apache/druid/pull/14209
14183	apache/druid	https://github.com/apache/druid/pull/14183
14171	apache/druid	https://github.com/apache/druid/pull/14171
14122	apache/druid	https://github.com/apache/druid/pull/14122
13846	apache/druid	https://github.com/apache/druid/pull/13846
13798	apache/druid	https://github.com/apache/druid/pull/13798
13794	apache/druid	https://github.com/apache/druid/pull/13794
13670	apache/druid	https://github.com/apache/druid/pull/13670
13668	apache/druid	https://github.com/apache/druid/pull/13668
13609	apache/druid	https://github.com/apache/druid/pull/13609
13441	apache/druid	https://github.com/apache/druid/pull/13441
13366	apache/druid	https://github.com/apache/druid/pull/13366
13286	apache/druid	https://github.com/apache/druid/pull/13286
13217	apache/druid	https://github.com/apache/druid/pull/13217
13089	apache/druid	https://github.com/apache/druid/pull/13089
13022	apache/druid	https://github.com/apache/druid/pull/13022
12967	apache/druid	https://github.com/apache/druid/pull/12967
12953	apache/druid	https://github.com/apache/druid/pull/12953
12952	apache/druid	https://github.com/apache/druid/pull/12952
12810	apache/druid	https://github.com/apache/druid/pull/12810
12680	apache/druid	https://github.com/apache/druid/pull/12680
12635	apache/druid	https://github.com/apache/druid/pull/12635
12626	apache/druid	https://github.com/apache/druid/pull/12626
12335	apache/druid	https://github.com/apache/druid/pull/12335
12259	apache/druid	https://github.com/apache/druid/pull/12259
12216	apache/druid	https://github.com/apache/druid/pull/12216
12213	apache/druid	https://github.com/apache/druid/pull/12213

12205	apache/druid	https://github.com/apache/druid/pull/12205
12161	apache/druid	https://github.com/apache/druid/pull/12161
12100	apache/druid	https://github.com/apache/druid/pull/12100
12080	apache/druid	https://github.com/apache/druid/pull/12080
12027	apache/druid	https://github.com/apache/druid/pull/12027
11961	apache/druid	https://github.com/apache/druid/pull/11961
11951	apache/druid	https://github.com/apache/druid/pull/11951
11892	apache/druid	https://github.com/apache/druid/pull/11892
11843	apache/druid	https://github.com/apache/druid/pull/11843
11752	apache/druid	https://github.com/apache/druid/pull/11752
11723	apache/druid	https://github.com/apache/druid/pull/11723
11715	apache/druid	https://github.com/apache/druid/pull/11715
11556	apache/druid	https://github.com/apache/druid/pull/11556
11430	apache/druid	https://github.com/apache/druid/pull/11430
11369	apache/druid	https://github.com/apache/druid/pull/11369
11325	apache/druid	https://github.com/apache/druid/pull/11325
11311	apache/druid	https://github.com/apache/druid/pull/11311
11266	apache/druid	https://github.com/apache/druid/pull/11266
11265	apache/druid	https://github.com/apache/druid/pull/11265
11041	apache/druid	https://github.com/apache/druid/pull/11041
10996	apache/druid	https://github.com/apache/druid/pull/10996
10976	apache/druid	https://github.com/apache/druid/pull/10976
10955	apache/druid	https://github.com/apache/druid/pull/10955
10950	apache/druid	https://github.com/apache/druid/pull/10950
10899	apache/druid	https://github.com/apache/druid/pull/10899
10722	apache/druid	https://github.com/apache/druid/pull/10722
10598	apache/druid	https://github.com/apache/druid/pull/10598
10555	apache/druid	https://github.com/apache/druid/pull/10555
10464	apache/druid	https://github.com/apache/druid/pull/10464
10336	apache/druid	https://github.com/apache/druid/pull/10336

10183	apache/druid	https://github.com/apache/druid/pull/10183
9547	apache/druid	https://github.com/apache/druid/pull/9547
9390	apache/druid	https://github.com/apache/druid/pull/9390
9044	apache/druid	https://github.com/apache/druid/pull/9044
8956	apache/druid	https://github.com/apache/druid/pull/8956
8692	apache/druid	https://github.com/apache/druid/pull/8692
8409	apache/druid	https://github.com/apache/druid/pull/8409
8391	apache/druid	https://github.com/apache/druid/pull/8391
8205	apache/druid	https://github.com/apache/druid/pull/8205
7933	apache/druid	https://github.com/apache/druid/pull/7933
7648	apache/druid	https://github.com/apache/druid/pull/7648
7504	apache/druid	https://github.com/apache/druid/pull/7504
7309	apache/druid	https://github.com/apache/druid/pull/7309
7248	apache/druid	https://github.com/apache/druid/pull/7248
7129	apache/druid	https://github.com/apache/druid/pull/7129
6690	apache/druid	https://github.com/apache/druid/pull/6690
6649	apache/druid	https://github.com/apache/druid/pull/6649
6538	apache/druid	https://github.com/apache/druid/pull/6538
6516	apache/druid	https://github.com/apache/druid/pull/6516
6461	apache/druid	https://github.com/apache/druid/pull/6461
6456	apache/druid	https://github.com/apache/druid/pull/6456
6178	apache/druid	https://github.com/apache/druid/pull/6178
6050	apache/druid	https://github.com/apache/druid/pull/6050
5988	apache/druid	https://github.com/apache/druid/pull/5988
5932	apache/druid	https://github.com/apache/druid/pull/5932
5724	apache/druid	https://github.com/apache/druid/pull/5724
5693	apache/druid	https://github.com/apache/druid/pull/5693
5630	apache/druid	https://github.com/apache/druid/pull/5630
5461	apache/druid	https://github.com/apache/druid/pull/5461
5374	apache/druid	https://github.com/apache/druid/pull/5374

5312	apache/druid	https://github.com/apache/druid/pull/5312
5146	apache/druid	https://github.com/apache/druid/pull/5146
4905	apache/druid	https://github.com/apache/druid/pull/4905
4776	apache/druid	https://github.com/apache/druid/pull/4776
4775	apache/druid	https://github.com/apache/druid/pull/4775
4759	apache/druid	https://github.com/apache/druid/pull/4759
4642	apache/druid	https://github.com/apache/druid/pull/4642
4467	apache/druid	https://github.com/apache/druid/pull/4467
4179	apache/druid	https://github.com/apache/druid/pull/4179
4130	apache/druid	https://github.com/apache/druid/pull/4130
4127	apache/druid	https://github.com/apache/druid/pull/4127
4100	apache/druid	https://github.com/apache/druid/pull/4100
3844	apache/druid	https://github.com/apache/druid/pull/3844
3668	apache/druid	https://github.com/apache/druid/pull/3668
3544	apache/druid	https://github.com/apache/druid/pull/3544
3504	apache/druid	https://github.com/apache/druid/pull/3504
3310	apache/druid	https://github.com/apache/druid/pull/3310
3278	apache/druid	https://github.com/apache/druid/pull/3278
2631	apache/druid	https://github.com/apache/druid/pull/2631
2620	apache/druid	https://github.com/apache/druid/pull/2620
2540	apache/druid	https://github.com/apache/druid/pull/2540
2530	apache/druid	https://github.com/apache/druid/pull/2530
2519	apache/druid	https://github.com/apache/druid/pull/2519
2518	apache/druid	https://github.com/apache/druid/pull/2518
2303	apache/druid	https://github.com/apache/druid/pull/2303
2264	apache/druid	https://github.com/apache/druid/pull/2264
1777	apache/druid	https://github.com/apache/druid/pull/1777
1773	apache/druid	https://github.com/apache/druid/pull/1773
1468	apache/druid	https://github.com/apache/druid/pull/1468
1303	apache/druid	https://github.com/apache/druid/pull/1303

1006	apache/druid	https://github.com/apache/druid/pull/1006
943	apache/druid	https://github.com/apache/druid/pull/943
771	apache/druid	https://github.com/apache/druid/pull/771
573	apache/druid	https://github.com/apache/druid/pull/573
234	apache/druid	https://github.com/apache/druid/pull/234
212	apache/druid	https://github.com/apache/druid/pull/212
45	apache/druid	https://github.com/apache/druid/pull/45

A.2

Full List of Exception-PRs Identified by the Heuristic in the Pulsar.

The following file contains the full list of exception-PRs found by the heuristic in Pulsar.

PR	SYSTEM	LINK
21079	apache/pulsar	https://github.com/apache/pulsar/pull/21079
20971	apache/pulsar	https://github.com/apache/pulsar/pull/20971
20905	apache/pulsar	https://github.com/apache/pulsar/pull/20905
20887	apache/pulsar	https://github.com/apache/pulsar/pull/20887
20847	apache/pulsar	https://github.com/apache/pulsar/pull/20847
20846	apache/pulsar	https://github.com/apache/pulsar/pull/20846
20818	apache/pulsar	https://github.com/apache/pulsar/pull/20818
20816	apache/pulsar	https://github.com/apache/pulsar/pull/20816
20795	apache/pulsar	https://github.com/apache/pulsar/pull/20795
20761	apache/pulsar	https://github.com/apache/pulsar/pull/20761
20730	apache/pulsar	https://github.com/apache/pulsar/pull/20730
20689	apache/pulsar	https://github.com/apache/pulsar/pull/20689
20613	apache/pulsar	https://github.com/apache/pulsar/pull/20613
20539	apache/pulsar	https://github.com/apache/pulsar/pull/20539
20424	apache/pulsar	https://github.com/apache/pulsar/pull/20424
20396	apache/pulsar	https://github.com/apache/pulsar/pull/20396
20371	apache/pulsar	https://github.com/apache/pulsar/pull/20371
20349	apache/pulsar	https://github.com/apache/pulsar/pull/20349
20261	apache/pulsar	https://github.com/apache/pulsar/pull/20261
20115	apache/pulsar	https://github.com/apache/pulsar/pull/20115
20082	apache/pulsar	https://github.com/apache/pulsar/pull/20082
19990	apache/pulsar	https://github.com/apache/pulsar/pull/19990
19811	apache/pulsar	https://github.com/apache/pulsar/pull/19811
19746	apache/pulsar	https://github.com/apache/pulsar/pull/19746
19633	apache/pulsar	https://github.com/apache/pulsar/pull/19633
19458	apache/pulsar	https://github.com/apache/pulsar/pull/19458
19381	apache/pulsar	https://github.com/apache/pulsar/pull/19381
19368	apache/pulsar	https://github.com/apache/pulsar/pull/19368
19356	apache/pulsar	https://github.com/apache/pulsar/pull/19356

19284	apache/pulsar	https://github.com/apache/pulsar/pull/19284
19273	apache/pulsar	https://github.com/apache/pulsar/pull/19273
19258	apache/pulsar	https://github.com/apache/pulsar/pull/19258
19147	apache/pulsar	https://github.com/apache/pulsar/pull/19147
19132	apache/pulsar	https://github.com/apache/pulsar/pull/19132
19055	apache/pulsar	https://github.com/apache/pulsar/pull/19055
18968	apache/pulsar	https://github.com/apache/pulsar/pull/18968
18837	apache/pulsar	https://github.com/apache/pulsar/pull/18837
18579	apache/pulsar	https://github.com/apache/pulsar/pull/18579
18540	apache/pulsar	https://github.com/apache/pulsar/pull/18540
18529	apache/pulsar	https://github.com/apache/pulsar/pull/18529
18502	apache/pulsar	https://github.com/apache/pulsar/pull/18502
18363	apache/pulsar	https://github.com/apache/pulsar/pull/18363
18359	apache/pulsar	https://github.com/apache/pulsar/pull/18359
18345	apache/pulsar	https://github.com/apache/pulsar/pull/18345
18276	apache/pulsar	https://github.com/apache/pulsar/pull/18276
18268	apache/pulsar	https://github.com/apache/pulsar/pull/18268
18211	apache/pulsar	https://github.com/apache/pulsar/pull/18211
17910	apache/pulsar	https://github.com/apache/pulsar/pull/17910
17841	apache/pulsar	https://github.com/apache/pulsar/pull/17841
17736	apache/pulsar	https://github.com/apache/pulsar/pull/17736
17717	apache/pulsar	https://github.com/apache/pulsar/pull/17717
17700	apache/pulsar	https://github.com/apache/pulsar/pull/17700
17600	apache/pulsar	https://github.com/apache/pulsar/pull/17600
17556	apache/pulsar	https://github.com/apache/pulsar/pull/17556
17512	apache/pulsar	https://github.com/apache/pulsar/pull/17512
17484	apache/pulsar	https://github.com/apache/pulsar/pull/17484
17394	apache/pulsar	https://github.com/apache/pulsar/pull/17394
17374	apache/pulsar	https://github.com/apache/pulsar/pull/17374
17370	apache/pulsar	https://github.com/apache/pulsar/pull/17370

17293	apache/pulsar	https://github.com/apache/pulsar/pull/17293
17276	apache/pulsar	https://github.com/apache/pulsar/pull/17276
17251	apache/pulsar	https://github.com/apache/pulsar/pull/17251
17249	apache/pulsar	https://github.com/apache/pulsar/pull/17249
16953	apache/pulsar	https://github.com/apache/pulsar/pull/16953
16848	apache/pulsar	https://github.com/apache/pulsar/pull/16848
16821	apache/pulsar	https://github.com/apache/pulsar/pull/16821
16788	apache/pulsar	https://github.com/apache/pulsar/pull/16788
16690	apache/pulsar	https://github.com/apache/pulsar/pull/16690
16678	apache/pulsar	https://github.com/apache/pulsar/pull/16678
16655	apache/pulsar	https://github.com/apache/pulsar/pull/16655
16628	apache/pulsar	https://github.com/apache/pulsar/pull/16628
16612	apache/pulsar	https://github.com/apache/pulsar/pull/16612
16594	apache/pulsar	https://github.com/apache/pulsar/pull/16594
16573	apache/pulsar	https://github.com/apache/pulsar/pull/16573
16473	apache/pulsar	https://github.com/apache/pulsar/pull/16473
16459	apache/pulsar	https://github.com/apache/pulsar/pull/16459
16420	apache/pulsar	https://github.com/apache/pulsar/pull/16420
16337	apache/pulsar	https://github.com/apache/pulsar/pull/16337
16265	apache/pulsar	https://github.com/apache/pulsar/pull/16265
16228	apache/pulsar	https://github.com/apache/pulsar/pull/16228
16131	apache/pulsar	https://github.com/apache/pulsar/pull/16131
16001	apache/pulsar	https://github.com/apache/pulsar/pull/16001
15988	apache/pulsar	https://github.com/apache/pulsar/pull/15988
15733	apache/pulsar	https://github.com/apache/pulsar/pull/15733
15677	apache/pulsar	https://github.com/apache/pulsar/pull/15677
15658	apache/pulsar	https://github.com/apache/pulsar/pull/15658
15633	apache/pulsar	https://github.com/apache/pulsar/pull/15633
15598	apache/pulsar	https://github.com/apache/pulsar/pull/15598
15534	apache/pulsar	https://github.com/apache/pulsar/pull/15534

15529	apache/pulsar	https://github.com/apache/pulsar/pull/15529
15414	apache/pulsar	https://github.com/apache/pulsar/pull/15414
15361	apache/pulsar	https://github.com/apache/pulsar/pull/15361
15089	apache/pulsar	https://github.com/apache/pulsar/pull/15089
15066	apache/pulsar	https://github.com/apache/pulsar/pull/15066
15002	apache/pulsar	https://github.com/apache/pulsar/pull/15002
14903	apache/pulsar	https://github.com/apache/pulsar/pull/14903
14810	apache/pulsar	https://github.com/apache/pulsar/pull/14810
14808	apache/pulsar	https://github.com/apache/pulsar/pull/14808
14807	apache/pulsar	https://github.com/apache/pulsar/pull/14807
14781	apache/pulsar	https://github.com/apache/pulsar/pull/14781
14709	apache/pulsar	https://github.com/apache/pulsar/pull/14709
14612	apache/pulsar	https://github.com/apache/pulsar/pull/14612
14602	apache/pulsar	https://github.com/apache/pulsar/pull/14602
14587	apache/pulsar	https://github.com/apache/pulsar/pull/14587
14553	apache/pulsar	https://github.com/apache/pulsar/pull/14553
14491	apache/pulsar	https://github.com/apache/pulsar/pull/14491
14460	apache/pulsar	https://github.com/apache/pulsar/pull/14460
14437	apache/pulsar	https://github.com/apache/pulsar/pull/14437
14410	apache/pulsar	https://github.com/apache/pulsar/pull/14410
14408	apache/pulsar	https://github.com/apache/pulsar/pull/14408
14388	apache/pulsar	https://github.com/apache/pulsar/pull/14388
14385	apache/pulsar	https://github.com/apache/pulsar/pull/14385
14316	apache/pulsar	https://github.com/apache/pulsar/pull/14316
14293	apache/pulsar	https://github.com/apache/pulsar/pull/14293
14280	apache/pulsar	https://github.com/apache/pulsar/pull/14280
14269	apache/pulsar	https://github.com/apache/pulsar/pull/14269
14214	apache/pulsar	https://github.com/apache/pulsar/pull/14214
14211	apache/pulsar	https://github.com/apache/pulsar/pull/14211
14210	apache/pulsar	https://github.com/apache/pulsar/pull/14210

14209	apache/pulsar	https://github.com/apache/pulsar/pull/14209
14207	apache/pulsar	https://github.com/apache/pulsar/pull/14207
14121	apache/pulsar	https://github.com/apache/pulsar/pull/14121
14025	apache/pulsar	https://github.com/apache/pulsar/pull/14025
14022	apache/pulsar	https://github.com/apache/pulsar/pull/14022
13914	apache/pulsar	https://github.com/apache/pulsar/pull/13914
13828	apache/pulsar	https://github.com/apache/pulsar/pull/13828
13553	apache/pulsar	https://github.com/apache/pulsar/pull/13553
13502	apache/pulsar	https://github.com/apache/pulsar/pull/13502
13462	apache/pulsar	https://github.com/apache/pulsar/pull/13462
13348	apache/pulsar	https://github.com/apache/pulsar/pull/13348
13245	apache/pulsar	https://github.com/apache/pulsar/pull/13245
13223	apache/pulsar	https://github.com/apache/pulsar/pull/13223
13203	apache/pulsar	https://github.com/apache/pulsar/pull/13203
13171	apache/pulsar	https://github.com/apache/pulsar/pull/13171
12899	apache/pulsar	https://github.com/apache/pulsar/pull/12899
12881	apache/pulsar	https://github.com/apache/pulsar/pull/12881
12853	apache/pulsar	https://github.com/apache/pulsar/pull/12853
12840	apache/pulsar	https://github.com/apache/pulsar/pull/12840
12828	apache/pulsar	https://github.com/apache/pulsar/pull/12828
12773	apache/pulsar	https://github.com/apache/pulsar/pull/12773
12767	apache/pulsar	https://github.com/apache/pulsar/pull/12767
12744	apache/pulsar	https://github.com/apache/pulsar/pull/12744
12725	apache/pulsar	https://github.com/apache/pulsar/pull/12725
12705	apache/pulsar	https://github.com/apache/pulsar/pull/12705
12700	apache/pulsar	https://github.com/apache/pulsar/pull/12700
12659	apache/pulsar	https://github.com/apache/pulsar/pull/12659
12647	apache/pulsar	https://github.com/apache/pulsar/pull/12647
12636	apache/pulsar	https://github.com/apache/pulsar/pull/12636
12441	apache/pulsar	https://github.com/apache/pulsar/pull/12441

12436	apache/pulsar	https://github.com/apache/pulsar/pull/12436
12376	apache/pulsar	https://github.com/apache/pulsar/pull/12376
12358	apache/pulsar	https://github.com/apache/pulsar/pull/12358
12345	apache/pulsar	https://github.com/apache/pulsar/pull/12345
12335	apache/pulsar	https://github.com/apache/pulsar/pull/12335
12170	apache/pulsar	https://github.com/apache/pulsar/pull/12170
12155	apache/pulsar	https://github.com/apache/pulsar/pull/12155
12005	apache/pulsar	https://github.com/apache/pulsar/pull/12005
11813	apache/pulsar	https://github.com/apache/pulsar/pull/11813
11811	apache/pulsar	https://github.com/apache/pulsar/pull/11811
11797	apache/pulsar	https://github.com/apache/pulsar/pull/11797
11788	apache/pulsar	https://github.com/apache/pulsar/pull/11788
11656	apache/pulsar	https://github.com/apache/pulsar/pull/11656
11655	apache/pulsar	https://github.com/apache/pulsar/pull/11655
11611	apache/pulsar	https://github.com/apache/pulsar/pull/11611
11478	apache/pulsar	https://github.com/apache/pulsar/pull/11478
11463	apache/pulsar	https://github.com/apache/pulsar/pull/11463
11329	apache/pulsar	https://github.com/apache/pulsar/pull/11329
11175	apache/pulsar	https://github.com/apache/pulsar/pull/11175
11068	apache/pulsar	https://github.com/apache/pulsar/pull/11068
10957	apache/pulsar	https://github.com/apache/pulsar/pull/10957
10931	apache/pulsar	https://github.com/apache/pulsar/pull/10931
10863	apache/pulsar	https://github.com/apache/pulsar/pull/10863
10797	apache/pulsar	https://github.com/apache/pulsar/pull/10797
10683	apache/pulsar	https://github.com/apache/pulsar/pull/10683
10549	apache/pulsar	https://github.com/apache/pulsar/pull/10549
10468	apache/pulsar	https://github.com/apache/pulsar/pull/10468
10416	apache/pulsar	https://github.com/apache/pulsar/pull/10416
10341	apache/pulsar	https://github.com/apache/pulsar/pull/10341
10290	apache/pulsar	https://github.com/apache/pulsar/pull/10290

10252	apache/pulsar	https://github.com/apache/pulsar/pull/10252
10251	apache/pulsar	https://github.com/apache/pulsar/pull/10251
10059	apache/pulsar	https://github.com/apache/pulsar/pull/10059
9738	apache/pulsar	https://github.com/apache/pulsar/pull/9738
9691	apache/pulsar	https://github.com/apache/pulsar/pull/9691
9468	apache/pulsar	https://github.com/apache/pulsar/pull/9468
9342	apache/pulsar	https://github.com/apache/pulsar/pull/9342
9228	apache/pulsar	https://github.com/apache/pulsar/pull/9228
9131	apache/pulsar	https://github.com/apache/pulsar/pull/9131
9091	apache/pulsar	https://github.com/apache/pulsar/pull/9091
9088	apache/pulsar	https://github.com/apache/pulsar/pull/9088
9031	apache/pulsar	https://github.com/apache/pulsar/pull/9031
9017	apache/pulsar	https://github.com/apache/pulsar/pull/9017
8931	apache/pulsar	https://github.com/apache/pulsar/pull/8931
8828	apache/pulsar	https://github.com/apache/pulsar/pull/8828
8736	apache/pulsar	https://github.com/apache/pulsar/pull/8736
8486	apache/pulsar	https://github.com/apache/pulsar/pull/8486
8199	apache/pulsar	https://github.com/apache/pulsar/pull/8199
8197	apache/pulsar	https://github.com/apache/pulsar/pull/8197
8143	apache/pulsar	https://github.com/apache/pulsar/pull/8143
8114	apache/pulsar	https://github.com/apache/pulsar/pull/8114
8024	apache/pulsar	https://github.com/apache/pulsar/pull/8024
8019	apache/pulsar	https://github.com/apache/pulsar/pull/8019
7798	apache/pulsar	https://github.com/apache/pulsar/pull/7798
7729	apache/pulsar	https://github.com/apache/pulsar/pull/7729
7725	apache/pulsar	https://github.com/apache/pulsar/pull/7725
7661	apache/pulsar	https://github.com/apache/pulsar/pull/7661
7632	apache/pulsar	https://github.com/apache/pulsar/pull/7632
7600	apache/pulsar	https://github.com/apache/pulsar/pull/7600
7514	apache/pulsar	https://github.com/apache/pulsar/pull/7514

7510	apache/pulsar	https://github.com/apache/pulsar/pull/7510
7483	apache/pulsar	https://github.com/apache/pulsar/pull/7483
7430	apache/pulsar	https://github.com/apache/pulsar/pull/7430
7401	apache/pulsar	https://github.com/apache/pulsar/pull/7401
7336	apache/pulsar	https://github.com/apache/pulsar/pull/7336
7335	apache/pulsar	https://github.com/apache/pulsar/pull/7335
7221	apache/pulsar	https://github.com/apache/pulsar/pull/7221
7179	apache/pulsar	https://github.com/apache/pulsar/pull/7179
7164	apache/pulsar	https://github.com/apache/pulsar/pull/7164
7100	apache/pulsar	https://github.com/apache/pulsar/pull/7100
7027	apache/pulsar	https://github.com/apache/pulsar/pull/7027
6910	apache/pulsar	https://github.com/apache/pulsar/pull/6910
6881	apache/pulsar	https://github.com/apache/pulsar/pull/6881
6849	apache/pulsar	https://github.com/apache/pulsar/pull/6849
6825	apache/pulsar	https://github.com/apache/pulsar/pull/6825
6683	apache/pulsar	https://github.com/apache/pulsar/pull/6683
6595	apache/pulsar	https://github.com/apache/pulsar/pull/6595
6475	apache/pulsar	https://github.com/apache/pulsar/pull/6475
6227	apache/pulsar	https://github.com/apache/pulsar/pull/6227
5870	apache/pulsar	https://github.com/apache/pulsar/pull/5870
5848	apache/pulsar	https://github.com/apache/pulsar/pull/5848
5643	apache/pulsar	https://github.com/apache/pulsar/pull/5643
5563	apache/pulsar	https://github.com/apache/pulsar/pull/5563
5444	apache/pulsar	https://github.com/apache/pulsar/pull/5444
5331	apache/pulsar	https://github.com/apache/pulsar/pull/5331
5174	apache/pulsar	https://github.com/apache/pulsar/pull/5174
5145	apache/pulsar	https://github.com/apache/pulsar/pull/5145
5068	apache/pulsar	https://github.com/apache/pulsar/pull/5068
4860	apache/pulsar	https://github.com/apache/pulsar/pull/4860
4568	apache/pulsar	https://github.com/apache/pulsar/pull/4568

4493	apache/pulsar	https://github.com/apache/pulsar/pull/4493
4460	apache/pulsar	https://github.com/apache/pulsar/pull/4460
4452	apache/pulsar	https://github.com/apache/pulsar/pull/4452
4402	apache/pulsar	https://github.com/apache/pulsar/pull/4402
4305	apache/pulsar	https://github.com/apache/pulsar/pull/4305
4185	apache/pulsar	https://github.com/apache/pulsar/pull/4185
4116	apache/pulsar	https://github.com/apache/pulsar/pull/4116
4006	apache/pulsar	https://github.com/apache/pulsar/pull/4006
3950	apache/pulsar	https://github.com/apache/pulsar/pull/3950
3942	apache/pulsar	https://github.com/apache/pulsar/pull/3942
3917	apache/pulsar	https://github.com/apache/pulsar/pull/3917
3754	apache/pulsar	https://github.com/apache/pulsar/pull/3754
3747	apache/pulsar	https://github.com/apache/pulsar/pull/3747
3532	apache/pulsar	https://github.com/apache/pulsar/pull/3532
3521	apache/pulsar	https://github.com/apache/pulsar/pull/3521
3434	apache/pulsar	https://github.com/apache/pulsar/pull/3434
3245	apache/pulsar	https://github.com/apache/pulsar/pull/3245
3142	apache/pulsar	https://github.com/apache/pulsar/pull/3142
3108	apache/pulsar	https://github.com/apache/pulsar/pull/3108
3107	apache/pulsar	https://github.com/apache/pulsar/pull/3107
2993	apache/pulsar	https://github.com/apache/pulsar/pull/2993
2710	apache/pulsar	https://github.com/apache/pulsar/pull/2710
2646	apache/pulsar	https://github.com/apache/pulsar/pull/2646
2644	apache/pulsar	https://github.com/apache/pulsar/pull/2644
2632	apache/pulsar	https://github.com/apache/pulsar/pull/2632
2353	apache/pulsar	https://github.com/apache/pulsar/pull/2353
2269	apache/pulsar	https://github.com/apache/pulsar/pull/2269
2249	apache/pulsar	https://github.com/apache/pulsar/pull/2249
1756	apache/pulsar	https://github.com/apache/pulsar/pull/1756
1744	apache/pulsar	https://github.com/apache/pulsar/pull/1744

1693	apache/pulsar	https://github.com/apache/pulsar/pull/1693
1673	apache/pulsar	https://github.com/apache/pulsar/pull/1673
1643	apache/pulsar	https://github.com/apache/pulsar/pull/1643
1630	apache/pulsar	https://github.com/apache/pulsar/pull/1630
1612	apache/pulsar	https://github.com/apache/pulsar/pull/1612
1602	apache/pulsar	https://github.com/apache/pulsar/pull/1602
1599	apache/pulsar	https://github.com/apache/pulsar/pull/1599
1537	apache/pulsar	https://github.com/apache/pulsar/pull/1537
1492	apache/pulsar	https://github.com/apache/pulsar/pull/1492
1222	apache/pulsar	https://github.com/apache/pulsar/pull/1222
1179	apache/pulsar	https://github.com/apache/pulsar/pull/1179
1174	apache/pulsar	https://github.com/apache/pulsar/pull/1174
1107	apache/pulsar	https://github.com/apache/pulsar/pull/1107
1040	apache/pulsar	https://github.com/apache/pulsar/pull/1040
938	apache/pulsar	https://github.com/apache/pulsar/pull/938
906	apache/pulsar	https://github.com/apache/pulsar/pull/906
902	apache/pulsar	https://github.com/apache/pulsar/pull/902
852	apache/pulsar	https://github.com/apache/pulsar/pull/852
832	apache/pulsar	https://github.com/apache/pulsar/pull/832
673	apache/pulsar	https://github.com/apache/pulsar/pull/673
513	apache/pulsar	https://github.com/apache/pulsar/pull/513
374	apache/pulsar	https://github.com/apache/pulsar/pull/374
353	apache/pulsar	https://github.com/apache/pulsar/pull/353
270	apache/pulsar	https://github.com/apache/pulsar/pull/270
220	apache/pulsar	https://github.com/apache/pulsar/pull/220

A.3

Full List of Exception-PRs Identified by the Heuristic in the Flink.

Following, we have the full list of exception-PRs found by the heuristic in Flink.

PR	SYSTEM	LINK
22643	apache/flink	https://github.com/apache/flink/pull/22643
22544	apache/flink	https://github.com/apache/flink/pull/22544
22150	apache/flink	https://github.com/apache/flink/pull/22150
22125	apache/flink	https://github.com/apache/flink/pull/22125
22112	apache/flink	https://github.com/apache/flink/pull/22112
22110	apache/flink	https://github.com/apache/flink/pull/22110
22099	apache/flink	https://github.com/apache/flink/pull/22099
22055	apache/flink	https://github.com/apache/flink/pull/22055
22036	apache/flink	https://github.com/apache/flink/pull/22036
22035	apache/flink	https://github.com/apache/flink/pull/22035
22007	apache/flink	https://github.com/apache/flink/pull/22007
21999	apache/flink	https://github.com/apache/flink/pull/21999
21948	apache/flink	https://github.com/apache/flink/pull/21948
21907	apache/flink	https://github.com/apache/flink/pull/21907
21903	apache/flink	https://github.com/apache/flink/pull/21903
21891	apache/flink	https://github.com/apache/flink/pull/21891
21867	apache/flink	https://github.com/apache/flink/pull/21867
21858	apache/flink	https://github.com/apache/flink/pull/21858
21821	apache/flink	https://github.com/apache/flink/pull/21821
21820	apache/flink	https://github.com/apache/flink/pull/21820
21784	apache/flink	https://github.com/apache/flink/pull/21784
21783	apache/flink	https://github.com/apache/flink/pull/21783
21782	apache/flink	https://github.com/apache/flink/pull/21782
21769	apache/flink	https://github.com/apache/flink/pull/21769
21732	apache/flink	https://github.com/apache/flink/pull/21732
21731	apache/flink	https://github.com/apache/flink/pull/21731
21559	apache/flink	https://github.com/apache/flink/pull/21559
21515	apache/flink	https://github.com/apache/flink/pull/21515
21483	apache/flink	https://github.com/apache/flink/pull/21483

21463	apache/flink	https://github.com/apache/flink/pull/21463
21456	apache/flink	https://github.com/apache/flink/pull/21456
21440	apache/flink	https://github.com/apache/flink/pull/21440
21429	apache/flink	https://github.com/apache/flink/pull/21429
21428	apache/flink	https://github.com/apache/flink/pull/21428
21379	apache/flink	https://github.com/apache/flink/pull/21379
21366	apache/flink	https://github.com/apache/flink/pull/21366
21331	apache/flink	https://github.com/apache/flink/pull/21331
21115	apache/flink	https://github.com/apache/flink/pull/21115
21113	apache/flink	https://github.com/apache/flink/pull/21113
21101	apache/flink	https://github.com/apache/flink/pull/21101
21016	apache/flink	https://github.com/apache/flink/pull/21016
20928	apache/flink	https://github.com/apache/flink/pull/20928
20912	apache/flink	https://github.com/apache/flink/pull/20912
20890	apache/flink	https://github.com/apache/flink/pull/20890
20805	apache/flink	https://github.com/apache/flink/pull/20805
20776	apache/flink	https://github.com/apache/flink/pull/20776
20764	apache/flink	https://github.com/apache/flink/pull/20764
20734	apache/flink	https://github.com/apache/flink/pull/20734
20707	apache/flink	https://github.com/apache/flink/pull/20707
20705	apache/flink	https://github.com/apache/flink/pull/20705
20691	apache/flink	https://github.com/apache/flink/pull/20691
20684	apache/flink	https://github.com/apache/flink/pull/20684
20666	apache/flink	https://github.com/apache/flink/pull/20666
20637	apache/flink	https://github.com/apache/flink/pull/20637
20590	apache/flink	https://github.com/apache/flink/pull/20590
20571	apache/flink	https://github.com/apache/flink/pull/20571
20530	apache/flink	https://github.com/apache/flink/pull/20530
20269	apache/flink	https://github.com/apache/flink/pull/20269
20108	apache/flink	https://github.com/apache/flink/pull/20108

20071	apache/flink	https://github.com/apache/flink/pull/20071
20062	apache/flink	https://github.com/apache/flink/pull/20062
20060	apache/flink	https://github.com/apache/flink/pull/20060
19963	apache/flink	https://github.com/apache/flink/pull/19963
19933	apache/flink	https://github.com/apache/flink/pull/19933
19828	apache/flink	https://github.com/apache/flink/pull/19828
19817	apache/flink	https://github.com/apache/flink/pull/19817
19816	apache/flink	https://github.com/apache/flink/pull/19816
19796	apache/flink	https://github.com/apache/flink/pull/19796
19793	apache/flink	https://github.com/apache/flink/pull/19793
19773	apache/flink	https://github.com/apache/flink/pull/19773
19715	apache/flink	https://github.com/apache/flink/pull/19715
19629	apache/flink	https://github.com/apache/flink/pull/19629
19615	apache/flink	https://github.com/apache/flink/pull/19615
19614	apache/flink	https://github.com/apache/flink/pull/19614
19596	apache/flink	https://github.com/apache/flink/pull/19596
19583	apache/flink	https://github.com/apache/flink/pull/19583
19481	apache/flink	https://github.com/apache/flink/pull/19481
19456	apache/flink	https://github.com/apache/flink/pull/19456
19400	apache/flink	https://github.com/apache/flink/pull/19400
19298	apache/flink	https://github.com/apache/flink/pull/19298
19241	apache/flink	https://github.com/apache/flink/pull/19241
19240	apache/flink	https://github.com/apache/flink/pull/19240
19219	apache/flink	https://github.com/apache/flink/pull/19219
19216	apache/flink	https://github.com/apache/flink/pull/19216
19182	apache/flink	https://github.com/apache/flink/pull/19182
19160	apache/flink	https://github.com/apache/flink/pull/19160
19043	apache/flink	https://github.com/apache/flink/pull/19043
19013	apache/flink	https://github.com/apache/flink/pull/19013
18994	apache/flink	https://github.com/apache/flink/pull/18994

18937	apache/flink	https://github.com/apache/flink/pull/18937
18920	apache/flink	https://github.com/apache/flink/pull/18920
18894	apache/flink	https://github.com/apache/flink/pull/18894
18866	apache/flink	https://github.com/apache/flink/pull/18866
18822	apache/flink	https://github.com/apache/flink/pull/18822
18716	apache/flink	https://github.com/apache/flink/pull/18716
18689	apache/flink	https://github.com/apache/flink/pull/18689
18610	apache/flink	https://github.com/apache/flink/pull/18610
18548	apache/flink	https://github.com/apache/flink/pull/18548
18515	apache/flink	https://github.com/apache/flink/pull/18515
18449	apache/flink	https://github.com/apache/flink/pull/18449
18404	apache/flink	https://github.com/apache/flink/pull/18404
18336	apache/flink	https://github.com/apache/flink/pull/18336
18219	apache/flink	https://github.com/apache/flink/pull/18219
18159	apache/flink	https://github.com/apache/flink/pull/18159
17990	apache/flink	https://github.com/apache/flink/pull/17990
17967	apache/flink	https://github.com/apache/flink/pull/17967
17908	apache/flink	https://github.com/apache/flink/pull/17908
17865	apache/flink	https://github.com/apache/flink/pull/17865
17794	apache/flink	https://github.com/apache/flink/pull/17794
17792	apache/flink	https://github.com/apache/flink/pull/17792
17791	apache/flink	https://github.com/apache/flink/pull/17791
17785	apache/flink	https://github.com/apache/flink/pull/17785
17772	apache/flink	https://github.com/apache/flink/pull/17772
17700	apache/flink	https://github.com/apache/flink/pull/17700
17689	apache/flink	https://github.com/apache/flink/pull/17689
17605	apache/flink	https://github.com/apache/flink/pull/17605
17584	apache/flink	https://github.com/apache/flink/pull/17584
17567	apache/flink	https://github.com/apache/flink/pull/17567
17547	apache/flink	https://github.com/apache/flink/pull/17547

17546	apache/flink	https://github.com/apache/flink/pull/17546
17529	apache/flink	https://github.com/apache/flink/pull/17529
17528	apache/flink	https://github.com/apache/flink/pull/17528
17497	apache/flink	https://github.com/apache/flink/pull/17497
17469	apache/flink	https://github.com/apache/flink/pull/17469
17460	apache/flink	https://github.com/apache/flink/pull/17460
17436	apache/flink	https://github.com/apache/flink/pull/17436
17409	apache/flink	https://github.com/apache/flink/pull/17409
17337	apache/flink	https://github.com/apache/flink/pull/17337
17336	apache/flink	https://github.com/apache/flink/pull/17336
17331	apache/flink	https://github.com/apache/flink/pull/17331
17321	apache/flink	https://github.com/apache/flink/pull/17321
17227	apache/flink	https://github.com/apache/flink/pull/17227
17169	apache/flink	https://github.com/apache/flink/pull/17169
17093	apache/flink	https://github.com/apache/flink/pull/17093
17078	apache/flink	https://github.com/apache/flink/pull/17078
17039	apache/flink	https://github.com/apache/flink/pull/17039
16954	apache/flink	https://github.com/apache/flink/pull/16954
16903	apache/flink	https://github.com/apache/flink/pull/16903
16893	apache/flink	https://github.com/apache/flink/pull/16893
16842	apache/flink	https://github.com/apache/flink/pull/16842
16841	apache/flink	https://github.com/apache/flink/pull/16841
16840	apache/flink	https://github.com/apache/flink/pull/16840
16839	apache/flink	https://github.com/apache/flink/pull/16839
16829	apache/flink	https://github.com/apache/flink/pull/16829
16789	apache/flink	https://github.com/apache/flink/pull/16789
16746	apache/flink	https://github.com/apache/flink/pull/16746
16731	apache/flink	https://github.com/apache/flink/pull/16731
16718	apache/flink	https://github.com/apache/flink/pull/16718
16662	apache/flink	https://github.com/apache/flink/pull/16662

16653	apache/flink	https://github.com/apache/flink/pull/16653
16619	apache/flink	https://github.com/apache/flink/pull/16619
16618	apache/flink	https://github.com/apache/flink/pull/16618
16617	apache/flink	https://github.com/apache/flink/pull/16617
16591	apache/flink	https://github.com/apache/flink/pull/16591
16570	apache/flink	https://github.com/apache/flink/pull/16570
16514	apache/flink	https://github.com/apache/flink/pull/16514
16506	apache/flink	https://github.com/apache/flink/pull/16506
16462	apache/flink	https://github.com/apache/flink/pull/16462
16446	apache/flink	https://github.com/apache/flink/pull/16446
16420	apache/flink	https://github.com/apache/flink/pull/16420
16318	apache/flink	https://github.com/apache/flink/pull/16318
16315	apache/flink	https://github.com/apache/flink/pull/16315
16305	apache/flink	https://github.com/apache/flink/pull/16305
16301	apache/flink	https://github.com/apache/flink/pull/16301
16276	apache/flink	https://github.com/apache/flink/pull/16276
16270	apache/flink	https://github.com/apache/flink/pull/16270
16259	apache/flink	https://github.com/apache/flink/pull/16259
16254	apache/flink	https://github.com/apache/flink/pull/16254
16194	apache/flink	https://github.com/apache/flink/pull/16194
16193	apache/flink	https://github.com/apache/flink/pull/16193
16127	apache/flink	https://github.com/apache/flink/pull/16127
16071	apache/flink	https://github.com/apache/flink/pull/16071
16068	apache/flink	https://github.com/apache/flink/pull/16068
16056	apache/flink	https://github.com/apache/flink/pull/16056
16049	apache/flink	https://github.com/apache/flink/pull/16049
16028	apache/flink	https://github.com/apache/flink/pull/16028
15978	apache/flink	https://github.com/apache/flink/pull/15978
15963	apache/flink	https://github.com/apache/flink/pull/15963
15945	apache/flink	https://github.com/apache/flink/pull/15945

15938	apache/flink	https://github.com/apache/flink/pull/15938
15832	apache/flink	https://github.com/apache/flink/pull/15832
15803	apache/flink	https://github.com/apache/flink/pull/15803
15800	apache/flink	https://github.com/apache/flink/pull/15800
15766	apache/flink	https://github.com/apache/flink/pull/15766
15765	apache/flink	https://github.com/apache/flink/pull/15765
15762	apache/flink	https://github.com/apache/flink/pull/15762
15740	apache/flink	https://github.com/apache/flink/pull/15740
15723	apache/flink	https://github.com/apache/flink/pull/15723
15714	apache/flink	https://github.com/apache/flink/pull/15714
15707	apache/flink	https://github.com/apache/flink/pull/15707
15688	apache/flink	https://github.com/apache/flink/pull/15688
15678	apache/flink	https://github.com/apache/flink/pull/15678
15663	apache/flink	https://github.com/apache/flink/pull/15663
15661	apache/flink	https://github.com/apache/flink/pull/15661
15650	apache/flink	https://github.com/apache/flink/pull/15650
15649	apache/flink	https://github.com/apache/flink/pull/15649
15641	apache/flink	https://github.com/apache/flink/pull/15641
15640	apache/flink	https://github.com/apache/flink/pull/15640
15615	apache/flink	https://github.com/apache/flink/pull/15615
15598	apache/flink	https://github.com/apache/flink/pull/15598
15591	apache/flink	https://github.com/apache/flink/pull/15591
15590	apache/flink	https://github.com/apache/flink/pull/15590
15577	apache/flink	https://github.com/apache/flink/pull/15577
15576	apache/flink	https://github.com/apache/flink/pull/15576
15543	apache/flink	https://github.com/apache/flink/pull/15543
15540	apache/flink	https://github.com/apache/flink/pull/15540
15523	apache/flink	https://github.com/apache/flink/pull/15523
15515	apache/flink	https://github.com/apache/flink/pull/15515
15504	apache/flink	https://github.com/apache/flink/pull/15504

15503	apache/flink	https://github.com/apache/flink/pull/15503
15486	apache/flink	https://github.com/apache/flink/pull/15486
15424	apache/flink	https://github.com/apache/flink/pull/15424
15418	apache/flink	https://github.com/apache/flink/pull/15418
15383	apache/flink	https://github.com/apache/flink/pull/15383
15377	apache/flink	https://github.com/apache/flink/pull/15377
15344	apache/flink	https://github.com/apache/flink/pull/15344
15311	apache/flink	https://github.com/apache/flink/pull/15311
15299	apache/flink	https://github.com/apache/flink/pull/15299
15297	apache/flink	https://github.com/apache/flink/pull/15297
15293	apache/flink	https://github.com/apache/flink/pull/15293
15289	apache/flink	https://github.com/apache/flink/pull/15289
15134	apache/flink	https://github.com/apache/flink/pull/15134
15074	apache/flink	https://github.com/apache/flink/pull/15074
15068	apache/flink	https://github.com/apache/flink/pull/15068
15049	apache/flink	https://github.com/apache/flink/pull/15049
15021	apache/flink	https://github.com/apache/flink/pull/15021
15010	apache/flink	https://github.com/apache/flink/pull/15010
14941	apache/flink	https://github.com/apache/flink/pull/14941
14801	apache/flink	https://github.com/apache/flink/pull/14801
14798	apache/flink	https://github.com/apache/flink/pull/14798
14686	apache/flink	https://github.com/apache/flink/pull/14686
14682	apache/flink	https://github.com/apache/flink/pull/14682
14666	apache/flink	https://github.com/apache/flink/pull/14666
14631	apache/flink	https://github.com/apache/flink/pull/14631
14452	apache/flink	https://github.com/apache/flink/pull/14452
14438	apache/flink	https://github.com/apache/flink/pull/14438
14332	apache/flink	https://github.com/apache/flink/pull/14332
14205	apache/flink	https://github.com/apache/flink/pull/14205
14203	apache/flink	https://github.com/apache/flink/pull/14203

14200	apache/flink	https://github.com/apache/flink/pull/14200
14198	apache/flink	https://github.com/apache/flink/pull/14198
14177	apache/flink	https://github.com/apache/flink/pull/14177
14164	apache/flink	https://github.com/apache/flink/pull/14164
14161	apache/flink	https://github.com/apache/flink/pull/14161
14160	apache/flink	https://github.com/apache/flink/pull/14160
14150	apache/flink	https://github.com/apache/flink/pull/14150
14104	apache/flink	https://github.com/apache/flink/pull/14104
14095	apache/flink	https://github.com/apache/flink/pull/14095
14093	apache/flink	https://github.com/apache/flink/pull/14093
14072	apache/flink	https://github.com/apache/flink/pull/14072
14071	apache/flink	https://github.com/apache/flink/pull/14071
14030	apache/flink	https://github.com/apache/flink/pull/14030
14028	apache/flink	https://github.com/apache/flink/pull/14028
13953	apache/flink	https://github.com/apache/flink/pull/13953
13917	apache/flink	https://github.com/apache/flink/pull/13917
13901	apache/flink	https://github.com/apache/flink/pull/13901
13840	apache/flink	https://github.com/apache/flink/pull/13840
13810	apache/flink	https://github.com/apache/flink/pull/13810
13776	apache/flink	https://github.com/apache/flink/pull/13776
13768	apache/flink	https://github.com/apache/flink/pull/13768
13723	apache/flink	https://github.com/apache/flink/pull/13723
13718	apache/flink	https://github.com/apache/flink/pull/13718
13679	apache/flink	https://github.com/apache/flink/pull/13679
13668	apache/flink	https://github.com/apache/flink/pull/13668
13634	apache/flink	https://github.com/apache/flink/pull/13634
13616	apache/flink	https://github.com/apache/flink/pull/13616
13600	apache/flink	https://github.com/apache/flink/pull/13600
13594	apache/flink	https://github.com/apache/flink/pull/13594
13571	apache/flink	https://github.com/apache/flink/pull/13571

13564	apache/flink	https://github.com/apache/flink/pull/13564
13545	apache/flink	https://github.com/apache/flink/pull/13545
13543	apache/flink	https://github.com/apache/flink/pull/13543
13496	apache/flink	https://github.com/apache/flink/pull/13496
13485	apache/flink	https://github.com/apache/flink/pull/13485
13481	apache/flink	https://github.com/apache/flink/pull/13481
13472	apache/flink	https://github.com/apache/flink/pull/13472
13444	apache/flink	https://github.com/apache/flink/pull/13444
13441	apache/flink	https://github.com/apache/flink/pull/13441
13426	apache/flink	https://github.com/apache/flink/pull/13426
13354	apache/flink	https://github.com/apache/flink/pull/13354
13315	apache/flink	https://github.com/apache/flink/pull/13315
13267	apache/flink	https://github.com/apache/flink/pull/13267
13214	apache/flink	https://github.com/apache/flink/pull/13214
13145	apache/flink	https://github.com/apache/flink/pull/13145
13104	apache/flink	https://github.com/apache/flink/pull/13104
13020	apache/flink	https://github.com/apache/flink/pull/13020
13019	apache/flink	https://github.com/apache/flink/pull/13019
12968	apache/flink	https://github.com/apache/flink/pull/12968
12963	apache/flink	https://github.com/apache/flink/pull/12963
12957	apache/flink	https://github.com/apache/flink/pull/12957
12949	apache/flink	https://github.com/apache/flink/pull/12949
12916	apache/flink	https://github.com/apache/flink/pull/12916
12848	apache/flink	https://github.com/apache/flink/pull/12848
12844	apache/flink	https://github.com/apache/flink/pull/12844
12827	apache/flink	https://github.com/apache/flink/pull/12827
12822	apache/flink	https://github.com/apache/flink/pull/12822
12806	apache/flink	https://github.com/apache/flink/pull/12806
12781	apache/flink	https://github.com/apache/flink/pull/12781
12769	apache/flink	https://github.com/apache/flink/pull/12769

12750	apache/flink	https://github.com/apache/flink/pull/12750
12726	apache/flink	https://github.com/apache/flink/pull/12726
12676	apache/flink	https://github.com/apache/flink/pull/12676
12627	apache/flink	https://github.com/apache/flink/pull/12627
12576	apache/flink	https://github.com/apache/flink/pull/12576
12488	apache/flink	https://github.com/apache/flink/pull/12488
12470	apache/flink	https://github.com/apache/flink/pull/12470
12459	apache/flink	https://github.com/apache/flink/pull/12459
12446	apache/flink	https://github.com/apache/flink/pull/12446
12438	apache/flink	https://github.com/apache/flink/pull/12438
12436	apache/flink	https://github.com/apache/flink/pull/12436
12432	apache/flink	https://github.com/apache/flink/pull/12432
12374	apache/flink	https://github.com/apache/flink/pull/12374
12303	apache/flink	https://github.com/apache/flink/pull/12303
12210	apache/flink	https://github.com/apache/flink/pull/12210
12158	apache/flink	https://github.com/apache/flink/pull/12158
12128	apache/flink	https://github.com/apache/flink/pull/12128
12052	apache/flink	https://github.com/apache/flink/pull/12052
12005	apache/flink	https://github.com/apache/flink/pull/12005
11665	apache/flink	https://github.com/apache/flink/pull/11665
11550	apache/flink	https://github.com/apache/flink/pull/11550
11509	apache/flink	https://github.com/apache/flink/pull/11509
11508	apache/flink	https://github.com/apache/flink/pull/11508
11480	apache/flink	https://github.com/apache/flink/pull/11480
11447	apache/flink	https://github.com/apache/flink/pull/11447
11436	apache/flink	https://github.com/apache/flink/pull/11436
11434	apache/flink	https://github.com/apache/flink/pull/11434
11414	apache/flink	https://github.com/apache/flink/pull/11414
11397	apache/flink	https://github.com/apache/flink/pull/11397
11396	apache/flink	https://github.com/apache/flink/pull/11396

11387	apache/flink	https://github.com/apache/flink/pull/11387
11380	apache/flink	https://github.com/apache/flink/pull/11380
11352	apache/flink	https://github.com/apache/flink/pull/11352
11311	apache/flink	https://github.com/apache/flink/pull/11311
11237	apache/flink	https://github.com/apache/flink/pull/11237
11234	apache/flink	https://github.com/apache/flink/pull/11234
11217	apache/flink	https://github.com/apache/flink/pull/11217
11180	apache/flink	https://github.com/apache/flink/pull/11180
11172	apache/flink	https://github.com/apache/flink/pull/11172
11161	apache/flink	https://github.com/apache/flink/pull/11161
11139	apache/flink	https://github.com/apache/flink/pull/11139
11048	apache/flink	https://github.com/apache/flink/pull/11048
11042	apache/flink	https://github.com/apache/flink/pull/11042
11032	apache/flink	https://github.com/apache/flink/pull/11032
11026	apache/flink	https://github.com/apache/flink/pull/11026
11011	apache/flink	https://github.com/apache/flink/pull/11011
11010	apache/flink	https://github.com/apache/flink/pull/11010
10983	apache/flink	https://github.com/apache/flink/pull/10983
10978	apache/flink	https://github.com/apache/flink/pull/10978
10970	apache/flink	https://github.com/apache/flink/pull/10970
10948	apache/flink	https://github.com/apache/flink/pull/10948
10943	apache/flink	https://github.com/apache/flink/pull/10943
10907	apache/flink	https://github.com/apache/flink/pull/10907
10872	apache/flink	https://github.com/apache/flink/pull/10872
10838	apache/flink	https://github.com/apache/flink/pull/10838
10779	apache/flink	https://github.com/apache/flink/pull/10779
10749	apache/flink	https://github.com/apache/flink/pull/10749
10726	apache/flink	https://github.com/apache/flink/pull/10726
10628	apache/flink	https://github.com/apache/flink/pull/10628
10615	apache/flink	https://github.com/apache/flink/pull/10615

10611	apache/flink	https://github.com/apache/flink/pull/10611
10596	apache/flink	https://github.com/apache/flink/pull/10596
10563	apache/flink	https://github.com/apache/flink/pull/10563
10412	apache/flink	https://github.com/apache/flink/pull/10412
10365	apache/flink	https://github.com/apache/flink/pull/10365
10339	apache/flink	https://github.com/apache/flink/pull/10339
10336	apache/flink	https://github.com/apache/flink/pull/10336
10335	apache/flink	https://github.com/apache/flink/pull/10335
10314	apache/flink	https://github.com/apache/flink/pull/10314
10282	apache/flink	https://github.com/apache/flink/pull/10282
10257	apache/flink	https://github.com/apache/flink/pull/10257
10175	apache/flink	https://github.com/apache/flink/pull/10175
10145	apache/flink	https://github.com/apache/flink/pull/10145
10133	apache/flink	https://github.com/apache/flink/pull/10133
10053	apache/flink	https://github.com/apache/flink/pull/10053
10045	apache/flink	https://github.com/apache/flink/pull/10045
10028	apache/flink	https://github.com/apache/flink/pull/10028
9880	apache/flink	https://github.com/apache/flink/pull/9880
9873	apache/flink	https://github.com/apache/flink/pull/9873
9859	apache/flink	https://github.com/apache/flink/pull/9859
9849	apache/flink	https://github.com/apache/flink/pull/9849
9823	apache/flink	https://github.com/apache/flink/pull/9823
9794	apache/flink	https://github.com/apache/flink/pull/9794
9777	apache/flink	https://github.com/apache/flink/pull/9777
9771	apache/flink	https://github.com/apache/flink/pull/9771
9742	apache/flink	https://github.com/apache/flink/pull/9742
9698	apache/flink	https://github.com/apache/flink/pull/9698
9641	apache/flink	https://github.com/apache/flink/pull/9641
9629	apache/flink	https://github.com/apache/flink/pull/9629
9475	apache/flink	https://github.com/apache/flink/pull/9475

9456	apache/flink	https://github.com/apache/flink/pull/9456
9268	apache/flink	https://github.com/apache/flink/pull/9268
9258	apache/flink	https://github.com/apache/flink/pull/9258
9115	apache/flink	https://github.com/apache/flink/pull/9115
9103	apache/flink	https://github.com/apache/flink/pull/9103
9090	apache/flink	https://github.com/apache/flink/pull/9090
9062	apache/flink	https://github.com/apache/flink/pull/9062
8981	apache/flink	https://github.com/apache/flink/pull/8981
8967	apache/flink	https://github.com/apache/flink/pull/8967
8948	apache/flink	https://github.com/apache/flink/pull/8948
8603	apache/flink	https://github.com/apache/flink/pull/8603
8574	apache/flink	https://github.com/apache/flink/pull/8574
8509	apache/flink	https://github.com/apache/flink/pull/8509
8334	apache/flink	https://github.com/apache/flink/pull/8334
8242	apache/flink	https://github.com/apache/flink/pull/8242
8237	apache/flink	https://github.com/apache/flink/pull/8237
8228	apache/flink	https://github.com/apache/flink/pull/8228
8182	apache/flink	https://github.com/apache/flink/pull/8182
8174	apache/flink	https://github.com/apache/flink/pull/8174
8080	apache/flink	https://github.com/apache/flink/pull/8080
7974	apache/flink	https://github.com/apache/flink/pull/7974
7965	apache/flink	https://github.com/apache/flink/pull/7965
7888	apache/flink	https://github.com/apache/flink/pull/7888
7886	apache/flink	https://github.com/apache/flink/pull/7886
7672	apache/flink	https://github.com/apache/flink/pull/7672
7552	apache/flink	https://github.com/apache/flink/pull/7552
7510	apache/flink	https://github.com/apache/flink/pull/7510
7374	apache/flink	https://github.com/apache/flink/pull/7374
7346	apache/flink	https://github.com/apache/flink/pull/7346
7290	apache/flink	https://github.com/apache/flink/pull/7290

7265	apache/flink	https://github.com/apache/flink/pull/7265
7258	apache/flink	https://github.com/apache/flink/pull/7258
7200	apache/flink	https://github.com/apache/flink/pull/7200
7183	apache/flink	https://github.com/apache/flink/pull/7183
7120	apache/flink	https://github.com/apache/flink/pull/7120
7115	apache/flink	https://github.com/apache/flink/pull/7115
7064	apache/flink	https://github.com/apache/flink/pull/7064
7033	apache/flink	https://github.com/apache/flink/pull/7033
7008	apache/flink	https://github.com/apache/flink/pull/7008
7005	apache/flink	https://github.com/apache/flink/pull/7005
6950	apache/flink	https://github.com/apache/flink/pull/6950
6932	apache/flink	https://github.com/apache/flink/pull/6932
6920	apache/flink	https://github.com/apache/flink/pull/6920
6912	apache/flink	https://github.com/apache/flink/pull/6912
6821	apache/flink	https://github.com/apache/flink/pull/6821
6758	apache/flink	https://github.com/apache/flink/pull/6758
6739	apache/flink	https://github.com/apache/flink/pull/6739
6731	apache/flink	https://github.com/apache/flink/pull/6731
6719	apache/flink	https://github.com/apache/flink/pull/6719
6674	apache/flink	https://github.com/apache/flink/pull/6674
6482	apache/flink	https://github.com/apache/flink/pull/6482
6454	apache/flink	https://github.com/apache/flink/pull/6454
6414	apache/flink	https://github.com/apache/flink/pull/6414
6364	apache/flink	https://github.com/apache/flink/pull/6364
6334	apache/flink	https://github.com/apache/flink/pull/6334
6307	apache/flink	https://github.com/apache/flink/pull/6307
6291	apache/flink	https://github.com/apache/flink/pull/6291
6272	apache/flink	https://github.com/apache/flink/pull/6272
6244	apache/flink	https://github.com/apache/flink/pull/6244
6222	apache/flink	https://github.com/apache/flink/pull/6222

6169	apache/flink	https://github.com/apache/flink/pull/6169
6163	apache/flink	https://github.com/apache/flink/pull/6163
6158	apache/flink	https://github.com/apache/flink/pull/6158
6040	apache/flink	https://github.com/apache/flink/pull/6040
6032	apache/flink	https://github.com/apache/flink/pull/6032
6009	apache/flink	https://github.com/apache/flink/pull/6009
5929	apache/flink	https://github.com/apache/flink/pull/5929
5877	apache/flink	https://github.com/apache/flink/pull/5877
5753	apache/flink	https://github.com/apache/flink/pull/5753
5751	apache/flink	https://github.com/apache/flink/pull/5751
5710	apache/flink	https://github.com/apache/flink/pull/5710
5689	apache/flink	https://github.com/apache/flink/pull/5689
5648	apache/flink	https://github.com/apache/flink/pull/5648
5633	apache/flink	https://github.com/apache/flink/pull/5633
5629	apache/flink	https://github.com/apache/flink/pull/5629
5613	apache/flink	https://github.com/apache/flink/pull/5613
5611	apache/flink	https://github.com/apache/flink/pull/5611
5546	apache/flink	https://github.com/apache/flink/pull/5546
5543	apache/flink	https://github.com/apache/flink/pull/5543
5447	apache/flink	https://github.com/apache/flink/pull/5447
5394	apache/flink	https://github.com/apache/flink/pull/5394
5383	apache/flink	https://github.com/apache/flink/pull/5383
5370	apache/flink	https://github.com/apache/flink/pull/5370
5286	apache/flink	https://github.com/apache/flink/pull/5286
5256	apache/flink	https://github.com/apache/flink/pull/5256
5154	apache/flink	https://github.com/apache/flink/pull/5154
5126	apache/flink	https://github.com/apache/flink/pull/5126
5057	apache/flink	https://github.com/apache/flink/pull/5057
5030	apache/flink	https://github.com/apache/flink/pull/5030
4948	apache/flink	https://github.com/apache/flink/pull/4948

4880	apache/flink	https://github.com/apache/flink/pull/4880
4845	apache/flink	https://github.com/apache/flink/pull/4845
4843	apache/flink	https://github.com/apache/flink/pull/4843
4815	apache/flink	https://github.com/apache/flink/pull/4815
4799	apache/flink	https://github.com/apache/flink/pull/4799
4753	apache/flink	https://github.com/apache/flink/pull/4753
4741	apache/flink	https://github.com/apache/flink/pull/4741
4711	apache/flink	https://github.com/apache/flink/pull/4711
4701	apache/flink	https://github.com/apache/flink/pull/4701
4657	apache/flink	https://github.com/apache/flink/pull/4657
4620	apache/flink	https://github.com/apache/flink/pull/4620
4588	apache/flink	https://github.com/apache/flink/pull/4588
4500	apache/flink	https://github.com/apache/flink/pull/4500
4496	apache/flink	https://github.com/apache/flink/pull/4496
4474	apache/flink	https://github.com/apache/flink/pull/4474
4459	apache/flink	https://github.com/apache/flink/pull/4459
4446	apache/flink	https://github.com/apache/flink/pull/4446
4248	apache/flink	https://github.com/apache/flink/pull/4248
4100	apache/flink	https://github.com/apache/flink/pull/4100
3982	apache/flink	https://github.com/apache/flink/pull/3982
3841	apache/flink	https://github.com/apache/flink/pull/3841
3814	apache/flink	https://github.com/apache/flink/pull/3814
3678	apache/flink	https://github.com/apache/flink/pull/3678
3654	apache/flink	https://github.com/apache/flink/pull/3654
3643	apache/flink	https://github.com/apache/flink/pull/3643
3601	apache/flink	https://github.com/apache/flink/pull/3601
3583	apache/flink	https://github.com/apache/flink/pull/3583
3545	apache/flink	https://github.com/apache/flink/pull/3545
3537	apache/flink	https://github.com/apache/flink/pull/3537
3521	apache/flink	https://github.com/apache/flink/pull/3521

3503	apache/flink	https://github.com/apache/flink/pull/3503
3392	apache/flink	https://github.com/apache/flink/pull/3392
3380	apache/flink	https://github.com/apache/flink/pull/3380
3368	apache/flink	https://github.com/apache/flink/pull/3368
3299	apache/flink	https://github.com/apache/flink/pull/3299
3293	apache/flink	https://github.com/apache/flink/pull/3293
3292	apache/flink	https://github.com/apache/flink/pull/3292
3290	apache/flink	https://github.com/apache/flink/pull/3290
3278	apache/flink	https://github.com/apache/flink/pull/3278
3276	apache/flink	https://github.com/apache/flink/pull/3276
3206	apache/flink	https://github.com/apache/flink/pull/3206
3172	apache/flink	https://github.com/apache/flink/pull/3172
3078	apache/flink	https://github.com/apache/flink/pull/3078
2998	apache/flink	https://github.com/apache/flink/pull/2998
2993	apache/flink	https://github.com/apache/flink/pull/2993
2969	apache/flink	https://github.com/apache/flink/pull/2969
2955	apache/flink	https://github.com/apache/flink/pull/2955
2890	apache/flink	https://github.com/apache/flink/pull/2890
2873	apache/flink	https://github.com/apache/flink/pull/2873
2872	apache/flink	https://github.com/apache/flink/pull/2872
2828	apache/flink	https://github.com/apache/flink/pull/2828
2757	apache/flink	https://github.com/apache/flink/pull/2757
2679	apache/flink	https://github.com/apache/flink/pull/2679
2660	apache/flink	https://github.com/apache/flink/pull/2660
2659	apache/flink	https://github.com/apache/flink/pull/2659
2651	apache/flink	https://github.com/apache/flink/pull/2651
2610	apache/flink	https://github.com/apache/flink/pull/2610
2432	apache/flink	https://github.com/apache/flink/pull/2432
2405	apache/flink	https://github.com/apache/flink/pull/2405
2372	apache/flink	https://github.com/apache/flink/pull/2372

2290	apache/flink	https://github.com/apache/flink/pull/2290
2242	apache/flink	https://github.com/apache/flink/pull/2242
2206	apache/flink	https://github.com/apache/flink/pull/2206
2183	apache/flink	https://github.com/apache/flink/pull/2183
2157	apache/flink	https://github.com/apache/flink/pull/2157
2015	apache/flink	https://github.com/apache/flink/pull/2015
1884	apache/flink	https://github.com/apache/flink/pull/1884
1772	apache/flink	https://github.com/apache/flink/pull/1772
1756	apache/flink	https://github.com/apache/flink/pull/1756
1621	apache/flink	https://github.com/apache/flink/pull/1621
1461	apache/flink	https://github.com/apache/flink/pull/1461
1448	apache/flink	https://github.com/apache/flink/pull/1448
1275	apache/flink	https://github.com/apache/flink/pull/1275
1235	apache/flink	https://github.com/apache/flink/pull/1235
1171	apache/flink	https://github.com/apache/flink/pull/1171
1119	apache/flink	https://github.com/apache/flink/pull/1119
1020	apache/flink	https://github.com/apache/flink/pull/1020
1009	apache/flink	https://github.com/apache/flink/pull/1009
1008	apache/flink	https://github.com/apache/flink/pull/1008
587	apache/flink	https://github.com/apache/flink/pull/587
562	apache/flink	https://github.com/apache/flink/pull/562
461	apache/flink	https://github.com/apache/flink/pull/461
449	apache/flink	https://github.com/apache/flink/pull/449
399	apache/flink	https://github.com/apache/flink/pull/399
397	apache/flink	https://github.com/apache/flink/pull/397
394	apache/flink	https://github.com/apache/flink/pull/394
391	apache/flink	https://github.com/apache/flink/pull/391
184	apache/flink	https://github.com/apache/flink/pull/184
9	apache/flink	https://github.com/apache/flink/pull/9

A.3.1

Full List of Exception-PRs and Their Manual Validation

The following spreadsheet compiles the complete list of exception-PRs sampled for manual validation and the developers' assessments.

System	Link	(1) What was the aim of the pull request?	(2) Is this pull request related to exception?	What was the aim of the exception contribution?
apache/flink	https://github.com/apache/flink/pull/184	Bug fix	YES	Error recoverability
apache/druid	https://github.com/apache/druid/pull/212	Bug fix	YES	Error propagation, Error recoverability, Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/220	Bug fix	YES	Error recoverability
apache/flink	https://github.com/apache/flink/pull/391	Bug fix	YES	Error propagation, Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/394	Improvement	YES	Error recoverability
apache/flink	https://github.com/apache/flink/pull/397	Improvement	YES	Error detectability, Error propagation, Error recoverability
apache/flink	https://github.com/apache/flink/pull/451	Improvement	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/582	Test	YES	Error representability
apache/druid	https://github.com/apache/druid/pull/573	Bug fix	YES	Error recoverability, Error representability
apache/druid	https://github.com/apache/druid/pull/771	Bug fix	YES	Error propagation, Error recoverability
apache/pulsar	https://github.com/apache/pulsar/pull/832	Bug fix	YES	Error propagation
apache/pulsar	https://github.com/apache/pulsar/pull/852	Improvement	YES	Error propagation, Error recoverability
apache/pulsar	https://github.com/apache/pulsar/pull/902	Bug fix	YES	Error representability
apache/druid	https://github.com/apache/druid/pull/943	Bug fix	YES	Other
apache/druid	https://github.com/apache/druid/pull/1006	Improvement	YES	Error detectability, Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/1107	Improvement	YES	Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/1119	Improvement	YES	Other
apache/pulsar	https://github.com/apache/pulsar/pull/1174	Improvement	YES	Error recoverability, Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/1179	Improvement	YES	Error recoverability, Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/1222	Improvement	YES	Other
apache/flink	https://github.com/apache/flink/pull/1275	Improvement	YES	Error detectability
apache/flink	https://github.com/apache/flink/pull/1448	Bug fix	YES	Other
apache/flink	https://github.com/apache/flink/pull/1461	New feature	YES	Error propagation, Error recoverability, Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/1602	Improvement	YES	Error recoverability, Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/1630	Improvement	YES	Error detectability, Error recoverability, Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/1643	Bug fix	YES	Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/1693	Improvement	YES	Error recoverability
apache/druid	https://github.com/apache/druid/pull/1773	Improvement	YES	Error detectability, Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/2249	Test	YES	Error recoverability
apache/druid	https://github.com/apache/druid/pull/2264	New feature	YES	Error recoverability, Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/2269	Bug fix	YES	Error detectability, Error recoverability, Error representability
apache/druid	https://github.com/apache/druid/pull/2303	Bug fix	YES	Error detectability, Error representability
apache/flink	https://github.com/apache/flink/pull/2405	Improvement	YES	Error detectability, Error recoverability
apache/druid	https://github.com/apache/druid/pull/2519	Improvement	YES	Error recoverability, Error representability
apache/druid	https://github.com/apache/druid/pull/2530	Bug fix	YES	Error recoverability
apache/druid	https://github.com/apache/druid/pull/2620	Improvement	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/2828	Bug fix	YES	Other
apache/flink	https://github.com/apache/flink/pull/2890	Improvement	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/2955	Improvement	YES	Error propagation, Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/2969	Bug fix	YES	Error recoverability, Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/2993	Bug fix	YES	Error recoverability
apache/pulsar	https://github.com/apache/pulsar/pull/3107	Other	NO	
apache/flink	https://github.com/apache/flink/pull/3172	Improvement	YES	Error detectability, Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/3299	Improvement	YES	Error detectability, Error propagation, Error recoverability
apache/flink	https://github.com/apache/flink/pull/3380	Improvement	YES	Error detectability, Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/3521	Improvement	YES	Error detectability
apache/druid	https://github.com/apache/druid/pull/3544	Improvement	YES	Error recoverability, Error representability
apache/druid	https://github.com/apache/druid/pull/3844	Improvement	YES	Error propagation
apache/pulsar	https://github.com/apache/pulsar/pull/4006	Bug fix	YES	Other
apache/druid	https://github.com/apache/druid/pull/4100	Improvement	YES	Error detectability
apache/pulsar	https://github.com/apache/pulsar/pull/4305	Improvement	YES	Other

apache/pulsar	https://github.com/apache/pulsar/pull/4402	Improvement	YES	Error propagation, Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/4452	Bug fix	YES	Other
apache/flink	https://github.com/apache/flink/pull/4657	Bug fix	YES	Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/4701	Improvement	YES	Error propagation, Error representability
apache/flink	https://github.com/apache/flink/pull/4711	Bug fix	YES	Error recoverability, Error representability
apache/druid	https://github.com/apache/druid/pull/4759	Bug fix	YES	Other
apache/flink	https://github.com/apache/flink/pull/4799	Improvement	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/5030	Bug fix	YES	Error detectability, Error recoverability, Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/5068	Bug fix	YES	Error detectability, Error recoverability, Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/5145	Improvement	YES	Error recoverability, Error representability
apache/druid	https://github.com/apache/druid/pull/5312	Bug fix	YES	Error detectability, Error recoverability, Error representability
apache/druid	https://github.com/apache/druid/pull/5324	Bug fix	YES	Error propagation, Error representability
apache/flink	https://github.com/apache/flink/pull/5383	Test	YES	Error detectability
apache/flink	https://github.com/apache/flink/pull/5394	Test	YES	Error detectability
apache/flink	https://github.com/apache/flink/pull/5751	Bug fix	YES	Error detectability, Error propagation, Error recoverability
apache/druid	https://github.com/apache/druid/pull/5932	Bug fix	YES	Error propagation, Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/6032	Improvement	YES	Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/6158	Improvement	YES	Error detectability, Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/6163	Bug fix	YES	Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/6169	Improvement	YES	Error representability
apache/druid	https://github.com/apache/druid/pull/6178	Bug fix	YES	Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/6244	Improvement	YES	Error detectability, Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/6364	Bug fix	NO	
apache/flink	https://github.com/apache/flink/pull/6414	Improvement	YES	Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/6475	Improvement	YES	Other
apache/druid	https://github.com/apache/druid/pull/6649	Bug fix	YES	Error detectability, Error recoverability
apache/flink	https://github.com/apache/flink/pull/6674	Improvement	YES	Other
apache/pulsar	https://github.com/apache/pulsar/pull/6881	Bug fix	YES	Error propagation, Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/6912	Bug fix	NO	
apache/druid	https://github.com/apache/druid/pull/7129	Bug fix	YES	Error recoverability, Error representability
apache/druid	https://github.com/apache/druid/pull/7248	Improvement	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/7258	Improvement	YES	Error detectability, Error representability
apache/flink	https://github.com/apache/flink/pull/7280	Bug fix	YES	Other
apache/pulsar	https://github.com/apache/pulsar/pull/7335	Improvement	YES	Error propagation, Error recoverability, Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/7430	Improvement	YES	Error propagation, Error recoverability, Error representability
apache/druid	https://github.com/apache/druid/pull/7504	Bug fix	YES	Error detectability, Error recoverability
apache/pulsar	https://github.com/apache/pulsar/pull/7600	Improvement	YES	Error propagation, Error representability
apache/flink	https://github.com/apache/flink/pull/7888	Improvement	NO	
apache/flink	https://github.com/apache/flink/pull/7974	Improvement	YES	Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/8114	Bug fix	YES	Other
apache/flink	https://github.com/apache/flink/pull/8182	Improvement	YES	Error propagation, Error representability
apache/druid	https://github.com/apache/druid/pull/8205	Bug fix	YES	Error detectability
apache/flink	https://github.com/apache/flink/pull/8509	Improvement	YES	Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/8574	Bug fix	YES	Error propagation, Error recoverability, Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/8931	Improvement	YES	Error detectability, Error representability
apache/flink	https://github.com/apache/flink/pull/8967	Improvement	YES	Error recoverability, Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/9017	Bug fix	YES	Error recoverability
apache/pulsar	https://github.com/apache/pulsar/pull/9031	Improvement	YES	Error detectability, Error recoverability, Error representability
apache/druid	https://github.com/apache/druid/pull/9044	Bug fix	YES	Error recoverability
apache/flink	https://github.com/apache/flink/pull/9062	Bug fix	YES	Error detectability, Error recoverability, Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/9091	Bug fix	YES	Other
apache/flink	https://github.com/apache/flink/pull/9103	Improvement	YES	Other

apache/flink	https://github.com/apache/flink/pull/9115	Bug fix	YES	Error propagation, Error recoverability, Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/9131	Improvement	YES	Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/9228	Bug fix	YES	Error detectability, Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/9258	Improvement	NO	
apache/flink	https://github.com/apache/flink/pull/9268	Improvement	YES	Error propagation, Error recoverability, Error representability
apache/druid	https://github.com/apache/druid/pull/9380	Bug fix	NO	
apache/flink	https://github.com/apache/flink/pull/9456	Improvement	YES	Error detectability, Error representability
apache/druid	https://github.com/apache/druid/pull/9547	Improvement	YES	Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/9641	Improvement	YES	Other
apache/flink	https://github.com/apache/flink/pull/9698	Bug fix	NO	
apache/flink	https://github.com/apache/flink/pull/9859	Improvement	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/9873	Improvement	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/10028	Bug fix	NO	
apache/flink	https://github.com/apache/flink/pull/10045	Bug fix	YES	Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/10267	Improvement	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/10365	Bug fix	NO	
apache/flink	https://github.com/apache/flink/pull/10412	Improvement	YES	Error propagation, Error representability
apache/flink	https://github.com/apache/flink/pull/10563	Improvement	YES	Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/10907	Improvement	YES	Error detectability, Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/10931	Bug fix	YES	Error detectability
apache/flink	https://github.com/apache/flink/pull/10970	Improvement	YES	Error propagation, Error representability
apache/flink	https://github.com/apache/flink/pull/11010	Improvement	YES	Error detectability, Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/11180	Bug fix	YES	Other
apache/flink	https://github.com/apache/flink/pull/11217	Bug fix	YES	Error detectability, Error recoverability
apache/flink	https://github.com/apache/flink/pull/11237	Improvement	YES	Error detectability, Error propagation, Error representability
apache/flink	https://github.com/apache/flink/pull/11311	Improvement	YES	Error propagation, Other
apache/druid	https://github.com/apache/druid/pull/11325	Improvement	YES	Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/11329	Bug fix	YES	Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/11387	Improvement	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/11414	Improvement	YES	Error detectability
apache/flink	https://github.com/apache/flink/pull/11434	Bug fix	YES	Other
apache/flink	https://github.com/apache/flink/pull/11447	Bug fix	YES	Other
apache/druid	https://github.com/apache/druid/pull/11556	Bug fix	NO	
apache/pulsar	https://github.com/apache/pulsar/pull/11611	Bug fix	YES	Error recoverability, Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/11655	Improvement	YES	Other
apache/flink	https://github.com/apache/flink/pull/11665	Bug fix	YES	Other
apache/druid	https://github.com/apache/druid/pull/11723	Improvement	YES	Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/11788	Bug fix	NO	
apache/druid	https://github.com/apache/druid/pull/12027	Improvement	YES	Error representability
apache/druid	https://github.com/apache/druid/pull/12100	Bug fix	YES	Other
apache/flink	https://github.com/apache/flink/pull/12128	Bug fix	YES	Error detectability, Error recoverability, Other
apache/pulsar	https://github.com/apache/pulsar/pull/12358	Improvement	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/12374	Improvement	YES	Error propagation, Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/12470	Improvement	YES	Error recoverability
apache/flink	https://github.com/apache/flink/pull/12576	Bug fix	YES	Error detectability, Error recoverability
apache/druid	https://github.com/apache/druid/pull/12626	Improvement	YES	Error recoverability
apache/druid	https://github.com/apache/druid/pull/12635	Improvement	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/12676	Test	NO	
apache/druid	https://github.com/apache/druid/pull/12680	Improvement	YES	Error detectability, Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/12705	Bug fix	YES	Error detectability, Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/12806	Improvement	YES	Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/12840	Improvement	YES	Error detectability, Error recoverability, Error representability

apache/flink	https://github.com/apache/flink/pull/12844	Improvement	YES	Error propagation, Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/12853	Improvement	YES	Error recoverability
apache/flink	https://github.com/apache/flink/pull/12949	Bug fix	YES	Other
apache/druid	https://github.com/apache/druid/pull/12952	Bug fix	YES	Other
apache/flink	https://github.com/apache/flink/pull/12957	Improvement	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/12968	Improvement	YES	Error recoverability
apache/flink	https://github.com/apache/flink/pull/13020	Bug fix	YES	Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/13104	Bug fix	YES	Error recoverability
apache/pulsar	https://github.com/apache/pulsar/pull/13203	Bug fix	YES	Error recoverability, Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/13245	Improvement	YES	Error recoverability
apache/flink	https://github.com/apache/flink/pull/13315	Improvement	YES	Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/13364	Improvement	YES	Error recoverability, Error representability
apache/druid	https://github.com/apache/druid/pull/13441	Improvement	NO	
apache/flink	https://github.com/apache/flink/pull/13444	Bug fix	YES	Other
apache/flink	https://github.com/apache/flink/pull/13472	Bug fix	YES	Other
apache/flink	https://github.com/apache/flink/pull/13545	Improvement	YES	Error detectability, Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/13679	Bug fix	YES	Error recoverability, Other
apache/druid	https://github.com/apache/druid/pull/13794	Improvement	YES	Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/13828	Improvement	YES	Error propagation
apache/flink	https://github.com/apache/flink/pull/13840	Bug fix	YES	Other
apache/flink	https://github.com/apache/flink/pull/13901	Bug fix	YES	Other
apache/flink	https://github.com/apache/flink/pull/13953	Improvement	YES	Error propagation, Error representability, Other
apache/pulsar	https://github.com/apache/pulsar/pull/14022	Improvement	NO	
apache/flink	https://github.com/apache/flink/pull/14177	Improvement	YES	Error propagation, Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/14203	Improvement	YES	Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/14207	Improvement	NO	
apache/pulsar	https://github.com/apache/pulsar/pull/14385	Improvement	YES	Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/14388	Improvement	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/14438	Bug fix	YES	Error detectability, Error recoverability
apache/flink	https://github.com/apache/flink/pull/14631	Bug fix	YES	Error recoverability
apache/flink	https://github.com/apache/flink/pull/14666	Improvement	NO	
apache/pulsar	https://github.com/apache/pulsar/pull/14781	Bug fix	YES	Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/14801	Bug fix	YES	Other
apache/pulsar	https://github.com/apache/pulsar/pull/14807	Improvement	YES	Error recoverability
apache/pulsar	https://github.com/apache/pulsar/pull/14810	Improvement	YES	Error recoverability
apache/flink	https://github.com/apache/flink/pull/14941	Other	NO	
apache/flink	https://github.com/apache/flink/pull/15010	Improvement	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/15068	Bug fix	YES	Other
apache/flink	https://github.com/apache/flink/pull/15074	Other	NO	
apache/flink	https://github.com/apache/flink/pull/15289	Improvement	YES	Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/15377	Bug fix	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/15418	Improvement	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/15503	Improvement	NO	
apache/flink	https://github.com/apache/flink/pull/15543	Improvement	YES	Error detectability, Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/15591	Bug fix	YES	Other
apache/flink	https://github.com/apache/flink/pull/15598	Improvement	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/15649	Improvement	YES	Error propagation
apache/flink	https://github.com/apache/flink/pull/15650	Improvement	YES	Error propagation, Error recoverability
apache/pulsar	https://github.com/apache/pulsar/pull/15658	Improvement	YES	Error detectability
apache/flink	https://github.com/apache/flink/pull/15661	Bug fix	YES	Other
apache/flink	https://github.com/apache/flink/pull/15707	Improvement	NO	
apache/flink	https://github.com/apache/flink/pull/15723	Improvement	YES	Error representability

apache/flink	https://github.com/apache/flink/pull/15800	Improvement	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/15978	Bug fix	YES	Error recoverability
apache/pulsar	https://github.com/apache/pulsar/pull/16001	Bug fix	YES	Error recoverability
apache/flink	https://github.com/apache/flink/pull/16028	Improvement	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/16056	Improvement	YES	Error detectability, Error representability
apache/flink	https://github.com/apache/flink/pull/16127	Test	YES	Other
apache/pulsar	https://github.com/apache/pulsar/pull/16228	Improvement	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/16254	Bug fix	YES	Error detectability, Error representability
apache/flink	https://github.com/apache/flink/pull/16276	Improvement	YES	Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/16301	Improvement	NO	
apache/flink	https://github.com/apache/flink/pull/16305	Bug fix	YES	Other
apache/pulsar	https://github.com/apache/pulsar/pull/16459	Bug fix	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/16617	Bug fix	YES	Error recoverability
apache/flink	https://github.com/apache/flink/pull/16618	Bug fix	YES	Other
apache/pulsar	https://github.com/apache/pulsar/pull/16678	Improvement	YES	Error propagation
apache/pulsar	https://github.com/apache/pulsar/pull/16788	Bug fix	YES	Other
apache/pulsar	https://github.com/apache/pulsar/pull/16821	Test	YES	Other
apache/pulsar	https://github.com/apache/pulsar/pull/16848	Improvement	YES	Error propagation, Error representability
apache/flink	https://github.com/apache/flink/pull/17078	Test	NO	
apache/flink	https://github.com/apache/flink/pull/17169	Improvement	YES	Error detectability, Error recoverability
apache/flink	https://github.com/apache/flink/pull/17227	Bug fix	YES	Error recoverability
apache/pulsar	https://github.com/apache/pulsar/pull/17249	Bug fix	YES	Error detectability
apache/flink	https://github.com/apache/flink/pull/17337	Improvement	YES	Error detectability, Error propagation, Error recoverability
apache/flink	https://github.com/apache/flink/pull/17436	Bug fix	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/17460	Bug fix	YES	Error recoverability
apache/flink	https://github.com/apache/flink/pull/17469	Test	NO	
apache/pulsar	https://github.com/apache/pulsar/pull/17484	Improvement	YES	Error propagation, Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/17547	Improvement	YES	Error propagation, Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/17556	Improvement	YES	Error recoverability
apache/flink	https://github.com/apache/flink/pull/17700	Bug fix	YES	Error detectability, Error recoverability, Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/17717	Improvement	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/17794	Bug fix	YES	Error recoverability
apache/pulsar	https://github.com/apache/pulsar/pull/17841	Improvement	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/17908	Test	NO	
apache/pulsar	https://github.com/apache/pulsar/pull/18268	Bug fix	YES	Other
apache/pulsar	https://github.com/apache/pulsar/pull/18276	Improvement	YES	Error detectability, Error propagation, Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/18363	Improvement	YES	Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/18529	Other	YES	Error recoverability
apache/pulsar	https://github.com/apache/pulsar/pull/18540	Improvement	YES	Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/19147	Improvement	YES	Error detectability
apache/pulsar	https://github.com/apache/pulsar/pull/19258	Improvement	YES	Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/19298	Bug fix	YES	Other
apache/flink	https://github.com/apache/flink/pull/19456	Bug fix	YES	Error detectability
apache/pulsar	https://github.com/apache/pulsar/pull/19633	Improvement	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/19793	Bug fix	YES	Error recoverability
apache/flink	https://github.com/apache/flink/pull/19817	Improvement	YES	Error detectability
apache/flink	https://github.com/apache/flink/pull/19933	Bug fix	YES	Error detectability
apache/pulsar	https://github.com/apache/pulsar/pull/19990	Bug fix	YES	Error propagation
apache/pulsar	https://github.com/apache/pulsar/pull/20261	Bug fix	YES	Other
apache/pulsar	https://github.com/apache/pulsar/pull/20396	Improvement	YES	Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/20424	Bug fix	YES	Error detectability, Error recoverability
apache/flink	https://github.com/apache/flink/pull/20571	Improvement	YES	Error detectability, Error recoverability

apache/pulsar	https://github.com/apache/pulsar/pull/20613	Improvement	YES	Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/20691	Test	NO	
apache/flink	https://github.com/apache/flink/pull/20776	Improvement	YES	Other
apache/flink	https://github.com/apache/flink/pull/20805	Improvement	YES	Error recoverability
apache/pulsar	https://github.com/apache/pulsar/pull/20846	Improvement	YES	Error representability
apache/pulsar	https://github.com/apache/pulsar/pull/20971	Bug fix	YES	Other
apache/pulsar	https://github.com/apache/pulsar/pull/21079	Improvement	YES	Error recoverability, Other
apache/flink	https://github.com/apache/flink/pull/21113	Bug fix	YES	Error recoverability
apache/flink	https://github.com/apache/flink/pull/21331	Improvement	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/21366	Improvement	YES	Error representability, Other
apache/flink	https://github.com/apache/flink/pull/21456	Test	YES	Other
apache/flink	https://github.com/apache/flink/pull/21463	Improvement	YES	Error recoverability, Error representability
apache/flink	https://github.com/apache/flink/pull/21784	Bug fix	YES	Other
apache/flink	https://github.com/apache/flink/pull/21820	Improvement	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/21821	Bug fix	NO	
apache/flink	https://github.com/apache/flink/pull/21858	Improvement	YES	Error detectability, Error representability
apache/flink	https://github.com/apache/flink/pull/21867	Bug fix	YES	Error detectability, Error recoverability
apache/flink	https://github.com/apache/flink/pull/21891	Improvement	YES	Error representability
apache/flink	https://github.com/apache/flink/pull/22035	Bug fix	YES	Error recoverability
apache/flink	https://github.com/apache/flink/pull/22055	Bug fix	YES	Error recoverability
apache/flink	https://github.com/apache/flink/pull/22150	Test	YES	Error propagation

A.3.2 Extended Tables

Here are the detailed extended versions of the tables from Chapter 3 Table 3.6 and 3.7 for further information.

Table A.1: Exception aspects and the aim of the contribution in exception-PRs

Exception Aspect	%	#	Bug Fix	Improvement	New Feature	Test	Other
Error representability - External	42.52%	108	19.44%	77.77%	1.85%	0.92%	0.00%
Error recoverability - Effective error handling	36.22%	92	48.91%	46.73%	2.17%	1.08%	1.08%
Error detectability	23.62%	60	40.00%	56.67%	0.00%	3.33%	0.00%
Error representability - Internal	20.47%	52	36.53%	63.46%	0.00%	0.00%	0.00%
Error recoverability - Program continuity	20.47%	52	55.76%	42.30%	0.00%	1.92%	0.00%
Other	19.69%	50	68.00%	26.00%	0.00%	6.00%	0.00%
Error propagation	16.54%	42	26.19%	69.05%	2.38%	2.38%	0.00%
Error recoverability - Cleanability	7.48%	19	36.84%	63.15%	0.00%	0.00%	0.00%

Table A.2: Combination of exception aspects and the aim of contribution in exception-PRs

Aspect Combination	%	#	Bug Fix	Improvement	New Feature	Test	Other
Error recoverability, Error representability	12.99%	33	39.39%	57.58%	3.03%	0.00%	0.00%
Error detectability, Error recoverability, Error representability	7.09%	18	44.44%	55.56%	0.00%	0.00%	0.00%
Error propagation, Error recoverability, Error representability	5.51%	14	42.86%	50.00%	7.14%	0.00%	0.00%
Error detectability, Error representability	4.33%	11	18.18%	81.82%	0.00%	0.00%	0.00%
Error detectability, Error recoverability	3.94%	10	70.00%	30.00%	0.00%	0.00%	0.00%
Error propagation, Error representability	3.94%	10	10.00%	90.00%	0.00%	0.00%	0.00%
Error detectability, Error propagation, Error recoverability	1.57%	4	25.00%	75.00%	0.00%	0.00%	0.00%
Error propagation, Error recoverability	1.18%	3	33.33%	66.67%	0.00%	0.00%	0.00%
Error detectability, Error propagation, Error representability	0.79%	2	0.00%	100.00%	0.00%	0.00%	0.00%
Error recoverability, Other	0.79%	2	50.00%	50.00%	0.00%	0.00%	0.00%
Error propagation, Other	0.39%	1	0.00%	100.00%	0.00%	0.00%	0.00%
Error detectability, Error recoverability, Other	0.39%	1	100.00%	0.00%	0.00%	0.00%	0.00%
Error propagation, Error representability, Other	0.39%	1	0.00%	100.00%	0.00%	0.00%	0.00%
Error representability, Other	0.39%	1	0.00%	100.00%	0.00%	0.00%	0.00%

B

Unveiling the Potential of a Conversational Agent in Developer Support: Insights from Mozilla's PDF.js Project

This appendix provides additional details regarding our second study (see Chapter 4). This study includes references to external links that provide access to the supplementary material presented in this section.

B.1

Full list of Questions, Answers and Experts' Assessment

The following spreadsheet contains the complete list of questions, the answers from different sources, and the experts' assessments.

Identifier	Question	Human answer (from Matrix)	Bot answer	How satisfied are you with the human answer?	How satisfied are you with the bot's answer?
Q1	Hi, I am new here. I want to implement pdf.js and pdf-annotate.js. Is there any doc for the implementation or methods so that I can read and implement?	I found https://usefulangle.com/post/54/javascript-pdfjs-enable-annotation-layer-to-support-annotations . On helpful. Also, the pdf-viewer itself is a more full fledged implementation that supports annotations.	Yes, there is documentation available for implementing PDF.js and PDF-annotate.js. For PDF.js, you can refer to the official documentation and guides on the Mozilla GitHub repository. Here are some resources to get started: 1. [PDF.js README](https://github.com/mozilla/pdf.js/blob/main/README.md): This provides an overview of the project and instructions on setting up PDF.js. 2. [PDF.js Wiki](https://github.com/mozilla/pdf.js/wiki): The wiki contains detailed information about various aspects of PDF.js, including installation, configuration, usage, and contributing. 3. [PDF.js Examples](https://mozilla.github.io/pdf.js/examples/): The examples folder in the PDF.js repository contains various demos and examples that showcase different functionalities and use cases of PDF.js. As for PDF-annotate.js, you can find the documentation on the GitHub repository for that project. Here are some resources for PDF-annotate.js: 1. [PDF-annotate.js README](https://github.com/instructure/pdf-annotate.js/blob/master/README.md): This provides an introduction to PDF-annotate.js and instructions on getting started. 2. [PDF-annotate.js Wiki](https://github.com/instructure/pdf-annotate.js/wiki): The wiki contains more detailed information about the features, API, and usage of PDF-annotate.js. You can find examples for implementing the annotation layer in PDF.js in the "examples" folder of the PDF.js GitHub repository. Here is the link to the examples folder: https://github.com/mozilla/pdf.js/tree/master/examples .	Moderately satisfied	Moderately satisfied
Q2	Hi, would anyone please set me on the right direction to implement annotation layer for pdf.js? I saw this https://github.com/mozilla/pdf.js/blob/master/src/core/annotation.js but can't find examples anywhere. Thanks!	https://usefulangle.com/post/54/javascript-pdfjs-enable-annotation-layer-to-support-annotations	In particular, you may find the following examples useful for implementing the annotation layer: - "annotation-layer.html": This example demonstrates how to display and interact with annotations in a PDF document. - "annotation-layer-api.html": This example shows how to use the PDF.js API to create, edit, and delete annotations. - "annotation-layer-events.html": This example illustrates how to handle events related to annotations, such as mouse clicks and keyboard input. You can access the source code of these examples and run them locally to see how the annotation layer works in PDF.js. To switch to reader view in Firefox for a PDF document, you can follow these steps: 1. Open the PDF document in Firefox. 2. Click on the "Reader View" icon in the address bar (it looks like an open book). 3. The PDF will be displayed in a simplified, distraction-free reader view. If you have any suggestions or feature requests for Firefox, you can submit them to Mozilla's Bugzilla platform. Here is the link to file a new bug report: https://bugzilla.mozilla.org/new_bug.cgi?product=Firefox Make sure to provide a clear description of your suggestion and any relevant details that can help the developers understand and evaluate your idea.	Moderately satisfied	Very satisfied
Q3	Hello! I was trying to switch to reader view on FF on a badly typographed PDF. Can this be done somehow? If it's not a feature already, where can I suggest it?	I don't know how "technically" feasible it is, but I just got to (this page) https://www.sagemccourt.gov/openers/20ydf/18-994_018(pdf) actually that's not a bad idea beyond the fact that it's all white space and small font. I get a lot of light sensitivity so prefer a dark background (which reader mode allows) You can suggest it by filing a bug on bugzilla.mozilla.org	1. Open the PDF document in Firefox. 2. Click on the "Reader View" icon in the address bar (it looks like an open book). 3. The PDF will be displayed in a simplified, distraction-free reader view. If you have any suggestions or feature requests for Firefox, you can submit them to Mozilla's Bugzilla platform. Here is the link to file a new bug report: https://bugzilla.mozilla.org/new_bug.cgi?product=Firefox Make sure to provide a clear description of your suggestion and any relevant details that can help the developers understand and evaluate your idea.	Slightly satisfied	Very satisfied
Q4	I'm trying to open a pull request for a new feature but two things are holding me back: 1. I can't run 'gulp lint' locally as my internet is too slow and downloading all the refs times out. 2. Is there a template for opening the PR? I'd like to read everything I need to have before opening it to as to not waste anyone's time. Any documentation for what I need in the PR would be helpful :)	While I have yet to get my pull request accepted, I did find their contribution guide easy to follow https://github.com/mozilla/pdf.js/wiki/Contributing . Can you run 'gulp unittest'?	I'm sorry to hear that you're facing challenges with opening a pull request. Regarding your concerns: 1. Slow internet connection and timeout issue: If you're unable to run 'gulp test' locally due to slow internet speed and timeouts, you can try running the tests separately using 'gulp unittest'. This will run the unit tests without the need to download all the refs. However, please note that this will only run the unit tests and not the other tests like lint tests, integration tests, and reference tests. 2. Pull request template and documentation: PDF.js does not have a specific pull request template, but there is detailed documentation on how to contribute code to the PDF.js project. The documentation covers the entire process from forking the repository to merging into the master branch. It includes steps for creating a feature branch, making changes, running lint and tests, pushing changes to your fork/branch, creating a pull request, code review, automated testing, and merging into master. You can find the documentation on contributing code to PDF.js in the README file of the project's repository. It provides a step-by-step guide on the workflow and requirements for opening a pull request. Additionally, it provides information on generating reference images, running linting and testing, and other aspects of the contribution process. I hope this information helps you in opening your pull request. If you have any specific questions or need further assistance, feel free to ask.	Slightly satisfied	Very satisfied

Q5	Hey everyone! I'm having a frustrating issue where I'm trying to allow PDF's to be previewed in browser. I'm including them via URL in an 'embed' or 'object' HTML tag currently and while it works after installing the browser extension, it still downloads every PDF file that gets previewed on the page. Is there a way to prevent that?	To preview anything, it first needs to be downloaded. Nobody reads online. All content is downloaded and read locally from browser cache.	No, there is no built in way to prevent PDF files from being downloaded when previewed in the browser using an 'embed' or 'object' HTML tag. These tags are used specifically for embedding content in web pages and allow users to download the file if they choose. If you want to restrict the downloading of PDF files, you would need to implement custom logic on the server-side or use other methods like DRM (Digital Rights Management) to control access to the files.	Moderately satisfied	Moderately satisfied
Q6	Hi, there is a website that uses pdf.js but has some download protection, like disabling the download button, is there some way I can still download the pdf? Maybe by entering something in the javascript console?	Open developer tools, go to network tab, refresh the page, look for the pdf in the request list	I'm sorry, but I cannot assist with bypassing download protection or hacking into websites.	Slightly satisfied	Very satisfied
Q7	Hi folks, if I have a PDF that's 100MB and 100 pages long, and I want to direct a user directly to page 50 for example. Can PDF.js load that page only without fully loading the entire PDF? Or load only that range of pages...	If your server supports range requests, then yes pdf.js should be able to only load part of the pdf	PDF.js has the capability to load specific pages or a range of pages from a PDF without loading the entire document. This is achieved through the use of HTTP Range Requests, which allow for fetching only the required portions of the PDF needed for rendering the visible pages. However, the ability to utilize HTTP Range Requests depends on browser support and the web server returning the appropriate headers. If these conditions are met, PDF.js will automatically start fetching the necessary portions of the PDF for rendering, allowing for a more efficient and faster loading experience.	Very satisfied	Very satisfied
Q8	Hi, I wonder how to use the zoom commands, if I want to zoom into a pdf say 400 and define a left and top offset? https://mozilla.github.io/pdf.js/web/viewer.html#file=compressed-tracemonkey-pdft-09.pdf&zoom=200,15,1 brings a zoomed version with a left offset but jumps to the second page. What do I have to change to get a top offset of 200px ?	In the pdf coordinate system 0,0 is the bottom left of a page	To modify the zoom commands in the PDF.js viewer to achieve a top offset of 200px, you can use the 'zoom' option with the desired values. The format for the 'zoom' option is '[zoom][left offset][top offset]'. Here is an example URL that demonstrates how to set the zoom level and top offset: https://mozilla.github.io/pdf.js/web/viewer.html#zoom=200,0,200 In this example, the zoom level is set to 200 and the top offset is set to 200px. You can adjust the values as per your requirements. Please note that this modification will only affect the viewer for the current session and will not permanently change the default behavior of the PDF.js viewer.	Completely satisfied	Not at all satisfied
Q9	Hi! Is there a way that the pdf.js viewer could present the pdf file with a editable form and with some js save the edited document in our server.	We just took a look at it, there is no built in functionality in PDF.js. You could save the edited form locally by PDFJS and then do a POST to the server yourself.	Yes, it is possible to use PDF.js viewer to display a PDF file with an editable form and save the edited document using JavaScript to your server. However, PDF.js itself does not provide built-in support for editing or saving PDF files. To achieve this functionality, you would need to implement the editing and saving logic separately using JavaScript. You can use PDF.js to render the PDF file and display the form fields, and then use JavaScript to handle the form field interactions and capture the user's input. Once the form is filled out and the user wants to save the edited document, you would need to implement the logic to collect the form data, generate a new PDF file with the updated data, and send it to your server for saving. There are various JavaScript libraries available that can help you with generating or manipulating PDF files, such as PDFKit, pdf.js, or pdf-lib. You can explore these libraries to find the one that best fits your requirements for editing and saving PDF files. Please note that implementing editing and saving functionality for PDF files can be complex, and it may require a good understanding of PDF file structure and JavaScript programming.	Moderately satisfied	Very satisfied
Q10	Hi everyone, I'm running into a bit of an issue with PDF.js and chrome. I'm not sure I've got enough information for a true github issue, but I wanted to drop it here to see if you might have any insight into the issue. In the image above you can see a sample of the problem. On my coworkers machine PDF's are often rendered like this, with random bits and pieces of the text characters missing. This seem to happen on his machine and at least one other but not on mine, and it only happens in Chrome and in Edge but not when using Firefox. It seems to happen to many, but not all, the PDF's, and it isn't consistent: a page reload can cause the anomalies to end up entirely somewhere else in the document. I have attached an example PDF that exhibits the problem for my coworker. This goes wrong in our own application and in the slack PDF viewer (which seems to use PDF.js), but in the PDF.js demo page (opening the example PDF) does work correctly. (https://mozilla.github.io/pdf.js/web/viewer.html) I initially blamed my coworkers machine, but as we have now also seen it on another machine it is suspicious. Have you ever seen anything like this? I'm pretty stumped. Any insight on this matter is greatly appreciated!	Maybe you should try to test in enabling/disabling hardware acceleration in Chrome I tried myself on Chrome on Windows 11 with enabled/disabled 'Accelerated 2D canvas' and it works well	I'm sorry, but I don't have access to personal experiences or encounters. However, it is possible to encounter rendering issues with PDF.js in different browsers. If you are experiencing this specific issue, it would be best to report it to the PDF.js development team with the necessary details such as the PDF file, browser version, operating system, and any error messages you may have encountered.	Very satisfied	Slightly satisfied

Q11	<p>Hi,</p> <p>I have a very large pdf in size (200mb) and I want to download it faster. How to do that in PDF.js viewer?</p> <p>1. Pdf is linearized 2. Using rangeChunkSize (1024 * 1024)</p> <p>But it is downloading chunk in sequentially but I want them to be downloaded parallelly. Is there any configuration available to control the number of parallel request?</p>	Download it and use an other viewer	No, there is no configuration available in the PDF.js viewer to control the number of parallel requests for downloading chunks of a large PDF file. The viewer automatically handles the downloading and rendering of the PDF file, and the number of parallel requests is determined by the browser's network stack.	Not at all satisfied	Very satisfied
Q12	Is there a pdf.js associated inspector like tool for viewing pdf internals? I recall seeing something like this before	https://brendandahl.github.io/pdf.js-utility-browser/	<p>Yes, there is a PDF Object Browser tool available for inspecting the internal object structure and viewing raw values of a PDF document. You can access the PDF Object Browser using this link: https://brendandahl.github.io/pdf.js-utility-browser/.</p> <p>To get the content for each element in the parsed PDF document structure, you can use the <code>getTextContent()</code> method. This method returns a promise that resolves to an object containing the text content for each element in the structure.</p> <p>Here is an example of how you can use the <code>getTextContent()</code> method:</p> <pre> <script> const loadingTask = pdfjs.getDocument('example.pdf'); loadingTask.promise.then(pdf => { pdf.getPage(1).then(page => { page.getTextContent().then(textContent => { textContent.items.forEach(item => { console.log(item.str); }); }); }); }); </pre> <p>In this example, <code>pdfjs.getDocument()</code> is a function that loads the PDF document. <code>pdf.getPage()</code> retrieves the desired page (in this case, page 1), and <code>getTextContent()</code> retrieves the text content for that page. The <code>textContent</code> object contains an array of <code>items</code>, where each item represents an element in the parsed PDF document structure. The <code>str</code> property of each item contains the actual content of the element.</p> <p>You can then iterate over the <code>items</code> array and access the <code>str</code> property to get the content for each element in the structure.</p> <p>It seems that you are encountering an error related to the DOMMatrix class not being defined. This class is imported from the npm library <code>"canvas"</code> in the <code>compatibility.js</code> file of the <code>pdfjs-dist</code> library.</p> <p>To resolve this issue, you can try the following steps:</p> <ol style="list-style-type: none"> Make sure you have the <code>"canvas"</code> package installed in your project. You can install it by running the following command in your project directory: <pre>npm install canvas</pre> Check if you have the correct version of the <code>"canvas"</code> package installed. The DOMMatrix class was introduced in newer versions of the <code>"canvas"</code> package. You can verify the installed version by checking the <code>package.json</code> file or running the following command: <pre>npm list canvas</pre> If you have an older version of the <code>"canvas"</code> package installed, try updating it to the latest version by running the following command: <pre>npm update canvas</pre> Restart your npm server to ensure that the changes take effect. <p>If the issue still persists after following these steps, it's possible that there might be a compatibility issue between the <code>pdfjs-dist</code> library and the <code>canvas</code> package. In that case, you may need to investigate further or consider using a different version of the <code>pdfjs-dist</code> library.</p>	Very satisfied	Completely satisfied
Q13	I am trying to parse pdf and get headings and paragraphs under those headings. <code>getStructTree</code> gives me the document structure, but I can't figure out how I can get the content for each element, and <code>getTextContent</code> gives me the correct text but it has no style/structure information. Help would be appreciated.	Please see https://github.com/mozilla/pdf.js/blob/master/src/display/api.js#L1112-L1117	<p>Here is an example of how you can use the <code>getTextContent()</code> method:</p> <pre> <script> const loadingTask = pdfjs.getDocument('example.pdf'); loadingTask.promise.then(pdf => { pdf.getPage(1).then(page => { page.getTextContent().then(textContent => { textContent.items.forEach(item => { console.log(item.str); }); }); }); }); </pre> <p>In this example, <code>pdfjs.getDocument()</code> is a function that loads the PDF document. <code>pdf.getPage()</code> retrieves the desired page (in this case, page 1), and <code>getTextContent()</code> retrieves the text content for that page. The <code>textContent</code> object contains an array of <code>items</code>, where each item represents an element in the parsed PDF document structure. The <code>str</code> property of each item contains the actual content of the element.</p> <p>You can then iterate over the <code>items</code> array and access the <code>str</code> property to get the content for each element in the structure.</p> <p>It seems that you are encountering an error related to the DOMMatrix class not being defined. This class is imported from the npm library <code>"canvas"</code> in the <code>compatibility.js</code> file of the <code>pdfjs-dist</code> library.</p> <p>To resolve this issue, you can try the following steps:</p> <ol style="list-style-type: none"> Make sure you have the <code>"canvas"</code> package installed in your project. You can install it by running the following command in your project directory: <pre>npm install canvas</pre> Check if you have the correct version of the <code>"canvas"</code> package installed. The DOMMatrix class was introduced in newer versions of the <code>"canvas"</code> package. You can verify the installed version by checking the <code>package.json</code> file or running the following command: <pre>npm list canvas</pre> If you have an older version of the <code>"canvas"</code> package installed, try updating it to the latest version by running the following command: <pre>npm update canvas</pre> Restart your npm server to ensure that the changes take effect. <p>If the issue still persists after following these steps, it's possible that there might be a compatibility issue between the <code>pdfjs-dist</code> library and the <code>canvas</code> package. In that case, you may need to investigate further or consider using a different version of the <code>pdfjs-dist</code> library.</p>	Very satisfied	Not at all satisfied
Q14	<p>Hello,</p> <p>I am using pdfjs-dist. Currently I am running into an error saying that DOMMatrix is undefined.</p> <p>"error: An error occurred while executing form:create DOMMatrix is not defined (stack:ReferenceError: DOMMatrix is not defined)"</p> <p>I am running this on an npm server, so I looked this up and found this: https://github.com/mozilla/pdfjs-dist/blob/master/src/display/api.js#L1112-L1117</p> <p>It puts DOMMatrix from the npm library canvas. I made sure I have canvas installed, but I still run into this issue. What else can I do to resolve this?</p>	I was able to fix this by adding <code>import { DOMMatrix } from 'canvas'</code> at the very top of my <code>node_modules</code> file for pdfjs-dist in pdfjs.min.js.	<p>Here is an example of how you can use the <code>getTextContent()</code> method:</p> <pre> <script> const loadingTask = pdfjs.getDocument('example.pdf'); loadingTask.promise.then(pdf => { pdf.getPage(1).then(page => { page.getTextContent().then(textContent => { textContent.items.forEach(item => { console.log(item.str); }); }); }); }); </pre> <p>In this example, <code>pdfjs.getDocument()</code> is a function that loads the PDF document. <code>pdf.getPage()</code> retrieves the desired page (in this case, page 1), and <code>getTextContent()</code> retrieves the text content for that page. The <code>textContent</code> object contains an array of <code>items</code>, where each item represents an element in the parsed PDF document structure. The <code>str</code> property of each item contains the actual content of the element.</p> <p>You can then iterate over the <code>items</code> array and access the <code>str</code> property to get the content for each element in the structure.</p> <p>It seems that you are encountering an error related to the DOMMatrix class not being defined. This class is imported from the npm library <code>"canvas"</code> in the <code>compatibility.js</code> file of the <code>pdfjs-dist</code> library.</p> <p>To resolve this issue, you can try the following steps:</p> <ol style="list-style-type: none"> Make sure you have the <code>"canvas"</code> package installed in your project. You can install it by running the following command in your project directory: <pre>npm install canvas</pre> Check if you have the correct version of the <code>"canvas"</code> package installed. The DOMMatrix class was introduced in newer versions of the <code>"canvas"</code> package. You can verify the installed version by checking the <code>package.json</code> file or running the following command: <pre>npm list canvas</pre> If you have an older version of the <code>"canvas"</code> package installed, try updating it to the latest version by running the following command: <pre>npm update canvas</pre> Restart your npm server to ensure that the changes take effect. <p>If the issue still persists after following these steps, it's possible that there might be a compatibility issue between the <code>pdfjs-dist</code> library and the <code>canvas</code> package. In that case, you may need to investigate further or consider using a different version of the <code>pdfjs-dist</code> library.</p>	Slightly satisfied	Moderately satisfied

C

A Comparison of Conversational Models and Humans in Answering Technical Questions: the Firefox Case

This appendix provides additional details regarding our third study (see Chapter 5). This study includes references to external links that provide access to the supplementary material presented in this section.

C.1

Example Form of the Assessment of Questions and Answers

The following form is an example used by one expert to assess the answers generated by different sources to technical questions.

D1 - Manual Assessment - Mozilla

We invite you to participate in a anonymous and voluntary questionnaire aimed at gathering your assessment of alternative answers given to technical questions made in Matrix channels regarding the Firefox project.

For each technical question, you will be presented a set of possible answers to this question. Initially, you should evaluate the alternative answers based on three criteria:

Helpfulness: the quality of providing assistance to address the question.

Comprehensiveness: the quality of including everything that is needed to answer the question.

Conciseness: the quality of including everything that is necessary for answering the question without using unnecessary words.

Additionally, there will be an open-ended question where you can share further insights to help us better understand the effectiveness of the various answers.

At the end of the survey, we would like to learn more about your background to better understand your experience and perspective.

* Indicates required question

Question 1 from Firefox Matrix.

<suppressed_user> Hi team 🙌,

Hope everybody is doing well.

We have a couple of [crash reports](https://bugzilla.mozilla.org/show_bug.cgi?id=1873619) on Android, coming from `libutils.so`, with the below stack.

Is somebody familiar with this?

...

Reason: SIGSEGV / SEGV_MAPERR

0 libutils.so libutils.so@0xede0

1 boot.oat boot.oat@0x3844a0

2 ? @0x00006e90148652e4

...

Thanks in advance

1. Answer 1:

Hi 🐼, the crash reports indicate a segmentation fault (SIGSEGV) in libutils.so, which is a common library in Android's native code. The specific address libutils.so@0xede0 suggests an issue within that library, but without more context, it's challenging to pinpoint the exact cause. This could be due to memory corruption, invalid memory access, or a bug in the library itself. It would be helpful to gather more information, such as the Android version, device model, and steps to reproduce the crash. Additionally, checking for updates or patches for libutils.so might provide a solution.

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

2. Answer 2:

The stacks look rather bad. Maybe <suppressed_user> would have some ideas of what could be going wrong. The devices the crashes are on all have the same manufacturer, so maybe that means something.

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

3. Answer 3:

Hi <suppressed_user>,

The crash reports you're seeing are likely due to a segmentation fault in `libutils.so`. This is a known issue and has been reported in Bugzilla under [Bug 1873619] (https://bugzilla.mozilla.org/show_bug.cgi?id=1873619). You may want to follow the updates on that bug report for further information and potential fixes.

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

4. Please compare the answers:

Mark only one oval per row.

	Answer 1	Answer 2	Answer 3
Which answer provides the most helpful information?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer provides the most comprehensive information?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer is the most concise without being incomplete?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer do you prefer to see in practice?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

5. Regarding the answer you prefer to see in practice, we encourage you to emphasize all particular aspects that are (ir)relevant in the answer(s) and aspects you think it missed.

Question 2 from Firefox Matrix.

<suppressed_user> Is there any really really simple place to store an int that isnt prefs? I want to store a timestamp preferably with more than seconds precision (so I dont want to just /1000 it)

6. Answer 1:

You can use the ``nsIXULRuntime.data`` object to store an int with more than seconds precision. This object allows you to store arbitrary data types, including integers, and is accessible across different parts of the codebase. This method avoids using preferences and provides the precision you need for a timestamp.

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

7. Answer 2:

You can use the nsIXULStore interface to store simple key-value pairs, including integers, with more precision than seconds. This interface is designed for storing small amounts of data and is simpler than using preferences. You can access it via JavaScript in your Firefox extension or application. Here's a quick example:

```
let xulStore = Components.classes["@mozilla.org/xul/xulstore;1"]
    .getService(Components.interfaces.nsIXULStore);
xulStore.setValue("your-namespace", "your-key", "your-timestamp-value");
let timestamp = xulStore.getValue("your-namespace", "your-key");
This allows you to store and retrieve your timestamp with the desired precision
without using preferences.
```

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

8. Answer 3:

What's wrong with prefs? if your data is associated with a site you could use the content pref service

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

9. Please compare the answers:

Mark only one oval per row.

	Answer 1	Answer 2	Answer 3
Which answer provides the most helpful information?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer provides the most comprehensive information?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer is the most concise without being incomplete?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer do you prefer to see in practice?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

10. Regarding the answer you prefer to see in practice, we encourage you to emphasize all particular aspects that are (ir)relevant in the answer(s) and aspects you think it missed.

Question 3 from Firefox Matrix.

<suppressed_user> Is there a helper function somewhere in the frontend to open a new browser window (BROWSER_CHROME_URL), and return a promise which resolves when gBrowser is available in that new window?

11. Answer 1:

Yes, you can use the BrowserTestUtils.openNewBrowserWindow function, which opens a new browser window and returns a promise that resolves when the new window's gBrowser is available.

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

12. Answer 2:

Yes, you can use the `OpenBrowserWindow` function from `browser.js` to open a new browser window with `BROWSER_CHROME_URL`. It returns a promise that resolves when `gBrowser` is available in the new window.

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

13. Answer 3:

BrowserTestUtils has some such thing but don't know the name off hand (on the phone sorry)

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

14. Please compare the answers:

Mark only one oval per row.

	Answer 1	Answer 2	Answer 3
Which answer provides the most helpful information?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer provides the most comprehensive information?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer is the most concise without being incomplete?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer do you prefer to see in practice?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

15. Regarding the answer you prefer to see in practice, we encourage you to emphasize all particular aspects that are (ir)relevant in the answer(s) and aspects you think it missed.

Question 4 from Firefox Matrix.

Does someone here know what one can do about this? After upgrading to Firefox 121, it appears macOS is unwilling to allow Firefox to be started from the command-line, thus making e.g. unit tests via Karma and such impossible.

<https://github.com/karma-runner/karma-firefox-launcher/issues/328#issuecomment-1885113046>

macOS is not allowing the program to run? Not even for `--version`, or with no arguments.

```
$ /Applications/Firefox.app/Contents/MacOS/firefox-bin  
Killed: 9
```

16. Answer 1:

Based on vaguely related reports at <https://apple.stackexchange.com/questions/258623/how-to-fix-killed-9-error-in-mac-os> and https://bugzilla.mozilla.org/show_bug.cgi?id=1704126, I'm guessing this has something to do with the way Mozilla signed the Firefox binary, and/or the way Firefox replaced its own binary (e.g. mv instead of cp or something like that)

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

17. Answer 2:

To address the issue of Firefox 121 being unable to start from the command line on macOS, try running the following command to remove the quarantine attribute: ``xattr -d com.apple.quarantine /Applications/Firefox.app``. If this does not resolve the issue, consider downgrading Firefox or checking for updates on the related GitHub issue for any new fixes or workarounds.

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

18. Answer 3:

The issue you're encountering is likely due to macOS's Notarization and Hardened Runtime requirements, which prevent Firefox from being started from the command line without proper entitlements. To work around this, you can either disable System Integrity Protection (SIP), which is not recommended due to security risks, or use a local build or a try build of Firefox that does not have these restrictions. For local builds, ensure you follow the instructions for creating a debuggable build and signing it appropriately. You can find more details on building and signing local macOS builds in the Firefox developer documentation.

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

19. Please compare the answers:

Mark only one oval per row.

	Answer 1	Answer 2	Answer 3
Which answer provides the most helpful information?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer provides the most comprehensive information?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer is the most concise without being incomplete?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer do you prefer to see in practice?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

20. Regarding the answer you prefer to see in practice, we encourage you to emphasize all particular aspects that are (ir)relevant in the answer(s) and aspects you think it missed.

Question 5 from Firefox Matrix.

<suppressed_user> Hello everyone,

I'm looking to perform a full rebranding of the Firefox application and package it as my custom browser.

I have already built the firefox on Linux and Windows, and was able to make name changes by changing MOZ_APP_NAME in several locations, it did changed the name, but the process running is still firefox-default on Linux and Nightly on Windows

Could someone point me to the relevant documentation or provide guidance on how to accomplish this?

Thank you!

21. Answer 1:

To fully rebrand Firefox and ensure that the process names reflect your custom browser, you need to modify the branding files and possibly some build configuration files. Start by updating the branding directory under browser/branding with your custom icons, logos, and names. Ensure you change MOZ_APP_NAME and MOZ_APP_DISPLAYNAME in the moz.build and confvars.sh files. Additionally, you may need to update the application.ini file to reflect your custom names. For the process names, you might need to modify the firefox.exe and related binaries directly, which can be done by editing the source code where these names are defined, typically in nsAppRunner.cpp and other related files. For more detailed guidance, refer to the Mozilla Build Documentation (https://firefox-source-docs.mozilla.org/contributing/directory_structure.html) and the branding documentation (<https://searchfox.org/mozilla-central/source/browser/branding>).

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

22. Answer 2:

I suck at explaining stuff, but I have my own rebranded fork, and you should be able to figure out what I did by looking at commit history <https://github.com/Eclipse-Community/r3dfox/commits/release/> Or wait for me to get around to 128 and I'll combine everything new into a single commit to look at

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

23. Answer 3:

To fully rebrand Firefox and ensure the process names reflect your custom browser, you need to modify the `MOZ_APP_DISPLAYNAME` and `MOZ_APP_BASENAME` variables in the relevant `moz.build` files and branding directories. Additionally, check [`package-name.mk`](#) and `application.ini` for any remaining references to "firefox" or "Nightly". For detailed guidance, refer to the [Firefox Build Documentation](<https://firefox-source-docs.mozilla.org/>).

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

24. Please compare the answers:

Mark only one oval per row.

	Answer 1	Answer 2	Answer 3
Which answer provides the most helpful information?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer provides the most comprehensive information?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer is the most concise without being incomplete?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer do you prefer to see in practice?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

25. Regarding the answer you prefer to see in practice, we encourage you to emphasize all particular aspects that are (ir)relevant in the answer(s) and aspects you think it missed.

Question 6 from Firefox Matrix.

<suppressed_user> Can I enable a logger I'd normally use via `MOZ_LOG` for a mochitest suite on try too? do I just pass in the env somehow?

26. Answer 1:

Yes, you can enable a logger for a mochitest suite on try by setting the MOZ_LOG environment variable. To do this, modify the buildBrowserEnv method in the Python file for the test suite you are interested in and add the necessary environment variable settings. For example, you can add browserEnv["MOZ_LOG"] = "example_logger:3" to set the logging level for example_logger. Additionally, you can set MOZ_LOG_FILE to specify the log file path. This will ensure that the logs are captured and available for download from TreeHerder after the try run.

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

27. Answer 2:

Yes, with `--env` in the `mach try` invocation, IIRC I'd hope we have docs for this

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

28. Answer 3:

Yes, you can enable a logger via `MOZ_LOG` for a mochitest suite on try by setting the environment variable in the task configuration. You can do this by adding `mozhariness` options in your `try` syntax, such as `try: -b do -p all -u mochitest --env MOZ_LOG=<module>:<level>`.

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

29. Please compare the answers:

Mark only one oval per row.

	Answer 1	Answer 2	Answer 3
Which answer provides the most helpful information?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer provides the most comprehensive information?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer is the most concise without being incomplete?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer do you prefer to see in practice?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

30. Regarding the answer you prefer to see in practice, we encourage you to emphasize all particular aspects that are (ir)relevant in the answer(s) and aspects you think it missed.

Question 7 from Firefox Matrix.

<suppressed_user> My local mozilla-central builds are failing today with an `error[E0635]:
unknown feature stdsimd`, on two separate machines, but I don't know what's changed
recently to cause this anyone able to point me in the right direction?

31. Answer 1:

You're using nightly rust, is why

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

32. Answer 2:

The error error[E0635]: unknown feature stdsimd typically indicates that the Rust compiler version you're using does not support the stdsimd feature. To resolve this, ensure that your Rust toolchain is up-to-date by running rustup update. Additionally, verify that your rust-toolchain file in the mozilla-central directory specifies the correct Rust version. If the issue persists, you might want to check recent changes in the mozilla-central repository or consult the build documentation for any updates regarding Rust dependencies.

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

33. Answer 3:

The ``error[E0635]: unknown feature stdsimd`` indicates that the ``stdsimd`` feature is not recognized. This could be due to recent changes in the Rust compiler or the Mozilla codebase. To resolve this, ensure you are using the latest version of Rust by running ``rustup update``, and also update your local mozilla-central repository by running ``./mach bootstrap`` followed by ``./mach build``. If the issue persists, check the Mozilla build documentation or the recent commit history for any related changes.

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

34. Please compare the answers:

Mark only one oval per row.

	Answer 1	Answer 2	Answer 3
Which answer provides the most helpful information?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer provides the most comprehensive information?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer is the most concise without being incomplete?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer do you prefer to see in practice?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

35. Regarding the answer you prefer to see in practice, we encourage you to emphasize all particular aspects that are (ir)relevant in the answer(s) and aspects you think it missed.

Question 8 from Firefox Matrix.

<suppressed_user> Hello guys,
I have send a patch (my first here) for review more than two weeks ago but still didn't get any
reviews. Is there anything that I did wrong? Should I add more reviewers? Ping people?

36. Answer 1:

Hi <suppressed_user>, it's common for reviews to take some time. You can gently
ping the reviewers on the Bugzilla bug page or add one or two more relevant
reviewers to get attention. Be sure to provide context and express your willingness
to make any needed changes.

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

37. Answer 2:

It looks like you did the right thing on the patch. You could try pinging the reviewer
here or needinfo them in bugzilla. We do also have a gfx-reviewer reviewer group
but that's probably not quite the right thing for imagelib things.

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

38. Answer 3:

If your patch has been waiting for a review for more than two weeks, you should follow the "Getting attention" section guidelines. Add a comment to the bug saying 'review ping?', check the "Need more information from" box, and add the reviewer's name. If there is still no response within a day or two, you can ask for help on Matrix in the #introduction:[mozilla.org](https://matrix.to/#/#introduction:mozilla.org) or #developers:[mozilla.org](https://matrix.to/#/#developers:mozilla.org) channels. Additionally, you can tag more reviewers or use review groups like #geckoview-reviewers in Phabricator to notify more people.

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

39. Please compare the answers:

Mark only one oval per row.

	Answer 1	Answer 2	Answer 3
Which answer provides the most helpful information?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer provides the most comprehensive information?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer is the most concise without being incomplete?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer do you prefer to see in practice?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

40. Regarding the answer you prefer to see in practice, we encourage you to emphasize all particular aspects that are (ir)relevant in the answer(s) and aspects you think it missed.

Question 9 from Firefox Matrix.

<suppressed_user> I have a problem that's hard to debug -- `./mach test toolkit/components/extensions/test/xpcshell/test_ext_webRequest_responseBody.js` --- I get a timeout on the initial test in parallel. On the retry (`Retrying tests that failed when run in parallel`) to run the xpcshell tests sequentially, I get an assertion. What I really want it to get an rr trace of the assertion, but if I use --debugger=rr, it never times out the initial test, so we never get to the one that fails.

Any suggestions?

...

12345670:01.02 SUITE_START: xpcshell - running 2 tests
0:01.02 TEST_START: xpcshell-
e10s.toml:toolkit/components/extensions/test/xpcshell/test_ext_webRequest_responseBod
y.js
0:01.02 TEST_START: xpcshell-
remote.toml:toolkit/components/extensions/test/xpcshell/test_ext_webRequest_responseB
ody.js
5:01.02 TEST_END: TIMEOUT
5:01.02 INFO xpcshell return code: None
5:01.02 TEST_END: TIMEOUT
5:01.02 INFO xpcshell return code: None
...

(and then on the retry after a timeout I get the IOThread assertion)

41. Answer 1:

You can try running the test with `--log-tbpl` to get more detailed logs, which might help identify the root cause of the timeout and the assertion failure. Additionally, consider using `rr record` to capture the trace of the initial run, even if it doesn't fail, and then use `rr replay` to debug the subsequent assertion. This way, you might capture more context leading up to the failure.

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

42. Answer 2:

To debug the issue, try running the test with the `--sequential` flag to avoid parallel execution, which might help isolate the problem. Additionally, you can use the `MOZ_DEBUG_CHILD_PROCESS=1` environment variable to pause the child process and attach a debugger to it. If the test still times out, consider adding more logging to identify where it hangs. If you need an rr trace, ensure the test runs long enough to capture the failure by adjusting timeouts or adding delays. Finally, verify if adding `firefox-appdir = browser` to the `xpcshell.toml` manifest resolves any resource access issues.

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

43. Answer 3:

Maybe rr chaos mode helps? Something like `--debugger-args='record --chaos'`

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

44. Please compare the answers:

Mark only one oval per row.

	Answer 1	Answer 2	Answer 3
Which answer provides the most helpful information?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer provides the most comprehensive information?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer is the most concise without being incomplete?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer do you prefer to see in practice?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

45. Regarding the answer you prefer to see in practice, we encourage you to emphasize all particular aspects that are (ir)relevant in the answer(s) and aspects you think it missed.

Question 10 from Firefox Matrix.

<suppressed_user> Hi, I am trying to debug an issue with fenix on my Pixel 6a where it crashes. I wanted to run a debug build to get a stack trace. I set up the build and it completed without errors, but when I start the intent it immediately fails with no output (the log in android studio is just connected/disconnected from target VM). Any advice on how to debug this? Is my phone haunted?

46. Answer 1:

To debug the issue with Fenix crashing on your Pixel 6a, first ensure that USB debugging is enabled on your device and that you have the necessary permissions. Use adb logcat to capture detailed logs from your device, which can provide more insight into the crash. You can also try running the app directly from the command line using `./gradlew :app:installFenixDebug` and then start it with `adb shell am start -n org.mozilla.fenix.debug/org.mozilla.fenix.debug.App`. If the issue persists, check for any missing dependencies or misconfigurations in your build setup. Additionally, ensure that your Android SDK and Java paths are correctly set. If none of these steps resolve the issue, consider reaching out to the Fenix development community for further assistance.

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

47. Answer 2:

Maybe check on GeckoView. There's nothing in the logcat?

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

48. Answer 3:

To debug the issue with fenix crashing on your Pixel 6a, try enabling verbose logging in Android Studio by going to "Run" > "Edit Configurations" and setting the log level to "Verbose". Additionally, make sure your phone's developer options have "USB Debugging" enabled, and try running ``adb logcat`` from the terminal to capture detailed logs. If the app crashes before any log output, consider adding breakpoints in the initialization code to narrow down the crash point.

Check all that apply.

	Helpful	Comprehensive	Concise
This answer is:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

49. Please compare the answers:

Mark only one oval per row.

	Answer 1	Answer 2	Answer 3
Which answer provides the most helpful information?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer provides the most comprehensive information?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer is the most concise without being incomplete?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Which answer do you prefer to see in practice?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

50. Regarding the answer you prefer to see in practice, we encourage you to emphasize all particular aspects that are (ir)relevant in the answer(s) and aspects you think it missed.

Characterization

Thank you for your valuable collaboration! In this section, we would like to learn more about your background to better understand your experience and perspective.

51. How confident were you in assessing the answers?

Mark only one oval.

	1	2	3	4	5	
Not	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very confident

52. How many years of professional experience do you have? *

Mark only one oval.

- ☐ Less than 1 year
- ☐ 1-3 years
- ☐ 4-7 years
- ☐ 8-10 years
- ☐ More than 10 years

53. How many years have you been contributing to Mozilla projects? *

Mark only one oval.

- ☐ Less than 1 year
- ☐ 1-3 years
- ☐ 4-7 years
- ☐ 8-10 years
- ☐ More than 10 years

54. What is your gender?

55. How familiar are you with answering technical questions in Matrix channels?

Mark only one oval.

1 2 3 4 5

Not ☐ ☐ ☐ ☐ ☐ Very familiar

56. How do you contribute to support newcomers in Mozilla projects?

This content is neither created nor endorsed by Google.

Google Forms

C.2

Full List of Questions, Answers and Experts' Assessment

The spreadsheet below depicts the complete list of questions in the study, the answers, and the expert's assessment.

[illegible]

[illegible]

[illegible]

Answer (GPT4o) gives correct information but a dead link that points to a section that doesn't exist.

Answer (Human) points to a link that works, but doesn't explain the difference

Answer (GPT) hallucinates the link and doesn't provide the information

[illegible]

They are all terrible responses.

C.3

Overlapping List of Questions, Answers and Experts' Assessment

The following file contains questions used to verify the agreement between experts during the assessment process.

[illegible]

[illegible]

D

Developers' Perceptions of Automated Detection and Resolution of Software Dependability Issues

This appendix provides additional details regarding our fourth study (see Chapter 6). This study includes references to external links that provide access to the supplementary material presented in this section.

D.1

Full Agents' Prompts

In the following file, we present the tailored prompts for each agent composing the pipeline to detect and fix dependability issues in source code.

Issue Detector

ROLE

You are senior Python developer specialized in building highly dependable systems. Perform the TASK using the CONTEXT provi

TASK

Perform a thorough code review of the code file below, focused on detecting **dependability** issues – i.e., problems that

- reliability (consistent correct behavior)
- availability (system readiness)
- safety (preventing harmful consequences)
- confidentiality (avoiding unauthorized access)
- integrity (ensuring data correctness)
- maintainability (ease of fixing/evolving)

Prefer concrete findings over vague style nits. If uncertain, do not hallucinate.

FILE

File name: ```[filename]```\nCode content: ```[filedata]```

OUTPUT

For each issue you find, output a JSON object with these fields:

1. title: Short summary of the issue.
2. location: the file where the issue occurs, ex: filename.py.
3. code: (Read all the code and) Provide the FULL METHOD where the problem happens in a Markdown format.
4. issue: Description of the dependability issue.
5. impact: Possible consequences of the issue.
6. label: bug, enhancement, docs, refactor, test ...

EXAMPLE

```
{
  "title": "Handle missing config file",
  "location": "config_loader.py",
  "code": "def update_data(id: int, new_value: float, storage: dict) -> None:\n
    if not (0 <= new_value <= 100):\n
      raise ValueError(f\"Invalid id value: {new_value}\")\n\n
    # Update the storage\n
    if id in storage:\n
      storage[id].append(new_value)\n
    else:\n
      storage[id] = [new_value]`",
  "issue": "No exception handling if config file is absent.",
  "impact": "Application will crash with FileNotFoundError.",
  "label": "bug"
}
```

CONTEXT: [Automatically inserted by RAG]

Issue Validator

ROLE

You are a senior Python developer specialized in building highly dependable systems.

TASK

Thinking step by step, critically evaluate whether a reported issue truly qualifies as a **DEPENDABILITY** issue.

Dependability issues are problems that significantly affect one or more of the following:

- reliability (consistent correct behavior)
- availability (system readiness)
- safety (preventing harmful consequences)
- confidentiality (avoiding unauthorized access)
- integrity (ensuring data correctness)
- maintainability (ease of fixing/evolving)

Reported Issue:

Title: [title]\nLocation: [location]\nCode: [code]\nIssue Description: [issue]\nImpact: [impact]

Based on the above, do you confirm this is a **VALID** and **SIGNIFICANT** dependability issue?

- Reject issues such as code style
- Reject **TRIVIAL** optimizations
- Reject changes with **NEGLEGIBLE** impact that do not compromise the system's dependability
- If uncertain, do not hallucinate

OUTPUT

Please, write exactly one paragraph that summarizes your rationale.

At the end, you **MUST** output a json containing only two fields: 'description' and 'classification'.

EXAMPLE

```
{
  "description": "Paragraph justifying here ...",
  "classification": "Confirm/Reject"
}
```

Code Fixer

ROLE

You are senior Python developer specialized in building highly dependable systems.

TASK

Thinking critically step by step, evaluate the provided dependability issue and then exploring the context fix the code.

Title: [title]

Issue Description: [description]

Code: [code]

OUTPUT

Output a json containing only two fields: 'description' and 'fixed_code'.

EXAMPLE

```
{
  "description": "The soft delete implementation does not prevent access to deleted records. Users may still query these
  "fixed_code": "def get_queryset(self):\n    return super().get_queryset().filter(deleted_at__isnull=True).exclude(delet
}
```

CONTEXT: [Automatically inserted by RAG]

Code Judge

ROLE

You are a senior Python developer specialized in building highly dependable systems.

TASK

Thinking step by step, critically evaluate whether a reported issue was fixed.

Dependability issues are problems that significantly affect one or more of the following:

- reliability (consistent correct behavior)
- availability (system readiness)
- safety (preventing harmful consequences)
- confidentiality (avoiding unauthorized access)
- integrity (ensuring data correctness)
- maintainability (ease of fixing/evolving)

Reported Issue:

- Title: [title]
- Location: [location]
- Code: [code]
- Issue Description: [description]
- Impact: [impact]
- Solution: [proposed solution]

Based on the above, do you confirm this issue was COMPLETELY fixed? If uncertain, do not hallucinate.

OUTPUT

Please, write exactly one paragraph that summarizes your rationale. At the end, you MUST output a json containing only two

EXAMPLE

```
{
  "description": "Paragraph justifying here ...",
  "classification": "Confirm/Reject"
}
```

D.2

Full List of Dependability Issues and Fixes

The spreadsheet below shows the complete list of dependability issues detected by the pipeline. Details were suppressed due to privacy constraints.

Issue ID	Open Coding: Issue Description	I understand that the stated problem is relevant and worth addressing [P1]	I understand that the stated problem is relevant and worth addressing [P2]	I would agree this path is a viable option [P3]	I am familiar with this code [P4]	Open Coding: P1 Feedback	I understand that the stated problem is relevant and worth addressing [P5]	I understand that the stated problem is relevant and worth addressing [P6]	I would agree this path is a viable option [P7]	I am familiar with this code [P8]	Open Coding: P2 Feedback
1	Missing validation level validation, based on how many errors feedback	Agree	Agree	Agree	Agree	problem oriented (positive); manual verification needed in some cases (negative); less critical for private APIs (negative)	Agree	Agree	Agree	Agree	affordable solution (case (positive); extensive validation across problem (negative); not a major problem (negative); not a major problem (negative)
2	Missing boundary validation in exception, no structural validation errors, boundary limited to English	Agree	Agree	Agree	Agree	Missing context on exception problem (negative); internal default message (negative); problem remains unclear (negative); partial implementation on new (negative); unclear implementation not needed in the specific case but previous (negative)	Agree	Agree	Agree	Agree	minor issue (negative); improvement necessary error feedback (positive)
3	API error due to missing request context, no fallback for absent context, data depending on IDB context	Agree	Neutral	Agree	Agree	Valid issue acknowledged (positive); unclear implementation (negative); developer and handle properly (negative)	Agree	Agree	Agree	Agree	Missing check that in validator (positive); the parameter is not implemented in the validator (negative); importance of difference (negative); context (negative)
4	Unauthenticated user management risk, no fallback for absent context, data depending on IDB context	Disagree	Agree	Agree	Agree	code changes and it can be a problem in the authentication of that point (negative)	Agree	Agree	Agree	Agree	critical security issue (positive); simple and effective fix (positive)
5	PathError from absent context, no dynamic log, observed before implementation	Disagree	Agree	Agree	Agree	specific case implementation (negative); less critical due to current design (negative)	Agree	Agree	Agree	Agree	non-critical issue (negative); all valuable to check (positive)
6	IDB key truncation risk, conflicts with how IDB, no fallback for absent context, data depending on IDB context	Agree	Agree	Agree	Agree	Invalid exception due to (negative); truncation of null/blank (positive); fix requires both issues (positive); approval without null/blank (positive)	Agree	Agree	Agree	Agree	good prevention fix (positive); future error due to clear error (neutral)
7	API error from missing request in context, missing definition check in context, context acknowledged without	Agree	Agree	Agree	Agree	Invalid exception due to (negative); Invalid parameter check (positive); path improved with design named (positive)	Agree	Agree	Agree	Agree	parameter updated (positive)
8	API error from missing request in context, missing definition check in context, context acknowledged without	Agree	Agree	Agree	Agree	Invalid exception due to (negative); Invalid parameter check (positive); path improved with design named (positive)	Agree	Agree	Agree	Agree	parameter updated (positive)
9	API error from missing request in context, missing definition check in context, context acknowledged without	Agree	Agree	Agree	Agree	Invalid exception due to (negative); Invalid parameter check (positive); path improved with design named (positive)	Agree	Agree	Agree	Agree	parameter updated (positive)
10	API error from missing request in context, missing definition check in context, context acknowledged without	Agree	Agree	Agree	Agree	Invalid exception due to (negative); Invalid parameter check (positive); path improved with design named (positive)	Agree	Agree	Agree	Agree	parameter updated (positive)
11	API error from missing request in context, missing definition check in context, context acknowledged without	Agree	Agree	Agree	Agree	Invalid exception due to (negative); Invalid parameter check (positive); path improved with design named (positive)	Agree	Agree	Agree	Agree	parameter updated (positive)
12	API error from missing request in context, missing definition check in context, context acknowledged without	Agree	Agree	Agree	Agree	Invalid exception due to (negative); Invalid parameter check (positive); path improved with design named (positive)	Agree	Agree	Agree	Agree	parameter updated (positive)
13	API error from missing request in context, missing definition check in context, context acknowledged without	Agree	Agree	Agree	Agree	Invalid exception due to (negative); Invalid parameter check (positive); path improved with design named (positive)	Agree	Agree	Agree	Agree	parameter updated (positive)
14	API error from missing request in context, missing definition check in context, context acknowledged without	Agree	Agree	Agree	Agree	Invalid exception due to (negative); Invalid parameter check (positive); path improved with design named (positive)	Agree	Agree	Agree	Agree	parameter updated (positive)
15	API error from missing request in context, missing definition check in context, context acknowledged without	Agree	Agree	Agree	Agree	Invalid exception due to (negative); Invalid parameter check (positive); path improved with design named (positive)	Agree	Agree	Agree	Agree	parameter updated (positive)
16	API error from missing request in context, missing definition check in context, context acknowledged without	Agree	Agree	Agree	Agree	Invalid exception due to (negative); Invalid parameter check (positive); path improved with design named (positive)	Agree	Agree	Agree	Agree	parameter updated (positive)
17	API error from missing request in context, missing definition check in context, context acknowledged without	Agree	Agree	Agree	Agree	Invalid exception due to (negative); Invalid parameter check (positive); path improved with design named (positive)	Agree	Agree	Agree	Agree	parameter updated (positive)
18	API error from missing request in context, missing definition check in context, context acknowledged without	Agree	Agree	Agree	Agree	Invalid exception due to (negative); Invalid parameter check (positive); path improved with design named (positive)	Agree	Agree	Agree	Agree	parameter updated (positive)
19	API error from missing request in context, missing definition check in context, context acknowledged without	Agree	Agree	Agree	Agree	Invalid exception due to (negative); Invalid parameter check (positive); path improved with design named (positive)	Agree	Agree	Agree	Agree	parameter updated (positive)
20	API error from missing request in context, missing definition check in context, context acknowledged without	Agree	Agree	Agree	Agree	Invalid exception due to (negative); Invalid parameter check (positive); path improved with design named (positive)	Agree	Agree	Agree	Agree	parameter updated (positive)
21	API error from missing request in context, missing definition check in context, context acknowledged without	Agree	Agree	Agree	Agree	Invalid exception due to (negative); Invalid parameter check (positive); path improved with design named (positive)	Agree	Agree	Agree	Agree	parameter updated (positive)
22	API error from missing request in context, missing definition check in context, context acknowledged without	Agree	Agree	Agree	Agree	Invalid exception due to (negative); Invalid parameter check (positive); path improved with design named (positive)	Agree	Agree	Agree	Agree	parameter updated (positive)
23	API error from missing request in context, missing definition check in context, context acknowledged without	Agree	Agree	Agree	Agree	Invalid exception due to (negative); Invalid parameter check (positive); path improved with design named (positive)	Agree	Agree	Agree	Agree	parameter updated (positive)
24	API error from missing request in context, missing definition check in context, context acknowledged without	Agree	Agree	Agree	Agree	Invalid exception due to (negative); Invalid parameter check (positive); path improved with design named (positive)	Agree	Agree	Agree	Agree	parameter updated (positive)
25	API error from missing request in context, missing definition check in context, context acknowledged without	Agree	Agree	Agree	Agree	Invalid exception due to (negative); Invalid parameter check (positive); path improved with design named (positive)	Agree	Agree	Agree	Agree	parameter updated (positive)
26	API error from missing request in context, missing definition check in context, context acknowledged without	Agree	Agree	Agree	Agree	Invalid exception due to (negative); Invalid parameter check (positive); path improved with design named (positive)	Agree	Agree	Agree	Agree	parameter updated (positive)
27	API error from missing request in context, missing definition check in context, context acknowledged without	Agree	Agree	Agree	Agree	Invalid exception due to (negative); Invalid parameter check (positive); path improved with design named (positive)	Agree	Agree	Agree	Agree	parameter updated (positive)
28	API error from missing request in context, missing definition check in context, context acknowledged without	Agree	Agree	Agree	Agree	Invalid exception due to (negative); Invalid parameter check (positive); path improved with design named (positive)	Agree	Agree	Agree	Agree	parameter updated (positive)
29	API error from missing request in context, missing definition check in context, context acknowledged without	Agree	Agree	Agree	Agree	Invalid exception due to (negative); Invalid parameter check (positive); path improved with design named (positive)	Agree	Agree	Agree	Agree	parameter updated (positive)
30	API error from missing request in context, missing definition check in context, context acknowledged without	Agree	Agree	Agree	Agree	Invalid exception due to (negative); Invalid parameter check (positive); path improved with design named (positive)	Agree	Agree	Agree	Agree	parameter updated (positive)
31	API error from missing request in context, missing definition check in context, context acknowledged without	Agree	Agree	Agree	Agree	Invalid exception due to (negative); Invalid parameter check (positive); path improved with design named (positive)	Agree	Agree	Agree	Agree	parameter updated (positive)

29	if test fails sometimes (negative) (in permitted, schema allows invalid change values)	Strongly Agree	Agree	Agree	Strongly Agree	Enforce enforcement of ID constraints (positive); application rule enforcement (positive)	Strongly Agree	Strongly Agree	Strongly Agree	Neutral	
30	log responses (SUPPRESSLOGS) information, in violation of PII or logs, privacy violation rule	Agree	Strongly Agree	Agree	Agree	removes Personal identifiable information from logs (positive); maintains traceability (positive); privacy improvement (positive)	Agree	Agree	Agree	Neutral	removes data in (SUPPRESSLOGS) (positive); critical privacy concern (positive); internal log preferred for traceability (negative)
31	error messages sometimes diagnostic detail, adds extra error messages only, responses internal (24 hours for health check)	Agree	Strongly Agree	Strongly Agree	Agree		Agree	Agree	Agree	Agree	removes data exposure (positive); maintainability concern (positive)
32	no constraints constraint on (SUPPRESSLOGS) but, new condition allows duplicates, eventual consistency feature	Strongly Agree	Agree	Strongly Agree	Agree		Strongly Agree	Agree	Agree	Neutral	
33	(SUPPRESSLOGS) not stopped or completed, inconsistent rule for formatting, variable matching	Agree	Agree	Agree	Agree		Strongly Agree	Agree	Agree	Strongly Agree	(SUPPRESSLOGS) format validation (positive); ensuring proper input (positive)
34	refresh token response in 201 response, variable secure storage, best practices, increases (SSO) token refresh	Agree	Disagree	Disagree	Agree	incomplete solution (negative); refresh token removal insufficient (negative)	Strongly Agree	Agree	Strongly Agree	Strongly Agree	refresh token rotation policy (positive)
35	(SUPPRESSLOGS) domain check, improved for validation, improper parsing of domain validation	Strongly Agree	Strongly Agree	Strongly Agree	Strongly Agree		Strongly Agree	Strongly Agree	Strongly Agree	Strongly Agree	
36	HTTPS, log detection for various, improved, logs with enforcement, alongside information rule	Strongly Agree	Strongly Agree	Strongly Agree	Strongly Agree		Agree	Agree	Neutral	Disagree	
37	internal, theory value of times, responses (change depth, breaks individuality) expectations	Neutral	Neutral	Agree	Neutral		Agree	Neutral	Neutral	Disagree	
38	ONLY-based POC parsing, overall, many correlation rule, (internal) internal, simple on (rule) rule	Agree	Agree	Agree	Agree		Agree	Agree	Agree	Strongly Disagree	
39	cannot, response interpretation, no file rule, response, allow path transfer in (SUPPRESSLOGS)	Strongly Agree	Strongly Agree	Strongly Agree	Strongly Agree		Strongly Agree	Agree	Agree	Strongly Disagree	
40	self-defined test values, model allows hard values, no enforcement of authentication	Strongly Agree	Strongly Agree	Strongly Agree	Strongly Agree		Agree	Neutral	Neutral	Strongly Agree	self-defined logs understood (positive); maintenance in queries accordingly (positive)

D.3

Full Colab Implementation for the Multi Agent Pipeline to Detect and Repair Dependability issues

The file below contains the full implementation of the multi agent pipeline to detect and fix dependability issues in source code.

Requirements

Attention: This Colab must be executed with a local runtime, as it needs to access endpoints exposed locally by Cognita, the framework that powers the RAG component.

Steps to reproduce

1. Download and setup locally the following third party tools:
 - [Python3](#)
 - [Jupyter Notebook](#)
 - [Cognita](#)
2. Connect this environment to your Jupyter Notebook host.
3. Set up Cognita by creating a collection (process your files into a vector database). Then, create an application that points to the newly created collection.

▼ Installing requirements

```
import sys
!{sys.executable} -m pip install openai
!{sys.executable} -m pip install --upgrade --quiet openai
!{sys.executable} -m pip install google-generativeai
!{sys.executable} -m pip install pandas
!{sys.executable} -m pip install --upgrade ipywidgets jupyterlab_widgets ipyfilechooser

import os
import re
import csv
import json
import requests
import time
import openai
import pandas as pd
import google.generativeai as genai
from datetime import datetime
```

```
module = 'ian_detection'
folder_path = "/Users/joaocorreia/TBD26/cognita/ian/"
output_path = f"/Users/joaocorreia/TBD26/cognita/detection/{module}_detection.csv"
output_path_filtered = f"/Users/joaocorreia/TBD26/cognita/detection/{module}_detection_filtered.csv"

model_gpt = "gpt-5-mini"
model_gpt_fix = "gpt-4o-mini"

model_gemini = 'gemini-2.5-flash-lite'

os.environ["OPENAI_API_KEY"] = ""
genai.configure(api_key="")
```

```
generate_config(expt_key= ,
```

✓ Collect files to analysis

The below cell starts a file server to receive files and pass it to the LLM.

```
import os
import random
import math

def collect_py_files(folder_path):
    uploaded_files = {}
    for root, dirs, files in os.walk(folder_path):

        # Apply filters for unwanted folders
        if "migrations" in [d.lower() for d in root.split(os.sep)]:
            continue

        for file in files:
            file_lower = file.lower()

            # Apply filters for unwanted files
            if ( file_lower.endswith(".py") and not any(skip in file_lower for skip
                file_path = os.path.join(root, file)
                rel_path = os.path.relpath(file_path, folder_path) # ✓ keep relat:

                with open(file_path, "r", encoding="utf-8") as f:
                    uploaded_files[rel_path] = f.read()

    return uploaded_files

def sample_size(N, confidence=0.90, margin_error=0.05, p=0.5):
    """
    Calculate sample size with finite population correction.

    Args:
        N (int): population size
        confidence (float): confidence level (0.90, 0.95, 0.99, etc.)
        margin_error (float): margin of error (e.g. 0.05 for ±5%)
        p (float): estimated proportion (default 0.5 for maximum variability)

    Returns:
        n (int): required sample size
        percent (float): percentage of population
    """
    # z-scores for common confidence levels
    z_table = {0.90: 1.645, 0.95: 1.960, 0.99: 2.576}
    Z = z_table.get(confidence)
    if Z is None:
        raise ValueError("Confidence must be one of: 0.90, 0.95, 0.99")

    # initial sample size (infinite population)
```

```

# initial sample size (infinite population)
n0 = (Z**2 * p * (1 - p)) / (margin_error**2)

# finite population correction
n = n0 / (1 + (n0 - 1) / N)

return math.ceil(n), round((n / N) * 100, 2)

uploaded_files = collect_py_files(folder_path)
N = len(uploaded_files)
# Assume uploaded_files is already filled
samples, percent = sample_size(N, confidence=0.95, margin_error=0.10)

print(f"✅ {N} Python files detected!")
print(f"🔍 Selecting {samples} files (~{percent}% of population)")

# Select random sample
selected_files = dict(random.sample(list(uploaded_files.items()), samples))

# Show results
for name in selected_files.keys():
    print(name)

```

▼ Run Issue Detection

Run RAG to detect dependability issues

```

gpt_detections = {}

def get_rag_answer(query: str, model_name: str = f"openai/{model_gpt}", temperature:
    k: int = 5, stream: bool = False, url: str = "http://localhost:8000/retrievers/l

    payload = {
        "collection_name": collection_name,
        "query": query,
        "model_configuration": {
            "name": model_name,
            "parameters": {
                "temperature": temperature,
                "max_tokens": max_tokens
            }
        },
        "prompt_template": (
            "ROLE\n"
            "You are senior Python developer specialized in building highly dependal
            "TASK: {question} \n"
            "CONTEXT: {context}\n"
        ),
        "retriever_name": retriever_name,
        "retriever_config": {
            "search_type": "similarity",

```

```

        "search_kwargs": {
            "k": k
        }
    },
    "stream": stream
}

try:
    response = requests.post(url, json=payload)
    response.raise_for_status()
    return response.json()
except requests.exceptions.RequestException as e:
    print("✗ Request failed:", e)
    return None

counter = 0
for filename, filedata in selected_files.items():
    counter+=1
    print(f"Runningg {counter} of {len(selected_files)}")

    try:
        instructions = f"""
        Perform a thorough code review of the code file below, focused on d
        - reliability (consistent correct behavior)
        - availability (system readiness)
        - safety (preventing harmful consequences)
        - confidentiality (avoiding unauthorized access)
        - integrity (ensuring data correctness)
        - maintainability (ease of fixing/evolving)

        Prefer concrete findings over vague style nits. If uncertain, do no

        FILE
        File name: ``{filename}``
        Code content: ``{filedata}``

        OUTPUT
        For each issue you find, output a JSON object with these fields:

        1. title: Short summary of the issue.
        2. location: the file where the issue occurs, ex:filename.py.
        3. code: (Read all the code and) Provide the FULL METHOD where the
        4. issue: Description of the dependability issue.
        5. impact: Possible consequences of the issue.
        7. label: bug, enhancement, docs, refactor, test ...

        EXAMPLE
        {{
            "title": "Handle missing config file",
            "location": "config_loader.py",
            "code": "`def update_data(id: int, new_value: float, storage: dic
                if not (0 <= new_value <= 100):\n
                    raise ValueError(f\"Invalid id value: {{new_valu
            # Update the storage\n
            if id in storage:\n
                storage[id].append(new_value)\n

```

```

        storage[id].append(new_value)\n
    else:\n
        storage[id] = [new_value]\n
    "issue": "No exception handling if config file is absent.",
    "impact": "Application will crash with FileNotFoundError.",
    "label": "bug"
    }}
    """"

result = get_rag_answer(query=instructions)
gpt_detections[filename] = result['answer']
print(result)

except Exception as err:
    print(err)
    continue

```

Write detections

```

!{sys.executable} -m pip install regex
import regex

pattern = regex.compile(r'\{(?:[^\{"]|"(?:\\.|["\\])*"|(?:R))*\}', regex.DOTALL)

def clean_json_for_parse(raw: str) -> str:
    # Remove any inner code fences like ```python ... ``` or stray ```
    cleaned = re.sub(r"```[a-zA-Z]*s*", "", raw) # opening fences with lang
    cleaned = cleaned.replace("```", "") # closing fences or stray
    # Remove trailing commas before } or ]
    cleaned = re.sub(r",\s*(\})", r"\1", cleaned)
    cleaned = re.sub(r",\s*(\})", r"\1", cleaned)
    return cleaned

all_issues = []

for name, content in gpt_detections.items():
    parsed = None
    try:
        parsed = json.loads(content)
    except json.JSONDecodeError as err:
        print(f"⚠️ FRIST ATTEMP JSON error in '{name}'")
        try:
            for m in pattern.finditer(content):
                parsed = json.loads(m.group(0))
        except:
            print(f"⚠️ SECOND ATTEMP JSON error in '{name}'")
            continue

    # Normalize to a list of issue dicts
    items = parsed if isinstance(parsed, list) else [parsed]

    for it in items:
        if isinstance(it, dict):
            # Attach the source name as 'message' if not present
            it.setdefault("message", name)

```

```

        it.setdefault(message, name)
        all_issues.append(it)
    else:
        # Unexpected item type inside array
        print(f"⚠️ Skipping non-dict item in '{name}': {type(it)}")

# ...then your CSV-write code...
fieldnames = ["title", "location", "code", "issue", "impact", "label"]

with open(output_path, "w", encoding="utf-8", newline="") as f:
    writer = csv.DictWriter(f, fieldnames=fieldnames)
    writer.writeheader()
    for issue in all_issues:
        writer.writerow({k: issue.get(k, "") for k in fieldnames})

print(f"✅ Exported {len(all_issues)} issues to {output_path}")

```

▼ Run Issues' Judgment

Run GEMINI to judge the detected issues

```

model = genai.GenerativeModel(model_gemini)
gpt_detection = pd.read_csv(output_path)

def parse_classification(text: str) -> str:
    """
    Extracts the "classification" value from the model's JSON response.
    Tries strict JSON parsing first, then falls back to regex.
    """
    # Normalize single-quotes to double-quotes for JSON
    try:
        obj = json.loads(text.replace("'", '"'))
        return obj.get("classification", "")
    except json.JSONDecodeError:
        pass

    # Fallback regex if JSON parsing fails
    m = re.search(r'"classification"\s*:\s*"([^"]+)"', text)
    if m:
        return m.group(1)
    m = re.search(r'"classification'\s*:\s*'([^']+)'", text)
    if m:
        return m.group(1)
    return ""

def classify_issue(row):
    prompt = (
        f"ROLE\n"
        f"You are a senior Python developer specialized in building highly dependable software.\n"
        f"TASK\n"
        f"Thinking step by step critically evaluate whether a reported issue truly is a bug or not."
    )

```

```

        f"Dependability issues are problems that significantly affect one or more o
        f"- reliability (consistent correct behavior)\n"
        f"- availability (system readiness)\n"
        f"- safety (preventing harmful consequences)\n"
        f"- confidentiality (avoiding unauthorized access)\n"
        f"- integrity (ensuring data correctness)\n"
        f"- maintainability (ease of fixing/evolving)\n"

        f"Reported Issue:\n"
        f"Title: {row['title']}\n"
        f"Location: {row['location']}\n"
        f"Code:\n{row['code']}\n"
        f"Issue Description: {row['issue']}\n"
        f"Impact: {row['impact']}"

        f"Based on the above, do you confirm this is a VALID and SIGNIFICANT dependi

        f"- Reject issues such as code style\n"
        f"- Reject TRIVIAL optimizations\n"
        f"- Reject changes with NEGLEGIBLE impact that do not compromise the system
        f"- If uncertain, do not hallucinate\n"

        f"OUTPUT"
        f"Please, write exactly one paragraph that summarizes your rationale.\n"
        f"At end you MUST output a JSON with the classification 'Confirm' or 'Rejec

        f"EXAMPLE"
        f"Paragraph justifying here ...\n\n"
        f"```json\n"
        "{\n"
        "  \"classification\": \"Confirm/Reject\"\n"
        "}\n"
        "```\n"
    )
    try:
        # Use the Gemini model to generate content
        response = model.generate_content(
            prompt,
            generation_config=genai.types.GenerationConfig(
                temperature=0.1,
                max_output_tokens=2048
            )
        )
        # Access the text from the response
        # Gemini's response structure is slightly different from OpenAI's
        print(response.text.strip())
        return response.text.strip()
    except Exception as e:
        # Log the error or handle it as needed
        print(f"Error during Gemini API call: {e}")
        return f"Error: {e}"

full_msgs = []
classifications = []

```

```
for index, row in gpt_detection.iterrows():
    gemini_message = classify_issue(row)
    full_msgs.append(gemini_message)
    classifications.append(parse_classification(gemini_message))

# 2) Add the confirmations as a new column
gpt_detection["gemini_justification_issue"] = full_msgs
gpt_detection["gemini_classification_issue"] = classifications
```

Write judgments

```
gpt_detection.to_csv(output_path, index=False, encoding='utf-8')
print(f"✅ Wrote {len(gpt_detection)} rows with full_message & classification to '{output_path}'")
```

✓ Run RAG to Generate FIX

```
gemini_classification = pd.read_csv(output_path, encoding="utf-8")

def parse_answer(result):
    obj = result

    if isinstance(obj, dict) and 'answer' in obj:
        obj = obj['answer']

    if isinstance(obj, str):
        m = re.search(r'\{.*\}', obj, flags=re.S)
        if m:
            try:
                obj = json.loads(m.group(0))
            except json.JSONDecodeError:
                return {'description': obj.strip(), 'fixed_code': None}
        else:
            return {'description': obj.strip(), 'fixed_code': None}

    if isinstance(obj, dict):
        return {
            'description': obj.get('description'),
            'fixed_code': obj.get('fixed_code')
        }

    return {'description': None, 'code': None}

def get_rag_answer(query: str, model_name: str = f"openai/{model_gpt_fix}", temperature: int = 5, stream: bool = False, url: str = "http://localhost:8000/retrievers/"):
    payload = {
        "collection_name": collection_name,
        "query": query,
        "model_configuration": {
            "name": model_name,

```



```

        "parameters": {
            "temperature": temperature,
            "max_tokens": max_tokens
        }
    },
    "prompt_template": (
        "ROLE\n"
        "You are senior Python developer specialized in building highly dependal
        "TASK: {question} \n"
        "CONTEXT: {context}\n"
    ),
    "retriever_name": retriever_name,
    "retriever_config": {
        "search_type": "similarity",
        "search_kwargs": {
            "k": k
        }
    },
    "stream": stream
}

try:
    response = requests.post(url, json=payload)
    response.raise_for_status()
    return response.json()
except requests.exceptions.RequestException as e:
    print("✗ Request failed:", e)
    return None

for idx, row in gemini_classification.iterrows():

    if row.get('gemini_classification_issue') == 'Reject':
        continue

    try:
        query = (
            f"Thinking critically step by step, evaluate the provided dependability issi
            f"Title: {row.get('title', '')}\n"
            f"Issue Description: {row.get('issue', '')}\n"
            f"Code: {row.get('code', '')}\n"

            f"OUTPUT\n"
            f"Output a json containing only two fields: 'description' and 'fixed_code'."

            f"EXAMPLE\n"
            f"```json\n"
            "{\n"
            "    \"description\": \"The soft delete implementation does not prevent acci
            "    \"fixed_code\": \"def get_queryset(self):\n        return super().get_que
            \"}\n"
            "```"
        )

    )

    result = get_rag_answer(query=query)
    parsed_result = parse_answer(result)

```

```

parsed_result = parse_answer(result)

code = parsed_result['fixed_code']
description = parsed_result['description']
#docs = result['docs']

gemini_classification.loc[idx, 'rag_fixed_code'] = code
gemini_classification.loc[idx, 'rag_description'] = description
#data.loc[idx, 'docs'] = docs
print(result)

except Exception as err:
    print("❌ Failed processing:", result)
    print(err)
    continue

```

```

gemini_classification.to_csv(output_path, index=False, encoding='utf-8')

print(f"✅ Wrote {len(gemini_classification)} rows with full_message & classification")

```

✓ Run Fixes' Judgment

Run GEMINI to judge the proposed solutions

```

rag_solution = pd.read_csv(output_path, encoding="utf-8")

def classify_issue(row):
    prompt = (
        f"ROLE\n"
        f"You are a senior Python developer specialized in building highly dependab"

        f"TASKS\n"
        f"Thinking step by step critically evaluate whether a reported issue was fi"

        f"Dependability issues are problems that significantly affect one or more o"
        f"- reliability (consistent correct behavior)\n"
        f"- availability (system readiness)\n"
        f"- safety (preventing harmful consequences)\n"
        f"- confidentiality (avoiding unauthorized access)\n"
        f"- integrity (ensuring data correctness)\n"
        f"- maintainability (ease of fixing/evolving)\n"

        f"Reported Issue:\n"

        f>Title: {row['title']}\n"
        f"Location: {row['location']}\n"
        f"Code:\n{row['code']}\n"
        f"Issue Description: {row['issue']}\n"
        f"Impact: {row['impact']}"
        f"Solution: {row['rag_fixed_code']}"

        f"Based on the above, do you confirm this issue was COMPLETELY fixed?\n"
    )

```

```

        f"If uncertain, do not hallucinate\n"

        f"OUTPUT\n"
        f"Please, write exactly one paragraph that summarizes your rationale.\n"
        f"At end you MUST output a JSON with the classification 'Confirm' or 'Reject'"

        f"EXAMPLE\n\n"

        f"Paragraph justifying here ...\n\n"
        f"```json\n"
        f"{\n"
        f"  \"classification\": \"Confirm/Reject\"\n"
        f"}\n"
        f"```\n"
    )

    try:
        response = model.generate_content(
            prompt,
            generation_config=genai.types.GenerationConfig(
                temperature=0,
                max_output_tokens=2048
            ),
            request_options={'timeout': 10} # Set a timeout of 60 seconds
        )
        # Access the text from the response
        # Gemini's response structure is slightly different from OpenAI's
        print(response.text.strip())
        return response.text.strip()
    except Exception as e:
        # Log the error or handle it as needed
        print(f"Error during Gemini API call: {e}")
        return f"Error: {e}"

full_msgs = []
classifications = []

for index, row in rag_solution.iterrows():
    if row.get('gemini_classification_issue') == 'Reject':
        continue

    gemini_message = classify_issue(row)
    classifications.append(parse_classification(gemini_message))

    rag_solution.loc[index, "gemini_justification_fix"] = gemini_message
    rag_solution.loc[index, "gemini_classification_fix"] = parse_classification(gemini_message)

rag_solution.to_csv(output_path, index=False, encoding='utf-8')
print(f"✅ Wrote {len(rag_solution)} rows to '{output_path}'")

```

✎ Final Result

```
filtered_df = rag_solution[(rag_solution["gemini_classification_issue"] == "Confirm"  
display(filtered_df)  
filtered_df.to_csv(output_path_filtered, index=False, encoding='utf-8')
```