Anna Carolina Fonseca Buarque de Verçosa

Chatbot de suporte acadêmico para universitários

PROJETO FINAL

CENTRO TÉCNICO CIENTÍFICO - CTC
DEPARTAMENTO DE DE INFORMÁTICA

Curso de Graduação em Engenharia da Computação

Rio de Janeiro
Junho de 2024

Pontifícia Universidade Católica
do Rio de Janeiro

Anna Carolina Fonseca Buarque de Verçosa

Chatbot de suporte acadêmico para universitários

Relatório de Projeto Final

Relatório de Projeto Final, apresentado ao programa de Engenharia da Computaçãoda PUC-Rio como requisito parcial para a obtenção do titulo de Engenheiro de Computação.

Orientador: Prof. Augusto Cesar Espíndola Baffa

Rio de Janeiro
Junho de 2024

Resumo

Fonseca Buarque de Verçosa, Anna Carolina; Espíndola Baffa, Augusto Cesar. **Chatbot de suporte acadêmico para universitários**. Rio de Janeiro, 2024. 44p. Projeto de Graduação – Departamento de de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

O objetivo deste projeto é desenvolver um chatbot para auxiliar estudantes universitários, utilizando o framework open-source Rasa para seu desenvolvimento. O chatbot deve ser capaz de responder à perguntas frequentes sobre a universidade, como cardápio do bandejão, informações sobre disciplinas, datas acadêmicas e grade de horário do aluno.

Palavras-chave

chatbot, rasa, sistemas conversacionais

Sumário

1
Introdução

O avanço da tecnologia promoveu uma transformação na forma como as informações são acessadas e compartilhadas. Nesse contexto, temos os *chatbots*, agentes conversacionais que utilizam da linguagem natural para interagir com os usuários, através de texto ou voz. (Hussain et al., 2019)

Além de serem usados para entretenimento, simulando interações humanas com outras pessoas, os *chatbots* também conseguem atuar em uma ampla gama de setores, incluindo atendimento ao cliente, suporte técnico, saúde e educação. A sua relevância vem da capacidade de proporcionar uma experiência de acesso à informação de forma a engajar mais o usuário, ao mesmo tempo que reduz os custos operacionais das empresas, sendo mais econômico do que um serviço de suporte prestado por humanos. (Caldarini et al., 2022) A sua crescente popularidade pode ser atribuída também à facilidade de integração com aplicativos e redes sociais como o Telegram, Discord, Facebook e Twitter, facilitando o seu uso sem que o usuário precise instalar outra aplicação em seu dispositivo além das que costuma utilizar. (Pérez-Soler et al., 2021)

No ambiente empresarial e acadêmico, esses agentes conversacionais podem auxiliar o acesso à informações atualizadas de forma rápida e eficiente, visto que muitas vezes esses dados estão dispersos em locais diferentes, como *websites* ou e-mails. O conhecimento e dúvidas de funcionários de uma empresa ficam registrados, de forma que recuperar essa memória através de um chatbot pode aumentar a produtividade e resolver problemas já enfrentados. Novos colaboradores podem ser mais eficientes se tivessem a possibilidade de acessar esse histórico mais antigo.

Com isso, o objetivo deste projeto é a criação de um *chatbot* voltado para o auxílio de alunos dentro do contexto da universidade que seja acessível em termos de custo, eficaz nas suas interações e capaz de ser implantada de forma ágil, avaliando o desempenho do *framework* Rasa utilizado em seu desenvolvimento.

2
Situação Atual

Em 1950, iniciou-se as primeiras discussões sobre a possibilidade de um programa de computador conseguir se comunicar com um ser humano sem que a mesma descobrisse. Alan Turing propôs esse conceito ao questionar se seria realmente possível uma máquina pensar, de forma que não seria possível destinguir ela de um ser humano. Esse poderia ser considerado o marco inicial para a área de sistemas conversacionais. (Caldarini et al., 2022)

Desde então, com o avanço das tecnologias no campo da computação e do processamento de linguagem natural, diversas abordagens foram utilizadas no desenvolvimento de *chatbots*. Existem duas principais áreas de implementação: a utilização de técnicas de correspondência de padrões (*pattern matching*) e a aplicação de algoritmos de aprendizado de máquina (*machine learning*).

No caso de *pattern matching*, o *chatbot* procura estabelecer uma relação entre a entrada fornecida pelo usuário e as respostas preestabelecidas em suas regras, as quais foram definidas pelo desenvolvedor. Uma linguagem muito popular para construção de *chatbots* que utiliza essa técnica é o AIML(*Artificial Intelligence Markup Language*).[1] Uma definição de regra de forma a responder uma pergunta do usuário seria:

```
<category>
  <pattern>EU QUERO SABER *</pattern>
  <template>Você quer saber <star/>, certo? Pode fornecer mais detalhes
  ↪  para que eu possa te ajudar?</template>
</category>
```

No entanto, tal abordagem tende a resultar em interações mais rígidas, uma vez que apresenta desafios na adaptação e no tratamento de erros das sentenças dos usuários.

Já no caso das implementações que utilizam *machine learning*, utiliza-se o Processamento de Linguagem Natural(Natural Language Processing, ou NLP) com a finalidade de extrair a intenção do usuário contida em sua sentença, não sendo necessário uma resposta correspondente a cada possível entrada do usuário em uma lista de regras. (Adamopoulou e Moussiades, 2020) Alguns

[1] http://www.aiml.foundation/

frameworks como o Dialogflow[2] do Google e o Rasa[3] utilizam dessa técnica para facilitar a implementação de um *chatbot*.

Para compreender o atual estado da arte e o potencial dessas aplicações, é essencial analisar a história da evolução dos chatbots. Portanto serão examinadas os principais marcos significativos ao longo do tempo e como elas construíram o cenário atual dos *chatbots*.

2.1 ELIZA

O *chatbot* ELIZA foi o primeiro a ser desenvolvido em 1966 por Joseph Weizenbaum, em uma linguagem chamada MAD-SLIP executada em um computador IBM 7094 no MIT. Durante a conversa com ELIZA, o usuário digitava na máquina de escrever uma frase em linguagem natural enquanto ELIZA respondia o seu questionamento utilizando o mesmo instrumento. (Weizenbaum, 1966)

ELIZA funcionava com um auxílio de um roteiro, como podemos observar na figura 2.1, e correspondência de padrões. Quando certas palavras eram reconhecidas na entrada do usuário e o contexto em volta delas por outras palavras auxiliares, a resposta apropriada de acordo com o script era enviada para o usuário, mantendo a ilusão de uma conversa com um ser humano.

O *chatbot* funcionava como uma conversa com um terapeuta, visto que, dessa forma, as duas pessoas não precisavam ter noção sobre o mundo externo, já que perguntas feitas por um terapeuta teriam a intenção de extrair mais informações do seu paciente na conversa. Esse formato é crucial pois o roteiro que ELIZA segue é limitado em relação a coisas que consideramos comuns no dia a dia. (ELIZAGEN, 2023)

Para contornar essas limitações, Weizenbaum sugeriu a adição da capacidade de ELIZA construir um modelo ao conversar com os usuários, de forma que essa base de conhecimento poderia ser usada para conversas futuras.

2.2 Parry

Desenvolvido em 1972 por Kenneth Colby, o *chatbot* Parry simulava a conversa com um paciente com esquizofrenia, com o objetivo de imitar os sintomas da doença, e assim melhorar a compreensão e tratamento dessa condição. Foi um dos primeiros *chatbots* a demonstrar a capacidade de simular uma personalidade.

[2]https://cloud.google.com/dialogflow
[3]https://rasa.com/

```
3        PRINT,T0109,2531,,TAPE,,102                          T0109  2531

(HOW DO YOU DO.  I AM THE DOCTOR.  PLEASE SIT DOWN AT THE TYPEWRITER      000010
AND TELL ME YOUR PROBLEM.)                                                000020
((IF 3 ((0 IF 0) (DO YOU THINK ITS LIKELY THAT 3) (DO YOU WISH THAT 3)    000030
(WHAT DO YOU THINK ABOUT 3) (REALLY, 2 3)))                               000040
(MEMORY MY (0 YOUR 0 = LETS DISCUSS FURTHER WHY YOUR 3)                   000050
(0 YOUR 0 = EARLIER YOU SAID YOUR 3)                                      000060
(0 YOUR 0 = BUT YOUR 3)                                                   000070
(0 YOUR 0 = DOES THAT HAVE ANYTHING TO DO WITH THE FACT THAT YOUR 3))     000080
(NONE ((0) (I AM NOT SURE I UNDERSTAND YOU FULLY)                         000090
(PLEASE GO ON)                                                            000100
(WHAT DOES THAT SUGGEST TO YOU)                                           000110
(DO YOU FEEL STRONGLY ABOUT DISCUSSING SUCH THINGS)))                     000120
(PERHAPS ((0) (YOU DON'T SEEM QUITE CERTAIN)                              000130
(WHY THE UNCERTAIN TONE)                                                  000140
(CAN'T YOU BE MORE POSITIVE)                                              000150
(YOU AREN'T SURE)))                                                       000180
(MAYBE (=PERHAPS))                                                        000190
(DON'T YOU KNOW)                                                          000200
(AM = ARE ((0 ARE YOU 0) (DO YOU BELIEVE YOU ARE 4)                       000210
(WOULD YOU WANT TO BE 4) (YOU WISH I WOULD TELL YOU YOU ARE 4)            000220
(WHAT WOULD IT MEAN IF YOU WERE 4))                                       000230
((0) (WHY DO YOU SAY 'AM') (I DON'T UNDERSTAND THAT)))                    000240
(ARE = AM ((0 AM I 0)                                                     000250
(WHY ARE YOU INTERESTED IN WHETHER I AM 4 OR NOT)                         000260
(WOULD YOU PREFER IF I WEREN'T 4) (PERHAPS I AM 4 IN YOUR                 000270
FANTASIES) (DO YOU SOMETIMES THINK I AM 4))                               000280
((0 AM 0) (DID YOU THINK THEY MIGHT NOT BE 3)                             000290
(WOULD YOU LIKE IT IF THEY WERE NOT 3) (WHAT IF THEY WERE NOT 3)          000300
(POSSIBLY THEY ARE 3)) )                                                  000310
(YOUR = MY ((0 MY 0) (WHY ARE YOU CONCERNED OVER MY 3)                    000320
(WHAT ABOUT YOUR OWN 3) (ARE YOU WORRIED ABOUT SOMEONE ELSES 3)           000330
(REALLY, MY 3)))                                                          000340
(WAS = WERE)                                                              000350
(WERE = WAS)                                                              000360
(ME = YOU)                                                                000370
(YOU'RE = I'M)                                                            000380
(I'M = YOU'RE)                                                            000390
(MYSELF = YOURSELF)                                                       000400
(YOURSELF = MYSELF)                                                       000410
(MOTHER DLIST(/NOUN FAMILY))                                              000420
(FATHER DLIST(/NOUN FAMILY))                                              000430
(SISTER DLIST(/FAMILY))                                                   000440
```

Figura 2.1: Trecho do script original utilizado por ELIZA.(ELIZAGEN, 2023)

O algoritmo foi escrito em MLISP e executado em um minicomputador DEC PDP-6 (que posteriormente foi substituído por um modelo mais novo, PDP-10) do Projeto de Inteligência Aritificial de Stanford. Ao receber uma entrada em linguagem natural, o programa identificava as intenções do usuário, procurando palavras que poderiam ser utilizadas para entender o seu significado, podendo ser classificado como malevolente, benevolente ou neutro. Dentro desses contextos, existe outras variáveis que também são responsáveis pelo resultado final, levando a uma resposta com uma emoção diferente. (Colby et al., 1972)

Contudo, é considerado um *chatbot* com baixa capacidade de entendimento de linguagem e expressividade de emoções, além de baixa velocidade ao responder o usuário. Também apresenta o mesmo problema de ELIZA em relação a uma base de conhecimento, uma vez que ambos não conseguem aprender com as interações com os usuários. (Adamopoulou e Moussiades, 2020)

2.3 Jabberwacky

Tendo seu nome similar ao poema nonsense Jabberwocky que aparece no livro Alice no País dos Espelhos, obra do escritor britânico Lewis Carroll, a resposta dada pelo *chatbot* Jabberwacky muitas vezes não faz sentido com o

que foi perguntado. Esse comportamento pode ser ser observado na figura 2.2, sendo um sistema mais voltado para o entretenimento com o usuário.

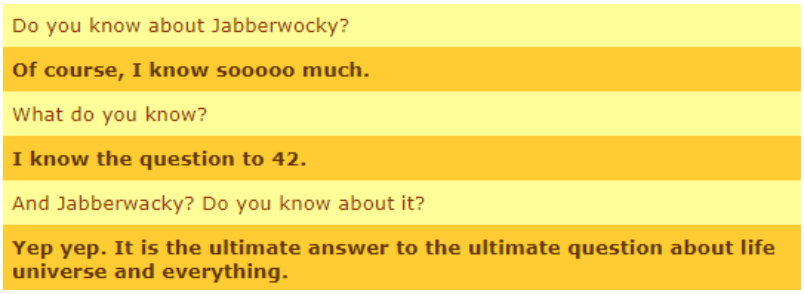

Figura 2.2: Conversa com o chatbot Jabberwacky.(JABBERWACKY, 2023)

Desenvolvido por Rollo Carpenter em 1988, e disponibilizado *online* em 1996, utiliza da correspondência de padrões para identificar uma resposta apropriada com base em planilhas que contém informações de interações passadas com outros usuários. Tem a vantagem de conseguir contruir uma base de conhecimento em comparação com os chatbots anteriormente mencionados, identificando até a linguagem do usuário e respondendo com a mesma. (Adamopoulou e Moussiades, 2020)

Ele evoluiu para o Cleverbot em 2006, e desde 2007 o seu algoritmo é utilizado para outros *chatbots* voltados para o entretenimento, como Eviebot e Boibot. Ambos possuem o adicional de utilizarem síntese de fala realista, também conhecido como *text to speech*, e avatares humanos para comunicarem com os usuários, como pode ser observado na figura 2.3

2.4 ChatterBot

O termo *chatbot* veio da palavra ChatterBot, que é o resultado da combinação das palavras *chatter* (que significa conversa) e *bot* (uma abreviação de robô, em inglês), cunhado por Michael Maudlin na década de 90. (Luo et al., 2022) Esse *chatbot* interagia em um jogo *multiplayer online* chamado TINYMUD de 1989, que permitia conversa entre os jogadores, descrição dos ambientes via texto e a habilidade dos jogadores de criarem subáreas dentro do jogo. (Mauldin, 1994)

O ChatterBot foi implementado em C e sua habilidade de conversação seguia regras de IF-THEN-ELSE, utilizando correspondência de padrões. Também tem como base os *chatbots* ELIZA e Parry, utilizando de estratégias para simular uma interação humana, como perguntar de maneira que o foco da conversa seja no usuário ou admitir quando não sabe a resposta de uma questão.

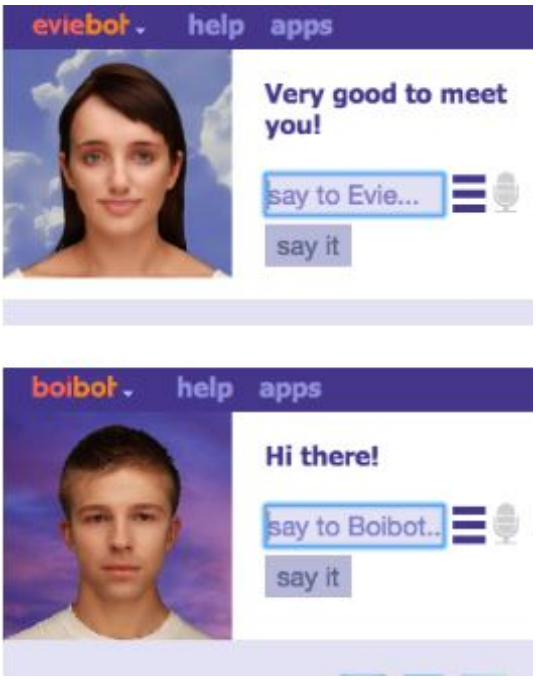

Figura 2.3: Interface de Eviebot e Boibot.(Tero et al., 2016)

A ilusão é mantida partindo do pressuposto que os usuários do jogo não sabem que estão interagindo com um *chatbot*, a não ser que o mesmo cometa uma falha grave. Um dos truques novos que Maudlin adicionou no ChatterBot é a simulação da digitação de um humano, incluindo pausas entre cada caracter digitado além de erros na digitação que poderiam ser plausíveis em um teclado QWERTY.

2.5 A.L.I.C.E.

O chatbot A.L.I.C.E., cuja sigla corresponde a *Artificial Linguistic Internet Computer Entity*, foi desenvolvido com a linguagem AIML em 1995 por Richard Wallace. A.L.I.C.E. teve inspiração em ELIZA, porém é superior em questão de categorias de conhecimento e também pela possibilidade de coletar dados em linguagem natural pela web. Dessa forma, o conjunto de perguntas e respostas na qual o *chatbot* irá se basear será o conjunto de assuntos mais prováveis que um usuário pode perguntar, evitando que regras pouco utilizadas sejam adicionadas. (Wallace, 2009)

O AIML é baseado no XML(*Extensible Markup Language*), e sua base de conhecimento é composto de categorias, cada uma com uma questão de entrada(*pattern*) e uma resposta de saída(*template*). Dessa forma ele procura a correspondência de frase mais longa entre a entrada do usuário e o *pattern* definido em suas regras.

Ele utiliza três tipos de categorias: *atomic*(atômica), *default*(padrão) e *recursive*(recursiva). As categorias atômicas consistem de frases mais básicas, que não utilizam símbolos como _e *, funcionando como coringas quando for feito a correspondência da pergunta do usuário com a regra implementada. As categorias do tipo padrão utilizam esses símbolos coringas, então se o usuário escrever algo que não existe exatamente igual na lista de regras, o chatbot procurará uma regra mais próxima com a presença desses símbolos. Por fim, as categorias recursivas, identificadas pela tag srai, utilizam da recursão para vários objetivos, como transformar regras gramaticais em outras mais simples, dividir e conquistar dividindo uma entrada em mais de uma para no final unir a resposta ou lidar com sinônimos e erros gramaticais.

Na figura 2.4, é possível observar um trecho de código originalmente escrito por Richard Wallace em seu artigo “The Anatomy of A.L.I.C.E."para correção gramatical.

```
<category>
<pattern>YOUR A *</pattern>
<template>I think you mean “you’re” or “you are” not “your.”
<srai>YOU ARE A <star/></srai>
</template>
</category>
```

Figura 2.4: Trecho de código AIML. (Wallace, 2009)

2.6 ChatGPT

O ChatGPT(*Chat Generative Pre-trained Transformer*) foi lançado em 2022 pela OpenAI, utilizando o modelo GPT-3, um modelo grande de linguagem(*Large Language Models*, ou LLM, em inglês). LLMs utilizam *machine learning* e são treinados em uma vasta quantidade de assuntos, sendo extremamente adaptáveis em suas conversas com os usuários, visto que possui mais de 175 bilhões de parâmetros que permitem uma geração de texto muito eficaz. (MarkTechPost, 2023)

O modelo GPT tem como base a arquitetura *Transformer*, uma arquitetura de redes neurais proposta por uma equipe de pesquisadores do Google e da Universidade de Toronto em 2017 no artigo “Attention Is All You Need". Tornou-se muito popular para lidar com tarefas da área de NLP e teve seu uso extendido também para tarefas de processamento de imagem.

A arquitetura *Transformer* tem como base a arquitetura *encoder-decoder*, muito utilizada na área de tradução de texto por máquina. O *encoder* codifica a

entrada para uma representação numérica, que então é passada para o *decoder* gerar a sequência de saída. No caso do modelo GPT, ele é considerado do tipo *decoder-only*, e dado uma frase ele procura autocompletar a sequência, considerando qual a próxima palavra seria a mais provável de ocorrer no texto. (Tunstall, 2022)

O diálogo do *chatbot* é otimizado pelo uso do aprendizado por reforço com feedback humano(*Reinforcement Lerning with Human Feedback*, ou RLHF, em inglês), na qual utiliza as preferências dos usuários para ajustar o modelo para algo mais correto e natural. (OpenAI, 2023)

2.7 BARD

Lançado em março de 2023 pela Google, utiliza o modelo de linguagem LaMDA(*Language Model for Dialogue Applications*), também lançado pela empresa no início do ano. Esse modelo também é baseado na arquitetura *Transformer*, assim como os modelos da família GPT, e foi treinado com foco em diálogos com o objetivo de tornar a conversação mais fliuída e natural para o usuário.. (Google, 2023)

No final de 2023, o BARD ganhou uma atualização para utilizar o mais novo modelo de linguagem desenvolvido pela Google, chamado Gemini. A sua versão Pro demonstrou desempenho melhor do que o GPT-3.5 em em testes que avaliam habilidades em resolver problemas matemáticos e conhecimentos gerais. No início de 2024, a Google rebatizou BARD para Gemini, de forma a refletir o novo nome do modelo utilizado. (Google, 2024)

2.8 LLaMA

Desenvolvido pela Meta AI, o Llama (*Large Language Model Meta AI*) foi disponibilizado para o público geral em Fevereiro de 2023, tendo suas versões seguintes, Llama 2 e Llama 3, lançadas em Julho de 2023 e Abril de 2024, respectivamente. O propósito do desenvolvimento foi a criação de um modelo menor porém mais poderoso, utilizando dados de treinamento de alta qualidade, e que exigisse menos recursos.

Sua versão mais recente possui modelos treinados em 15 trilhões de tokens originados de textos em domínio público, cobrindo mais de 30 linguagens. O Llama 3 também é utilizado no assistente virtual Meta AI, que pode ser utilizado no Facebook, WhatsApp e Instagram, gerando não apenas texto mas como imagens. (Meta, 2024)

2.9 Alpaca

Divulgado em março de 2023, o Alpaca foi desenvolvido por pesquisadores da Universidade de Stanford, com base no modelo LLaMA. Seu funcionamento é similar ao modelo text-davinci-003 da OpenAI, com a vantagem de ser menor e mais barato em sua utilização. O treinamento inicial teve um custo de menos de 100 dólares, demorando apenas 3 horas para finalizar. (Taori et al., 2024)

Diferente de outros modelos, o seu foco é na pesquisa acadêmica, tendo seu uso comercial proibido. Com isso, as instruções de uso para ajuste fino do LLaMA assim como os dados de treinamento utilizado são disponibilizados no GitHub no repositório Stanford Alpaca.[4]

[4]https://github.com/tatsu-lab/stanford$_a lpaca$

3
Objetivos do Trabalho

O objetivo do projeto é desenvolver um *chatbot* ajudar os estudantes universitários com suas dúvidas. O *chatbot* deve ser capaz de responder perguntas frequentes sobre a universidade, como cardápio do bandejão, datas acadêmicas, consulta de disciplinas e salas no microhorário, informações sobre ementa e cadastro de grade horária no *chatbot*, permitindo que os alunos consultem as disciplinas posteriormente. Para isso, serão utilizados dados da Pontifícia Universidade Católica do Rio de Janeiro como base de conhecimento e pesquisa para o *chatbot*.

Será utilizado o *framework open-source* Rasa, escolhido após uma pesquisa dos *frameworks* mais utilizados no mercado, com o objetivo de achar a melhor opção em relação a custo, facilidade de desenvolvimento, boa documentação e suporte na construção dos diálogos do *chatbot* com foco na linguagem portuguesa. O funcionamento do Rasa é detalhado na seção 4.3.

Além da possibilidade de processar a entrada do usuário via expressões regulares ou correspondência de padrões, também é possível o uso de modelos NLP pré-treinados para diferentes linguagens. Por não oferecer uma interface gráfica para configuração e desenvolvimento do chatbot, é utilizado a linguagem de programação Python junto a outros arquivos de configuração escritas em YAML. (Pérez-Soler et al., 2021)

A arquitetura do Rasa permite a utilização do chatbot criado em várias plataformas, sendo de fácil configuração para adaptar seu uso em outros ambientes. Para o trabalho, foi focado primeiramente o seu uso no Telegram, um aplicativo de mensagens instantâneas disponível para *mobile* e *desktop*, porém também foi possível testar o seu funcionamento dentro do Discord, uma plataforma de comunicação muito voltada para a comunidade de jogos mas que seu uso vem sido disseminado para outros grupos.

4
Atividades Realizadas

4.1
Estudos preliminares

Antes de iniciar o trabalho, a autora teve contato com criação e utilização de *bots* no Telegram com auxílio do Python porém tinha um conhecimento superficial sobre a área de sistemas conversacionais e processamento de linguagem natural. Em decorrência disso, foi cursado disciplinas que cobriam esses assuntos em sua ementa para um entendimento mais profundo, junto com auxílio de material compartilhado pelo orientador.

Ao longo do projeto, também foi estudado técnicas utilizadas nos mais vários *chatbots*, como regras de correspondência de padrões, redes neurais e aprendizado de máquina, assim como sobre a história dos *chatbots*, desde os primeiros desenvolvidos na década de 60 até os mais modernos. Esses estudos preliminares foram fundamentais para a compreensão do tema e para o desenvolvimento do trabalho.

4.2
Estudos conceituais

Considerando o objetivo do protótipo a ser desenvolvido, inicialmente foi feita a definição das categorias mais adequadas para o chatbot e técnicas a serem utilizadas em seu desenvolvimento. Algumas categorias que um *chatbot* pode ser categorizado são: domínio de conhecimento, objetivos, modo de interação e abordagem de desenvolvimento. (Hussain et al., 2019)

Em relação ao domínio de conhecimento, optou-se pela construção de um *chatbot* de domínio fechado. Chatbots de domínio fechado, ou *closed domain*, operam com uma base de conhecimento específica, tendo um custo menor de desenvolvimento. Em contrapartida, não são eficazes em lidar com perguntas que fogem do seu escopo, o que pode frustar os usuários, principalmente aqueles que estão acostumados a utilizar outros chatbots do tipo de domínio aberto (*open domain*), como o ChatGPT e o Gemini (vide seção 2.7).

No que diz respeito à categoria de objetivo, um *chatbot* pode ser orientado para tarefas ou não. Esta classificação possui uma proximidade com o domínio

de conhecimento, já que *chatbots* orientados para tarefas costumam ser também de domínio fechado. Eles são projetados para diálogos mais curtos e para a resolução de problemas do usuário. Em contrapartida, *chatbots* não orientados para tarefas costumam também ser de domínio aberto.

Em questão dos tipos de interação que o usuário pode fazer com o *chatbot*, podem ser através de texto, voz ou imagem. *Chatbots* mais recentes e avançados conseguem interagir com o usuário em todas as frentes descritas, oferecendo uma experiência mais natural. No entanto, devido ao custo e tempo de desenvolvimento, o protótipo focará apenas na comunicação via texto, tanto para a interpretação das entradas do usuário quanto para a geração de respostas.

Por fim, existe a categorização de acordo com as técnicas de processamento de linguagem e geração de resposta que o *chatbot* utiliza. Antes de aprofundar nas diferentes técnicas, é importante destacar as áreas de *Natural Language Processing* (NLP), *Natural Language Understanding* (NLU) e *Natural Language generation* (NLG) dentro desse contexto. O estudo do entendimento de linguagem natural por computadores é multidisciplinar, com atuação de áreas como inteligência artificial e linguística. Sentenças que podem ter algum tipo de ambiguidade, como “Eu vi a mulher com um binóculo", ou que requerem algum contexto cultural para o entendimento da expressão, como é o caso “Ela está com a faca e o queijo na mão", são problemas difíceis para um computador, já que não requerem apenas o conhecimento de palavras separadas mas a compreensão dos papéis e múltiplos significados que elas podem ter.

K.R. Chowdhary define em seu livro *Fundamentals of Artificial Intelligence* a seguinte árvore de componentes de NLP, representado na figura 4.1. Para uma análise da linguagem, seria necessário o entendimento da estrutura de discurso e diálogo junto com a análise da sentença, sendo essa quebrada nos ramos de análise sintática e semântica.

A análise sintática é responsável pelo identificação dos pontos gramaticais mais importantes da sentença, como sujeito, objeto, verbos e adjetivos, enquanto a análise semântica é encarregada de entender o que realmente a frase significa. A partir dos dois, a estrutura de diálogo e discurso se torna possível. Apesar de não ser necessário em todas as aplicações, essa análise é complexa porém necessária para o entendimento da ideia que o texto quer passar por um todo, e não apenas o sentido de frases soltas.

Considerando o objetivo do NLU e NLP, ao longos das décadas foram desenvolvidos várias técnicas para facilitar a atuação dos computadores nessa área. Conforme abordado anteriormente, os primeiros sistemas conversacionais

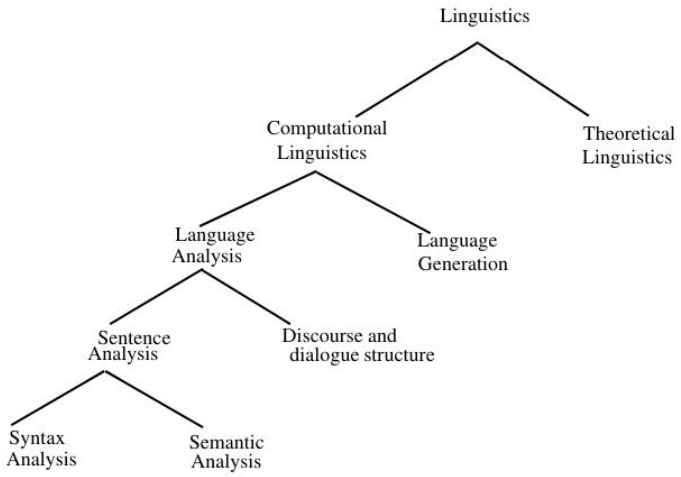

Figura 4.1: Componentes de Natural Language Processing (Chowdhary, 2020)

utilizavam uma técnica conhecida como *pattern matching*, com o AIML tendo o maior destaque, tanto pelo seu próprio uso como também por sua influência em outras linguagens. O principal ponto negativo é que, por precisar corresponder o padrão de entrada do usuário com algum padrão já definido em sua base de conhecimento pelo desenvolvedor, para um funcionamento mais natural é necessário a definição de todas essas prováveis perguntas. Dependendo do escopo que o *chatbot* pretende atuar, essa técnica pode ser inviável.

Dentro do contexto de técnicas de *machine learning*, existem algumas técnicas utilizadas para o préprocessamento do texto antes da aplicação de um algoritmo de *machine learning* para melhorar a precisão dos resultados. Algumas dessas técnicas são a tokenização, segmentação de palavras, *part of speech tagging* e *parsing*. (Sun et al., 2017) A diferença de cada técnica é mostrada na figura 4.2.

Linguagens como o português e o inglês se beneficiam da técnica de *tokenization* visto que apresentam espaços para a separação de palavras, o que não é o caso de outras linguagens como o japonês, onde as palavras não possuem esse tipo de divisão. Para esses casos pode ser utilizada a segmentação de palavras, identificando quais ideogramas formam cada palavra para assim ser feita a separação necessária.

Como mencionado na seção 2, a arquitura de rede neural conhecida como *Transformer* prevalece na área de NLP desde que foi divulgada em 2017 devido ao seus resultados superiores a outras arquiteturas que eram baseadas em rede neural recorrente(RNN), além de serem extremamente versáteis, podendo ser usado não apenas em textos mas com imagens também. (Tunstall, 2022) Na figura 4.3, pode ser observado o esquema da arquitetura com mais detalhes,

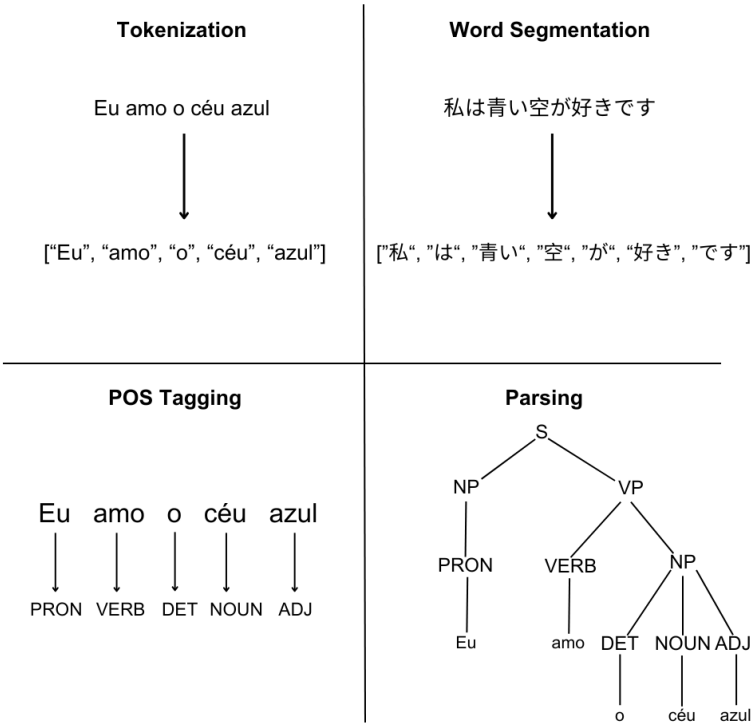

Figura 4.2: Técnicas de préprocessamento de texto

com o bloco da esquerda sendo a sua parte chamada *encoder*, enquanto o da direita é o *decoder*. Apesar que em sua concepção o intuito era atuar em tarefas de tradução, sua arquitetura já foi adaptada de forma que existem modelos que utilizam apenas o componente *encoder* ou *decoder*.

Modelos baseados apenas no bloco *encoder* são voltados para tarefas de classificação de texto e NER(*named entity recognition*), onde o objetivo é identificar em um texto quais são as entidades considerando categorias prédefinidas, como país, nome ou data. Já modelos baseados unicamente no bloco *decoder* trabalham de forma a gerar textos palavra por palavra, de forma que a próxima palavra na sequência é aquela que é mais provável de aparecer dado o contexto da prórpia sentença. Os modelos BERT e GPT são, respectivamente, os principais representantes de cada tipo.

Apesar de todos os avanços, ainda existem desafios que modelos que utilizam a arquitetura *Transformer* precisam enfrentar, como em relação a limitação da linguagem, visto que muita das pesquisas e modelos prétreinados existentes são voltados para a língua inglesa, assim como o fato de que eles funcionam como uma caixa preta. É difícil entender porque um modelo fez certa predição, já que seu processo de transformação de entrada para saída não é transparente. (Tunstall, 2022)

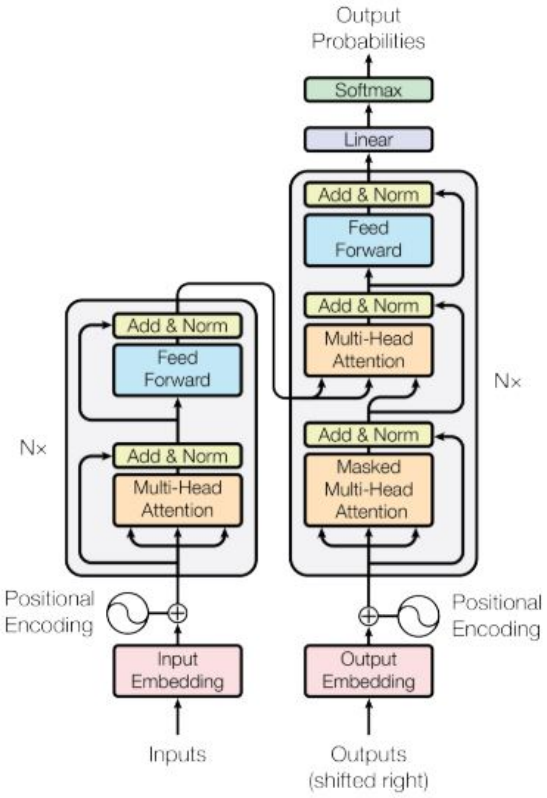

Figura 4.3: Arquitetura do transformer (Vaswani et al., 2017)

4.3 Estudos da tecnologia

O Rasa é um *framework open-source* voltado para o desenvolvimento de *chatbots*, oferecendo suporte para NLU, gerenciamento de diálogos e integração à diversas plataformas, como o Telegram, Facebook e Slack. Esse *framework* já foi utilizado em empresas ao redor do mundo, sendo uma delas o PicPay, um aplicativo brasileiro de pagamentos, que buscava uma solução escalável e com rapidez na implantação. (RASA, 2023) O assistente virtual foi implementado dentro do próprio aplicativo com o objetivo de informar famílias afetadas pela pandemia sobre a disponibilidade de auxílio e ajudar no resgate do dinheiro na carteira digital, como pode ser observado na figura 4.4.

Esse *framework* utiliza o conceito de intenções e entidades para compreender as frases enviadas pelo usuário e conseguir responder de forma apropriada. A intenção é o objetivo do usuário ao enviar uma mensagem para o chatbot, enquanto uma entidade é uma palavra ou frase que é relevante dentro da intenção identificada. Considerando como exemplo a frase “Quero consultar a ementa da matéria Inteligência Artificial”, a intenção identificada poderia ser “Consultar ementa” enquanto a entidade seria “Inteligência Artificial”. Para

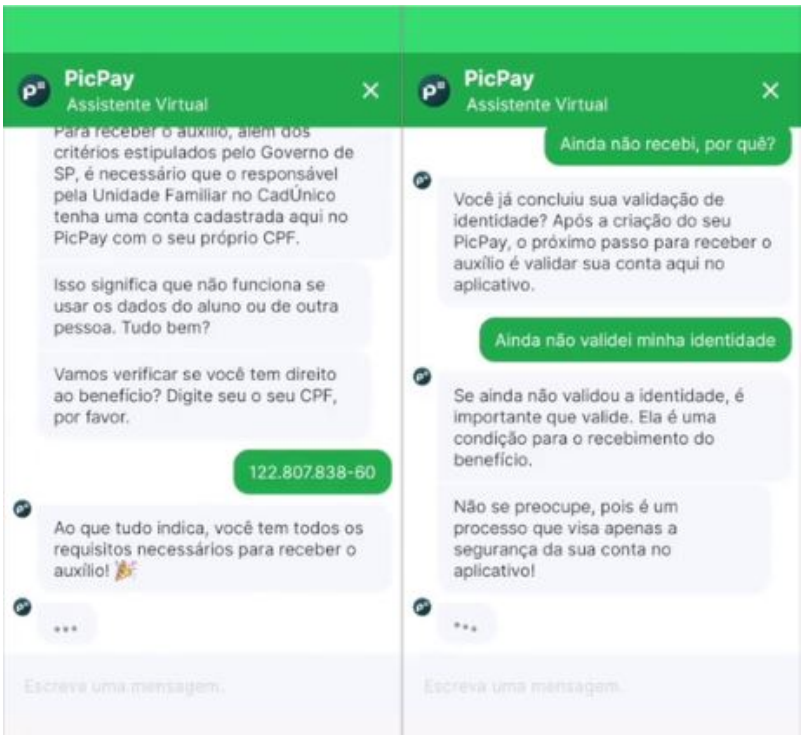

Figura 4.4: Chatbot do PicPay utilizando Rasa

que isso ocorra, é necessário fornecer dados de treinamento com as intenções e entidades mapeadas para que o modelo consiga aprender e assim conseguir generalizar para outras frases similares.

Para entendimento do texto, o Rasa utiliza o conceito de *pipeline*. Esse *pipeline* é uma sequência de etapas de processamento de texto para auxiliar no treinamento do modelo e interpretação da mensagem enviada pelo usuário. Cada componente do *pipeline* possui configurações que podem ser alteradas, assim como podem ser retirados ou adicionados componentes extras. (Warmerdam, 2021)

Considerando o *pipeline inicial* sugerido pelo Rasa na figura 4.5, o processo é iniciado pela etapa de tokenização, com o componente *WhitespaceTokenizer* separarando a frase do usuário utilizando o espaço como divisor. O resultado dessa etapa é um vetor de palavras que é então passado passado para a próxima etapa, conhecida como featurizer.

Essa etapa de featurização tem o intuito de criar uma representação numérica de forma a destacar as principais características do texto para o aprendizado do modelo e interpretação do texto de entrada. No pipeline básico é utilizado três tipos diferentes de componentes para criar *features* mais robustas. O componente *RegexFeaturizer* utiliza da capacidade do Rasa de definição de expressões regulares para entidades, criando uma representação vetorial levando em conta quais *regex* foram usados em uma sentença.

```yaml
pipeline:
  - name: WhitespaceTokenizer
  - name: RegexFeaturizer
  - name: LexicalSyntacticFeaturizer
  - name: CountVectorsFeaturizer
  - name: CountVectorsFeaturizer
    analyzer: char_wb
    min_ngram: 1
    max_ngram: 4
  - name: DIETClassifier
    epochs: 100
    constrain_similarities: true
  - name: EntitySynonymMapper
  - name: ResponseSelector
    epochs: 100
    constrain_similarities: true
  - name: FallbackClassifier
    threshold: 0.3
    ambiguity_threshold: 0.1
```

Figura 4.5: Pipeline pré-configurado do Rasa.

Em seguida, o componente *LexicalSyntaticFeaturizer* cria *features* léxicas e sintáticas para auxiliar na extração das entidades, considerando cada palavra da sentença e as palavras adjacentes. A configuração *default* considera se a palavra está no início ou final de uma sentença, se é em caixa alta ou baixa, se começa com uma letra maiúscula enquanto o restante está em minúscula e se é formado apenas de dígitos numéricos. Em relação as palavras adjacentes, é considerado apenas se a palavra está em caixa alta ou baixa e se inicia por uma letra maiúscula.

Para finalizar o bloco de featurização, o componente *CountVectorsFeaturizer* é utilizado duas vezes, sendo a primeira para a criação de uma representação no estilo *bag of words*, onde é considerada as ocorrências de cada palavra em um texto, e a segunda considera essa lógica aplicada à caracteres de 1 a 4 *n-grams*, limitando aos limites das palavras. Usando o caso da palavra "bandejão", exemplos de 1 a 4 *n-grams* no nível de caracter seriam "b", "ba", "ban" e "band", respectivamente.

As *features* geradas pelo bloco anterior serão utilizadas pelo bloco responsável pela identificação de intenções e entidades, sendo esse no pipeline o *DIETClassifier*. DIET é a sigla para *Double Intent Entity Transformer*, um modelo baseado em *transformers*, com sua arquitetura desenvolvida pela equipe do Rasa e utilizada em seu produto. Divulgado em 2020 no artigo "*DIET: Lightweight Language Understanding for Dialogue Systems*", possui o objetivo de ser um modelo multitarefa, extraindo entidades e intenções simultaneamente, além de não ter como obrigatoriedade o uso de modelos prétreinados

em uma certa linguagem, facilitando o desenvolvimento de *chatbots* que não utilizam a língua inglesa como base. (Bunk et al., 2020) Na figura 4.6 é possível observar o esquema da arquitetura do DIET, utilizando a frase "*play ping pong*" como entrada.

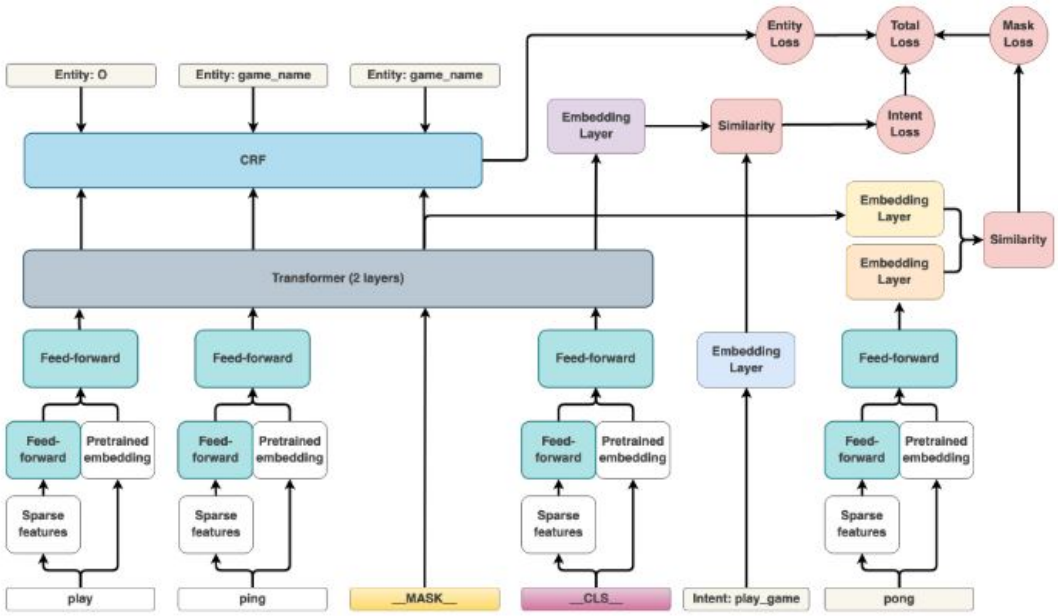

Figura 4.6: Arquitetura do DIET. (Bunk et al., 2020)

O próximo componente no pipeline é o *EntitySynonymMapper*, que funciona mapeando palavras que foram definidas como sinônimos nos dados de treinamento para um mesmo valor. Por exemplo, se o chatbot precisa reconhecer países, as variações "BR"e "Brazil"poderiam ser consideradas todas como "Brasil", dessa forma facilitando o tratamento da resposta adequada para o usuário.

Depois o *ResponseSelecto*r considera todas as possíveis respostas dada uma entrada e retorna a que tem similaridade maior, e por fim o *FallbackClassifier* atua nos casos quando o DIET não conseguiu classificar uma intenção para uma mensagem acima do limite definido. A intenção da mensagem é considerada como *"nlu_fallback"*, de forma que pode ser tratada separadamente pelo desenvolvedor, como a exibição de uma mensagem solicitando que o usuário reescreva o seu pedido.

Além de todo o apoio que o Rasa oferece em relação a NLU, o *framework* também auxilia com o histórico de conversas individuais com cada usuário, que podem ser inseridas em um banco de dados caso o desenvolvedor deseje, possibilitando novos dados para o treinamento e melhora do *chatbot*, assim

como no gerenciamento de diálogos, essencial para o funcionamento do chatbot. Tal qual o *pipeline* de NLU, o Rasa oferece um conjunto de técnicas voltadas para a escolha da próxima ação que o *chatbot* irá tomar na conversa, chamadas de *policies*. Elas podem ser baseada em uma lógica de regras definidas pelo desenvolvedor ou utilizando *machine learning*, e a configuração inicial sugerida se encontra na figura 4.7

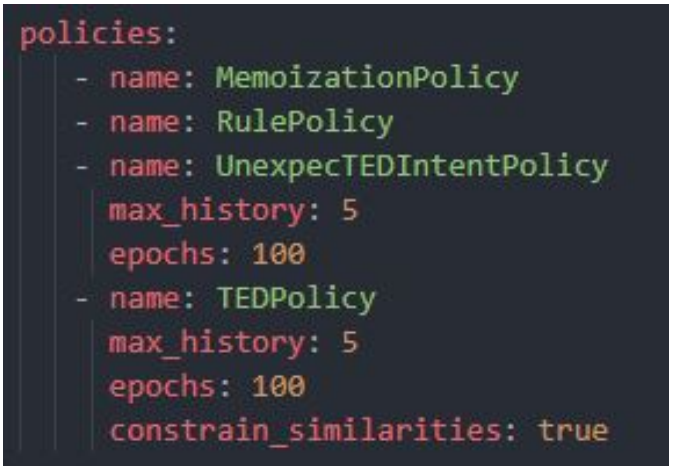

Figura 4.7: Policies no Rasa.

Assim como o treinamento do modelo utilizando frases com suas intenções e entidadas marcadas, o Rasa também permite o treinamento utilizando históricos de conversas como exemplos, onde o passo a passo das entradas, saídas e ações tomadas em uma conversa é descrito para que o *chatbot* consiga aprender. O *MemoizationPolicy* utiliza dessas stories para o seu funcionamento, e checa se existe um exemplo igual a conversa atual, é uam técnica baseada em machine learning.

Já o *RulePolicy*, como o próprio nome evidencia, é uma técnica com a lógica de regra, e considera as regras definidas pelo desenvolvedor de qual sequência de ações o chatbot deve fazer após reconhecer uma intenção específica.

O *TED(Transformer Embedding Dialogue) Policy*, assim como o DIET, utiliza *Transformer* em sua arquitetura, levando em conta as *features* de um determinado texto enviado pelo usuário e as ações passadas tomadas pelo chatbot, assim como informações de contexto salvas durante a conversa. Enquanto isso, o *UnexpecTEDIntentPolicy* também utiliza a mesma arquitetura do *TEDPolicy*, a diferença sendo que seu objetivo é encontrar as intenções com maiores chances de serem usadas por um usuário de acordo com as *stories* de exemplo que foram definidas. Dessa forma o *chatbot* conseguiria lidar com usuários que interagem de uma forma inesperada.

Apesar de várias técnicas serem definidas para lidar com o diálogo do *chatbot*, apenas aquela que retorna a maior confiança na sua predição é considerada. Em caso de empate, existe uma lista de prioridade. No contexto das técnicas sugeridas pelo Rasa, seria na seguinte ordem, de maior para menor: *RulePolicy, MemoizationPolicy, UnexpecTEDIntentPolicy* e *TEDPolicy*. (RASA, 2024)

4.4 Testes e protótipos

Para os testes iniciais de viabilidade do projeto, foi desenvolvido um *chatbot* com a possibilidade de consultar o cardápio do bandejão do dia atual e do próximo, extraíndo a informação da página da web onde fica o menu semanal, assim como solicitar ajuda sobre as funções que as funcionalidades do chatbot. Dessa forma foi definido três intenções com seus respectivos dados de treinamento. Também foi testado a conexão com o bot criado no Telegram, como pode ser observado na figura 4.8.

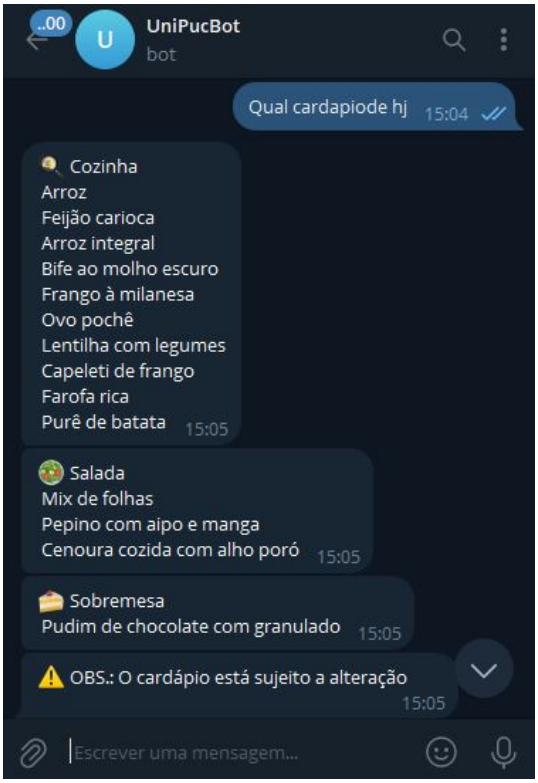

Figura 4.8: Usuário consulta cardápio do bandejão do dia através do bot no Telegram.

Utilizando o *pipeline* indicado pelo Rasa e separarando os dados para treino do modelo e teste na proporção de 80% e 20%, respectivamente, foi

calculado o desempenho do modelo 5 vezes consecutivas. Os resultados obtidos encontram-se na tabela 4.1 e na matriz de confusão na figura 4.9.

Métrica	**Score**	**Desvio Padrão**
Acurácia	0.918	0.041
F1-score	0.916	0.042
Precisão	0.937	0.031

Tabela 4.1: Avaliação do DIET no reconhecimento de intenções no teste inicial

Foi observado uma boa pontuação nas três principais métricas, enquanto a matriz de confusão complementa a análise informando que existe uma leve confusão entre as intenções cardapio_hoje e cardapio_amanha. Como o usuário pode questionar o *chatbot* de formas muito semelhantes em ambos os casos, uma forma de contornar esse problema no desenvolvimento do protótipo final seria mesclar ambas as intenções em apenas uma, e enquanto o dia de referência da consulta poderia ser uma entidade dentro da frase. De forma geral, os resultados desse teste inicial foram positivos, e com isso foi possível expandir as funcionalidades para que o *chatbot* auxilie o usuário de outras formas.

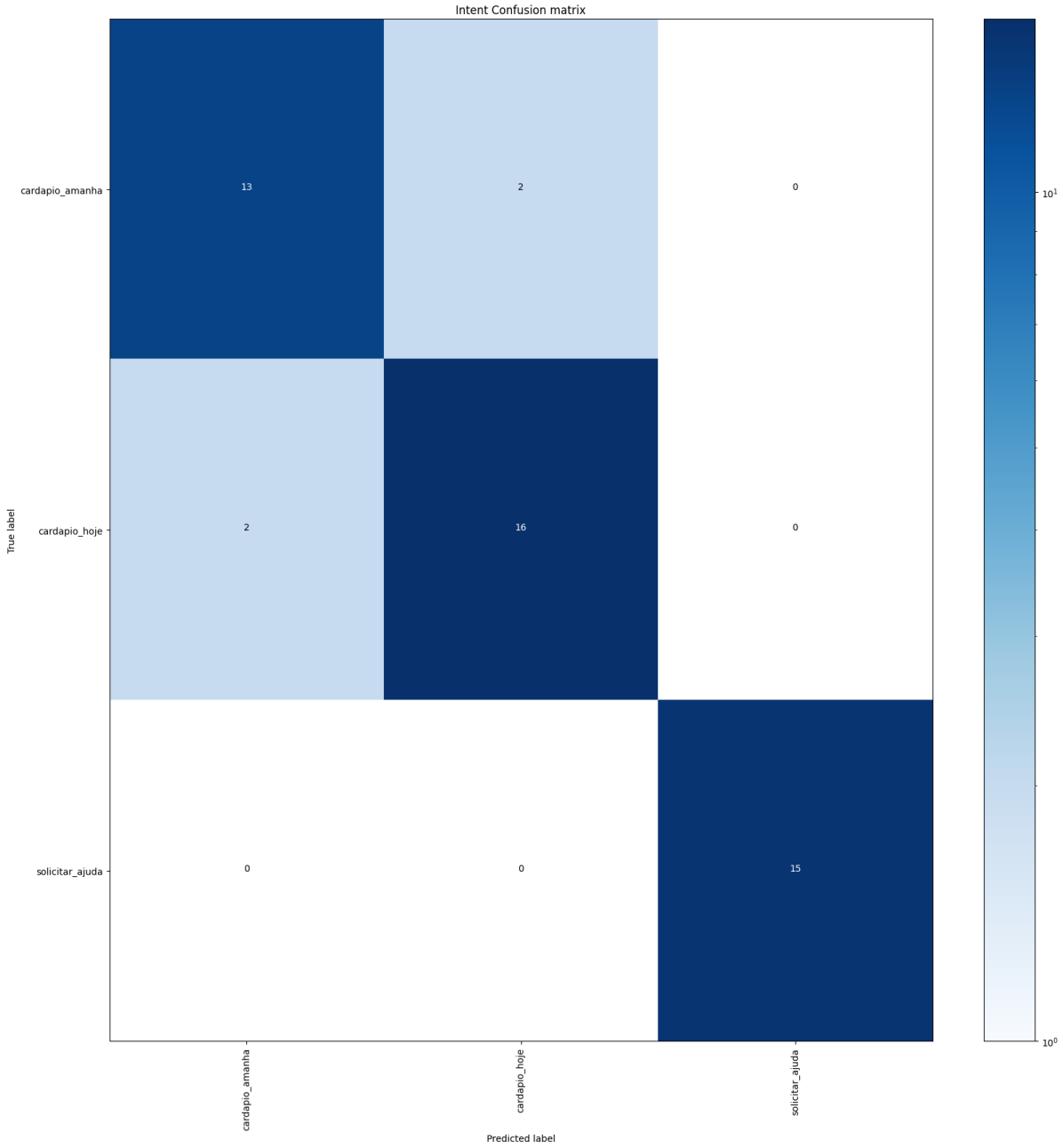

Figura 4.9: Matriz de confusão das intenções do teste inicial

5
Projeto e especificação do sistema

Em relação aos requisitos do sistema, foi definido que o usuário, utilizando a língua portuguesa, poderá:

- Consultar o cardápio do bandejão
- Consultar datas no calendário acadêmico
- Consultar o microhorário através do código ou nome da disciplina
- Consultar o microhorário através do nome do professor
- Consultar ementa de uma disciplina através de seu código ou nome da disciplina
- Cadastrar uma disciplina em sua grade horária virtual
- Consultar sua grade horária virtual
- Excluir sua grade horária virtual
- Consultar qual é a sua próxima aula
- Solicitar ajuda em relação ao funcionamento do *bot*

A arquitetura do *framework* Rasa que será utilizado para auxiliar o desenvolvimento do *chatbot* é apresentada na figura 5.1.

O usuário interage com o *chatbot* via Telegram, recebendo e enviando mensagens. A comunicação entre o Telegram e o *framework* será feita através do ngrok, uma ferramenta que permite o acesso do que está sendo desenvolvido na máquina de forma segura para a *Internet*, facilitando o desenvolvimento do protótipo.

A classe Agent funciona como uma interface para outros blocos principais do Rasa, que cuidam de funções como treinamento e carregamento do modelo de linguagem, conexão com plataformas externas, como Slack, Telegram e Facebook, tratamento de mensagens e controle do diálogo. O *NLU Pipeline* é o responsável por fazer o processamento da entrada do usuário utilizando o modelo de linguagem treinado, encontrando as entidades e intenções. Enquanto isso, o bloco nomeado *Dialogue Policies* é encarregado por escolher qual a próxima ação que o chatbot irá tomar.

Quando o Rasa é iniciado pela primeira vez com *rasa init* através da linha de comando, o *framework* gera a seguinte hierarquia de pastas e arquivos,

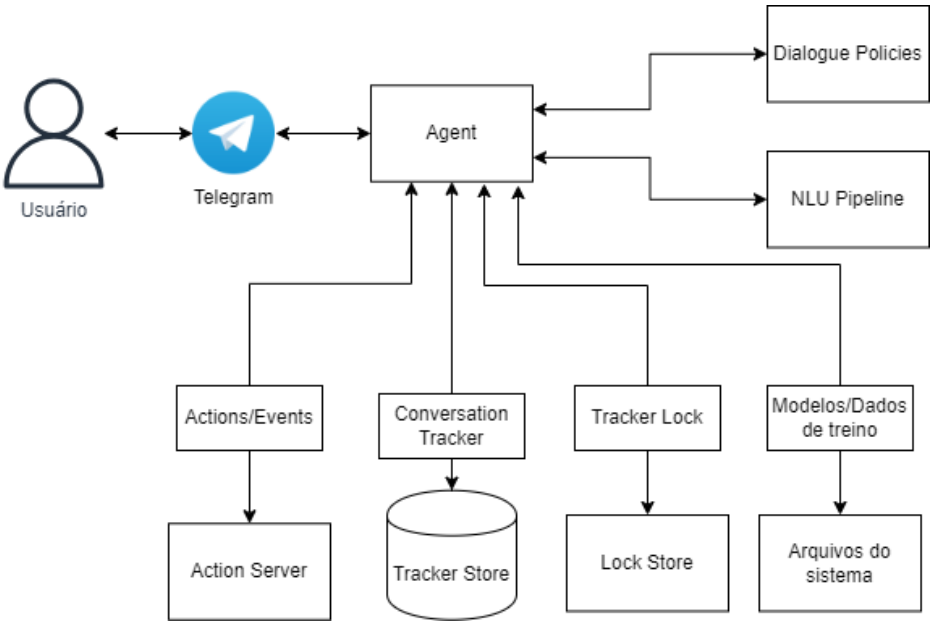

Figura 5.1: Arquitetura do framework

conforme a figura 5.1. Todas as modificações a serem feitas para criação de um *chatbot* serão realizadas dentro dessa estrutura de arquivos.

A maioria dos arquivos utiliza a linguagem de serialização de dados YAML(*YAML Ain't Markup Language*), que possui um formato simples de ser entendido por humanos e versátil, compatível com várias linguagens de programação. O conteúdo do arquivos do formato yml pode ser definido como:

- **nlu.yml:** definição das frases utilizadas para o treinamento do modelo, com as intenções e entidades anotadas
- **rules.yml:** conjunto de regras que o *chatbot* deve seguir após identificar uma determinada intenção pelo usuário
- **stories.yml:** conversas a serem utilizadas como exemplo, de forma que o modelo possa aprender a estrutura de ações tomadas de acordo com o contexto. Todas as ações tomadas pelo chatbot assim como o usuário devem ser explicitadas no arquivo, conforme o exemplo na figura 5.3.
- **test_stories.yml:** *stories* a serem utilizadas como teste, possuem a mesma estrutura de uma *story* definida para treinamento, sendo a única diferença que a mensagem por completa do usuário deve ser descrita também
- **config.yml:** arquivo de definição do *pipeline* de NLU e *policies*
- **credentials.yml:** credenciais das plataformas externas onde o *chatbot* irá ser utilizado. No caso do Telegram, é necessário o token de acesso e

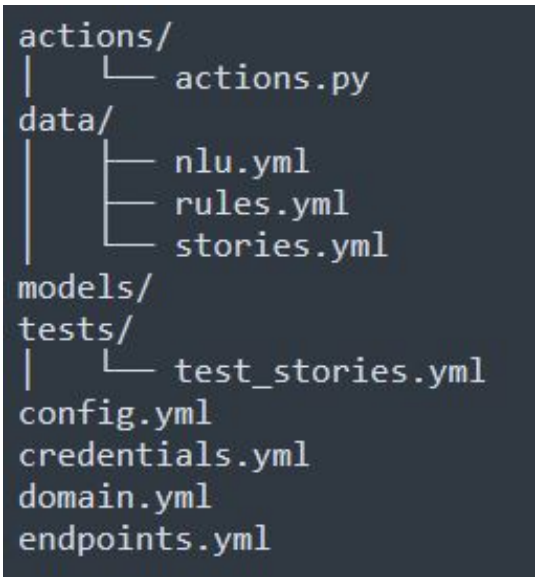

Figura 5.2: Hierarquia de arquivos

nome do bot assim como a URL do *webhook*, que no caso do protótipo será a URL disponibilizada pelo ngrok

- **domain.yml:** é o arquivo responsável pelo domínio que o *chatbot* irá atuar. Consiste na definição de entidades, intenções, *slots*, formulários e outras ações.
- **endpoints.yml:** definição do servidor onde as Actions costumizadas irão ser executadas assim como a configuração de eventuais bancos de dados utilizados

O *Action Server* fica encarregado de executar o código em Python criado pela pessoa desenvolvedora, com as funcionalidades personalizadas para o domínio do chatbot. No contexto desse protótipo, as Actions vão consultar informações salvas durante a conversa com o usuário em *slots*, assim como salvar novas informações, como é o caso da funcionalidade da grade horária. Também utilizará de *webscraping* para extrair informações de páginas na *Internet*, como o cardápio do bandejão e o informações sobre as disciplinas. Todas as Custom Actions utilizam a classe Action definida internamente no código do *framework*, sendo que os métodos *name* e *run* deverão ser reescritos de acordo com a lógica necessária. Durante o fluxo da conversa com o usuário, o *chatbot* irá procurar pelo nome da ação definido no método name, que anteriormente foi definido no arquivo domain.yml. Com isso, executará o código do método *run*. O diagrama das classes criadas para o *bot* encontra-se na figura 5.4.

```
- story: Último dia de aula da pós
  steps:
  - intent: consultar_calendario_academico
    entities:
    - assunto: término das atividades academicas
  - slot_was_set:
    - assunto: término das atividades academicas
  - action: calendario_academico_form
  - active_loop: calendario_academico_form
  - slot_was_set:
    - assunto: término das atividades academicas
  - slot_was_set:
    - requested_slot: tipo_curso
  - slot_was_set:
    - tipo_curso: pós
  - slot_was_set:
    - requested_slot: null
  - active_loop: null
  - action: action_consultar_calendario_academico
```

Figura 5.3: Definição de uma story de consulta ao calendàrio acadêmico

A *Tracker Store* é reponsável pelo armazenamento de informações do histórico de conversas com os usuários, porém não é persistente. Quando o servidor do Rasa é reiniciado os dados também são excluídos. No entanto, é possível fazer a conexão com algum banco de dados, como MongoDB ou Redis, para um armazenamento persistente, de forma que os dados podem ser utilizados futuramente para um novo treinamento do modelo. Para esse protótipo não será utilizado banco de dados para armazenamento, porém a *Tracker Store* vai ser acessada durante as conversas, visto que é onde os *slots* capturados durante a conversa são armazenados para serem utilizados pelas *Custom Actions*. Os *slots* definidos no arquivo domain.yml são nome, disciplina, turma, data, grade_horaria, tipo_curso e assunto.

Por fim, o bloco *Lock Store* certifica que as mensagens dos múltiplos usuários que estejam interagindo com o *chatbot* sejam processadas na ordem correta e para o usuário correto. O Rasa utiliza do mecanismo de sincronização conhecido como *ticket lock*, com funcionamento similar ao algoritmo de fila FIFO(*First in, first out*).

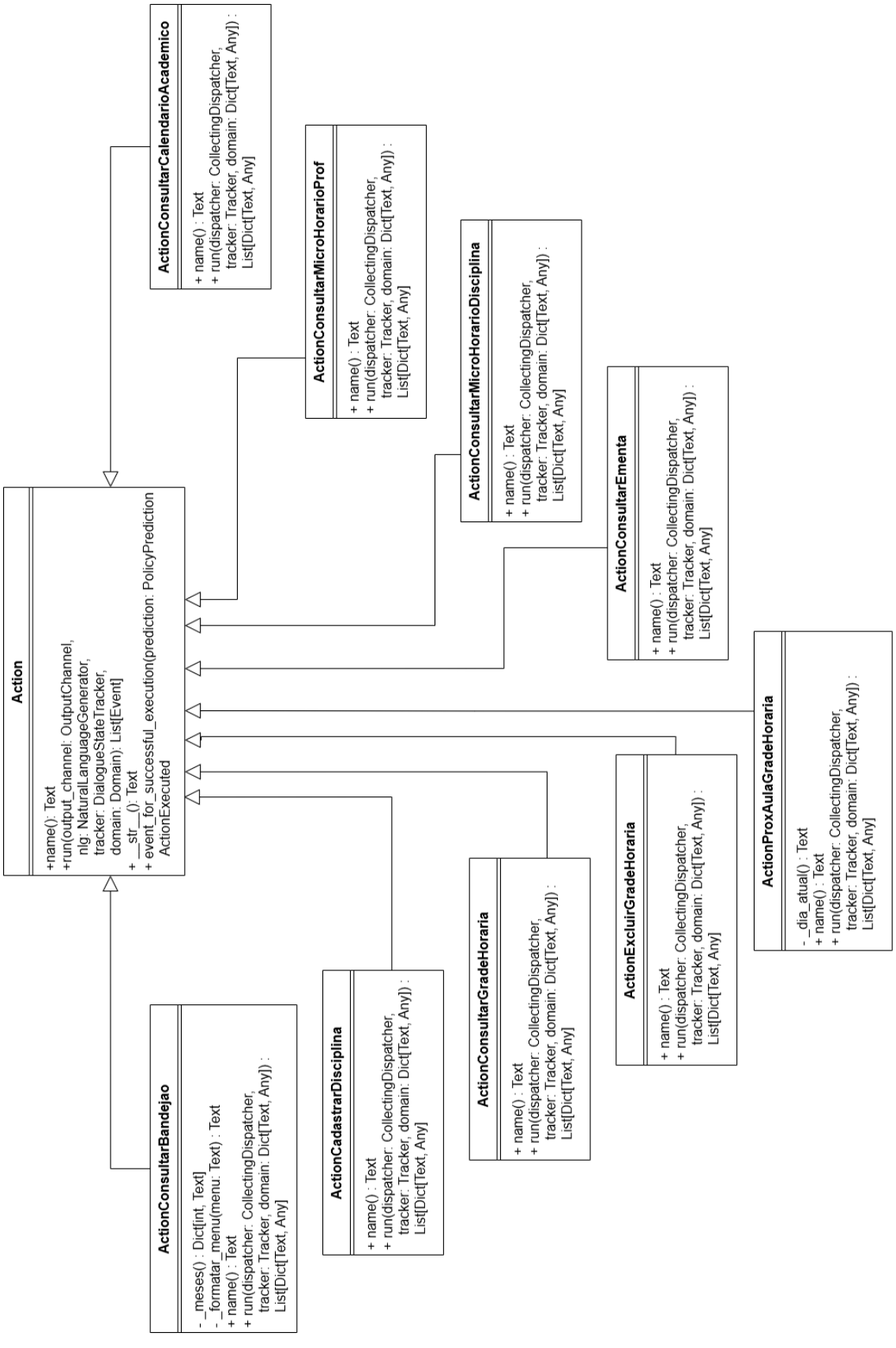

Figura 5.4: Diagrama da classe Action e subclasses herdeiras

6
Implementação e avaliação

Como descrito anteriormente, para o desenvolvimento do *chatbot* foi utilizado o Rasa, sendo criado o código em Python necessário para as funcionalidades definidas no escopo assim como os dados de treino anotados com suas respectivas entidades e intenções. O projeto foi desenvolvido em um computador com um processador Intel Core i7-5500U e 8GB de RAM, não sendo utilizado GPU para treinamento do modelo e desenvolvimento do protótipo. A versão do Python utilizada foi a 3.8.10 enquanto a versão do Rasa foi a 3.6.6.

A seguir será descrito a implementação das funcionalidades separadas por assunto.

6.1
Cardápio do bandejão

Na primeira versão do protótipo, a visualização do cardápio era feita por meio de duas intenções distintas: uma para o menu do dia e outra para o dia seguinte. Como foi observado anteriormente na avaliação com a matriz de confusão, essa abordagem não se mostrou a mais eficaz, visto que o *chatbot* tem dificuldade em classificar as intenções devido à semelhança das frases utilizadas pelo usuário.

Para contornar isso, as duas intenções foram mescladas em apenas uma intenção(consultar_bandejao), e a determinação de qual dia o usuário deseja consultar o cardápio é feita pelo reconhecimento da entidade data, que consiste de palavras como “hoje"e “amanhã", assim como suas abreviações. Todas as entidades são armazenadas em um *slot* correspondente para uso futuro, sendo que no caso do *slot* data ele já é inicializado com a palavra “hoje". Portanto, se o usuário perguntar “Qual é o cardápio do bandejão?", sem especificar a data,o *chatbot* retorna o menu do bandejão do dia atual.

Para retornar a informação do cardápio, foi criada a *Custom Action* ActionConsultarBandejao. O conteúdo do bandejão é retirada do *site* correspondente através de *webscraping*, acessando o conteúdo dos elementos HTML no código fonte e utilizando a palavra salva no *slot* de referência para exibir o cardápio do dia correto. Durante o desenvolvimento, foi observado que frequentemente a página não era atualizada. Portanto, além de um retorno de

uma mensagem de erro caso a página esteja fora do ar, também pode ser retornado quando o cardápio disponível não corresponde ao da semana atual.

6.2 Grade horária

Essa funcionalidade foi desenvolvida para facilitar a consulta do aluno a sua grade horária, permitindo o cadastro de aulas disponíveis no microhorário em uma grade virtual dentro do *chatbot*, que é armazenada para consulta posterior. O objetivo dessa função do protótipo não é estabelecer uma conexão direta com a grade oficial do aluno conforme sua matrícula no semestre, o que exigiria acesso aos dados de cadastro da faculdade e autenticação do usuário com suas credenciais, algo que está fora do escopo definido.

Em relação a grade horária, o usuário pode cadastrar uma disciplina passando o seu nome(ou código) e a turma, consultar, excluir e perguntar qual é a próxima aula na grade. A lógica dessas funções encontram-se nas classes ActionCadastrarDisciplina, ActionConsultarGradeHoraria, ActionExcluirGradeHoraria e ActionProxAulaGradeHoraria, que são excutadas após o reconhecimento das intenções cadastrar_aula, consultar_grade_horaria, excluir_grade_horaria e prox_aula_grade_horaria respectivamente.

Um ponto a ser considerado é que a identificação do DIET em relação a entidades com múltiplas palavras não é perfeita. O que ocorre é que em certos casos ele identifica uma parte da entidade e ignora o restante. Isso se torna problemático visto que existem disciplinas com nomes similares, como “Programação em C"e “Programação Modular", por exemplo. A forma encontrada de contornar esse problema é utilizar a estrutura que o Rasa denomina de *lookup table*, que gera expressões regulares de acordo com uma lista de termos definida. Caso a extração da entidade pelo DIET não funcione corretamente, considerando que o usuário escreveu uma disciplina que exista no microhorário, o *chatbot* conseguirá fazer a correspondência de forma certa.

Abaixo é possível ver o exemplo de uma frase que o DIET teve sucesso em extrair a entidade da turma, mas não do nome da disciplina durante o teste do modelo. Como foi encontrada uma correspondência na lista de disciplinas definida, a entidade correta será considerada.


```
{
    "text": "ADICIONE PROGRAM DE MICROCONTROLADORES NA GRADE, NA TURMA
    ↪  4YZ",
    "entities": [
      {
        "start": 9,
        "end": 38,
```

```
        "value": "PROGRAM DE MICROCONTROLADORES",
        "entity": "disciplina"
      },
      {
        "start": 58,
        "end": 61,
        "value": "4YZ",
        "entity": "turma"
      }
    ],
    "predicted_entities": [
      {
        "entity": "disciplina",
        "start": 9,
        "end": 19,
        "confidence_entity": 0.789400041103363,
        "value": "PROGRAM DE",
        "extractor": "DIETClassifier"
      },
      {
        "entity": "turma",
        "start": 58,
        "end": 61,
        "confidence_entity": 0.3518977761268616,
        "value": "4YZ",
        "extractor": "DIETClassifier"
      },
      {
        "entity": "disciplina",
        "start": 9,
        "end": 38,
        "value": "PROGRAM DE MICROCONTROLADORES",
        "extractor": "RegexEntityExtractor"
      }
    ]
}
```


Caso o usuário não escreva a disciplina ou a turma em sua mensagem, o *chatbot* responde pedindo que informe a informação faltante. Com as entidades necessárias reconhecidas, é feito a pesquisa no microhorário e através de *webscraping* é retornado as informações da disciplina como nome do professor e horários que ela acontece. Tudo é salvo no *slot* grade_horaria, que é responsável por guardar a lista de todas as disciplinas cadastradas. Na figura 6.1 é possível observar como é feita a adição de disciplinas na grade horária, reconhecendo tanto o nome quanto o seu código.

A consulta à grade pode ser feita de forma geral, onde todas as disciplinas

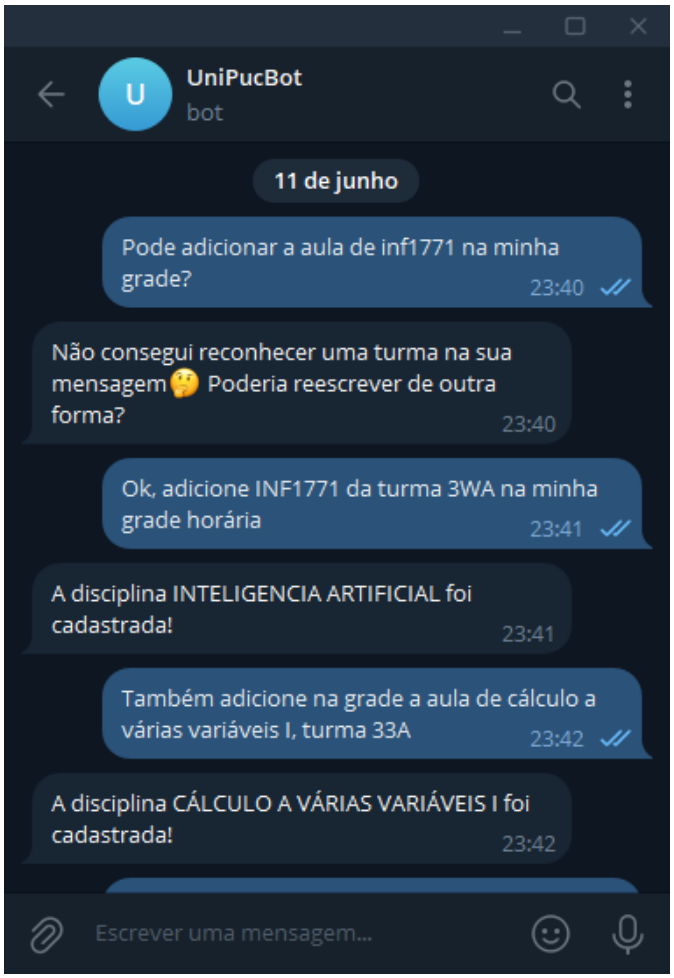

Figura 6.1: Usuário adicionando disciplinas em sua grade

e suas informações principais são mostradas para o usuário, ou em relação a aula mais próxima, que retorna o nome e sala da disciplina. Caso não tenha mais disciplinas no dia, é retornado o mais próximo em relação a horário e dia da semana, como pode ser visto na figura 6.1.

Ao excluir a sua grade horária, o *slot* que é responsável por guardar essa informação é apagado, e com isso alerta o chatbot alerta o usuário caso ele tente consultar uma grade inexistente, como é apresentado na figura 6.3.

6.3 Microhorário e Ementa

O usuário pode consultar o microhorário da faculdade através da pesquisa pelo nome do professor e da disciplina, assim como consultar mais informações de uma disciplina em sua ementa. Como explicado na seção anterior, o DIET também apresentou problemas no reconhecimento de nomes, certas vezes reconhecendo apenas o primeiro nome ou parte do sobrenome. Foi observado uma maior dificuldade do DIET de generalizar os nomes utilizados no treino

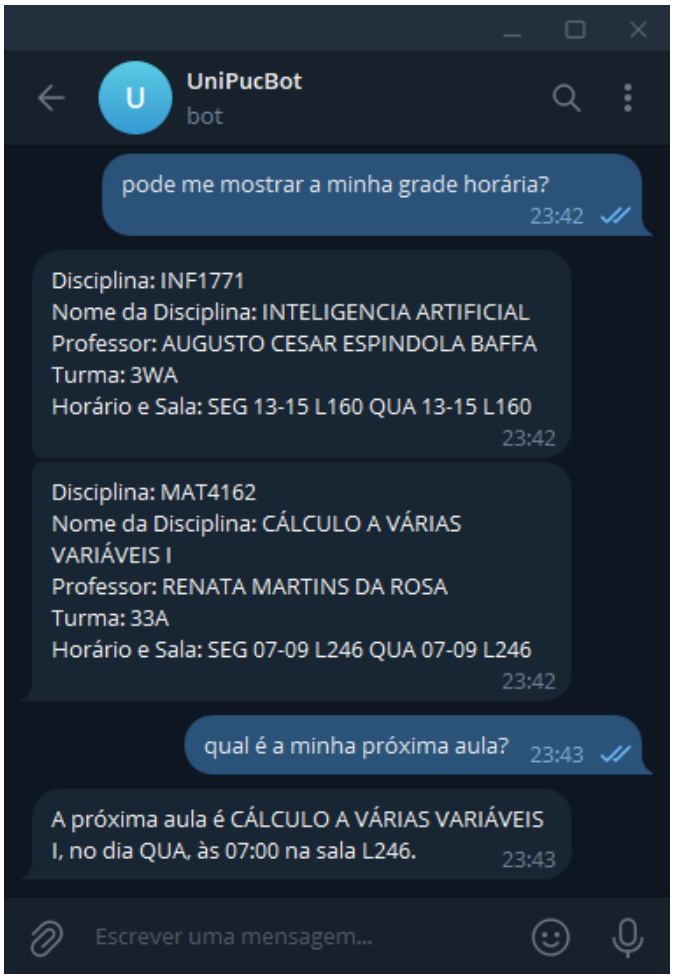

Figura 6.2: Usuário consultando sua grade horária

para outros caso em comparação com a entidade de disciplina. Por isso foi utilizado a mesma técnica de listagem dos nomes de todos os professores da faculdade para o uso da estrutura *lookup table*, de forma a atuar como *fallback* caso a extração principal de entidades não funcione.

Após a identificação das intenções consultar_microhorario_prof ou consultar_microhorario_disciplina ou consultar_ementa, o chatbot executa o código contido nas classes ActionConsultarMicroHorarioProf ou ActionConsultarMicroHorarioDisciplina ou ActionConsultarEmenta, respectivamente. O código original de pesquisa de professores, disciplina e ementa foi fornecido pelo orientador e realizada as devidas alterações para formatar a informação retornada para o *chatbot*.

6.4 Calendário Acadêmico

Ao pesquisar sobre uma data acadêmica, além da entidade assunto também é necessário o reconhecimento da entidade tipo_curso, visto que o

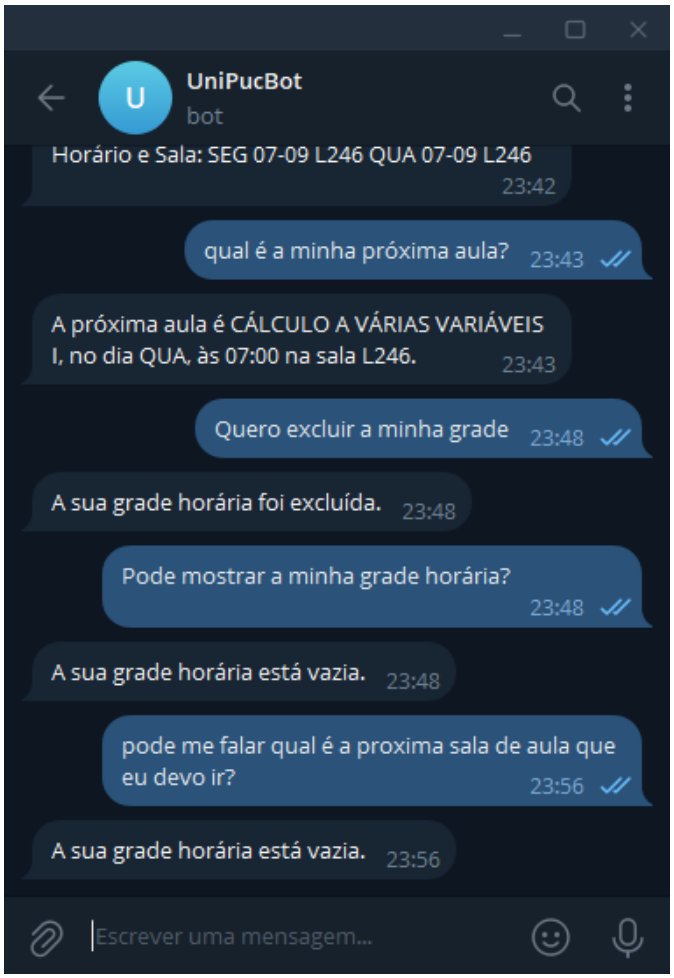

Figura 6.3: Usuário deleta sua grade horária

calendário acadêmico se divide em datas de graduação e pós-graduação, muitas delas com a mesma descrição de data em ambos os casos. Dessa forma, caso o *chatbot* não reconheça uma das entidades, é perguntado para o usuário a informação faltante, para que assim seja feita a pesquisa, como pode ser visto na figura

Ao reconhecer a intenção consultar_calendario_academico, o *chatbot* executa a classe ActionConsultarCalendarioAcademico. É utilizada a técnica de *webscraping* na página do calendário da universidade, acessando as tabelas que contém as informações tanto da graduação quanto da pós-graduação. A diferenciação das tabelas é feito através do prefixo nas classes HTML, que se iniciam com grad_ ou pos_.

Além de ser feita a desambiguação à nível de curso, também é feita com assuntos similares. Para isso é utilizada a distância de Levenshtein, que calcula a similaridade de duas strings levando em conta quantas transformações são necessárias para uma string se tornar a outra, através da técnica *fuzzy string matching*. Considerando que um usuário possa perguntar “Quando é a

renovação de matrícula?", a data que ele estaria perguntando teria a descrição "Solicitação de Renovação de matrícula"e não "Trancamento de matrícula, após a renovação de matrícula". Então o algoritmo de retorno das datas não deve considerar apenas se o termo de pesquisa está contido em uma das descrições de data, mas também qual realmente seria a intenção do usuário ao fazer aquela pesquisa.

Com o auxílio da biblioteca fuzzywuzzy no Python, após a extração das datas do calendário, a entidade assunto escrita pelo usuário é comparada com cada descrição de data de forma a encontrar as que apresentam maior similaridade. Por isso termos que são similares porém encontrados no final da frase ou após uma vírgula, como é o caso de "renovação de matrícula"em "Trancamento de matrícula, após a renovação de matrícula", sofrem uma penalidade na sua pontuação, de forma a não ter tanta influência na pontuação final de similaridade. Para o cálculo da pontuação é utilizado dois tipos de versão dessa algoritmo: *partial ratio* e *token set ratio*.

O *partial ratio* retorna a pontuação de similaridade em função da *substring* menor, sendo uma boa opção para os casos em que um texto está contido dentro do outro. Já o *token set ratio* calcula a pontuação em função do conjunto de tokens que formam a frase. Usando a proporção de peso 2 para o *partial ratio* e peso 1 para *token set ratio* para cálculo do *score* final da frase com uma média ponderada, o objetivo é facilitar o encontro de frases que não são exatamente iguais mas que passam a mesma ideia.

6.5 Ajuda e fora do escopo

Por fim, o *chatbo*t também foi treinado para identificar uma solicitação de ajuda sobre sua funcionalidade, através da intenção solicitar_ajuda, exibindo uma mensagem com os pontos principais que ele pode auxiliar. Essa mensagem também é exibida assim que o usuário começa a sua primeira conversa com o *bot*.

Foi criada também uma intenção chamada fora_do_escopo, para lidar com casos de frases do usuário que não correspondem a exemplos treinados pelo modelo.

6.6 Avaliação

O protótipo final teve 11 intenções e 6 entidades, contendo uma quantidade total de 260 frases treinadas no modelo de linguagem. Para testar o modelo NLU foi utilizado *crossvalidation*, onde os dados são separados em k

partes(*folds*), e a cada bateria de testes os dados de k-1 *folds* são usados para teste, enquanto o restante é utilizado para treinamento. Esse processo é repetido k vezes de forma que em cada iteração é feito o teste com um conjunto diferentes de frases.

Utilizando k = 10, o resultado das métricas principais em relação a identificação de intenções se encontra na tabela 6.1, enquanto as métricas da identificação de entidades podem ser observadas na tabela 6.2.

Métrica	**Score**	**Desvio Padrão**
Acurácia	0.935	0.046
F1-score	0.929	0.049
Precisão	0.944	0.042

Tabela 6.1: Avaliação do DIETClassifier no reconhecimento de intenções

Métrica	**Score**	**Desvio Padrão**
Acurácia	0.901	0.025
F1-score	0.758	0.060
Precisão	0.857	0.060

Tabela 6.2: Avaliação do DIETClassifier no reconhecimento de entidades

É possível notar que o DIET se comporta melhor em relação as intenções do que as entidades. Apesar da accuracy e precision ser acima de 0,8 nas entidades, o F1-score, sendo uma média harmônica da cobertura e da precisão, resultou em apenas 0,697, o que significa que existe um número razoável de entidades que ele não obteve sucesso na extração.

Essa análise se consolida ao olhar para as matrizes de confusão da intenção e da entidade, nas figuras 6.4 e 6.5, respectivamente. A análise é retornada em relação ao DIET e não um conjunto de todas as técnicas, como é o caso das entidades nome e disciplina que para auxiliar esse problema utilizam de regex junto com a lista de informações de suporte para cada caso. É notável que o Rasa obteve um desempenho bem melhor no reconhecimento das intenções, visto que a matriz de confusão das entidades teve um resultado bem mais espalhado, apesar de conseguir acertar um número significativo, não tem uam confiabilidade muito alta.

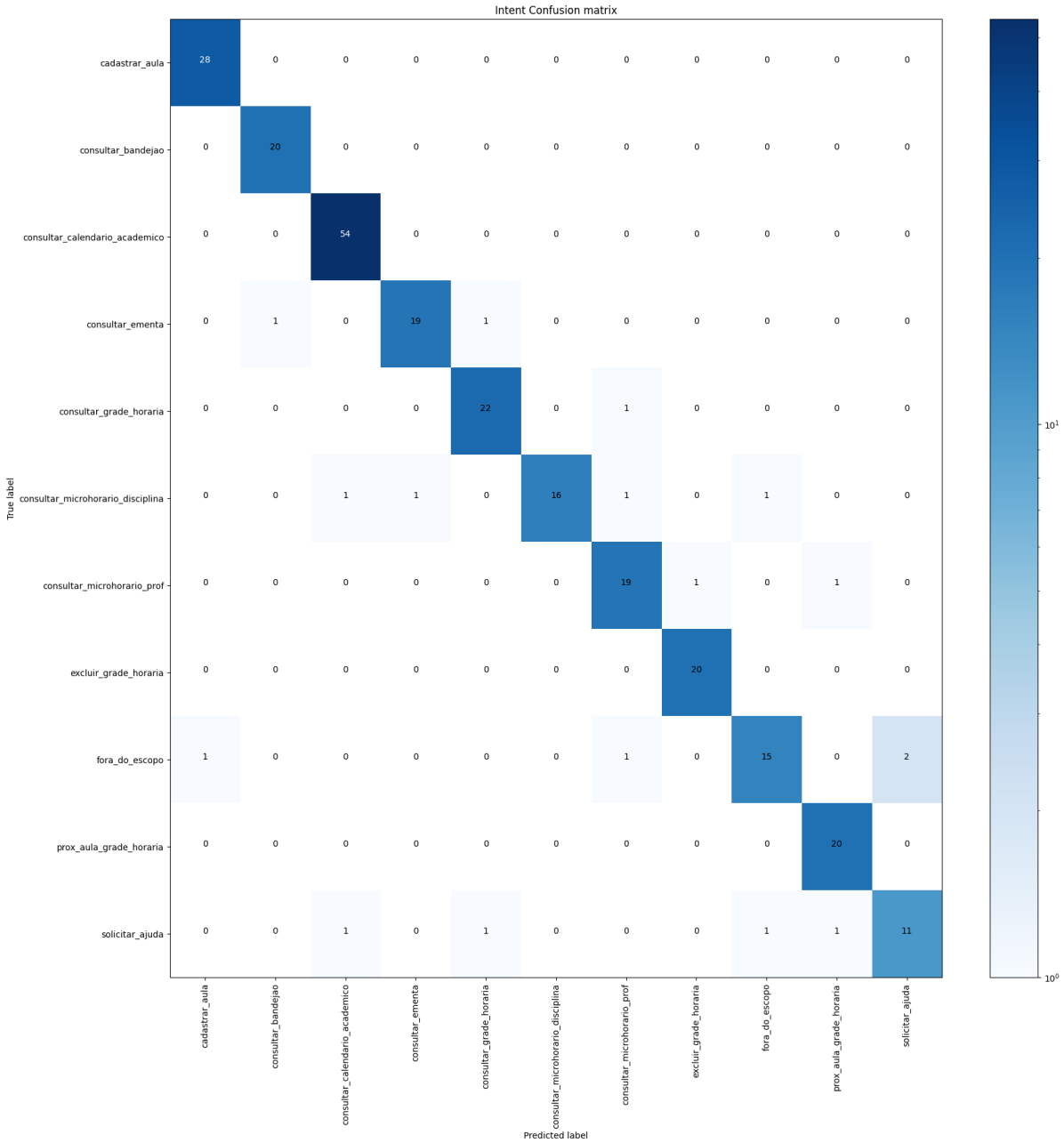

Figura 6.4: Matriz de confusão das intenções

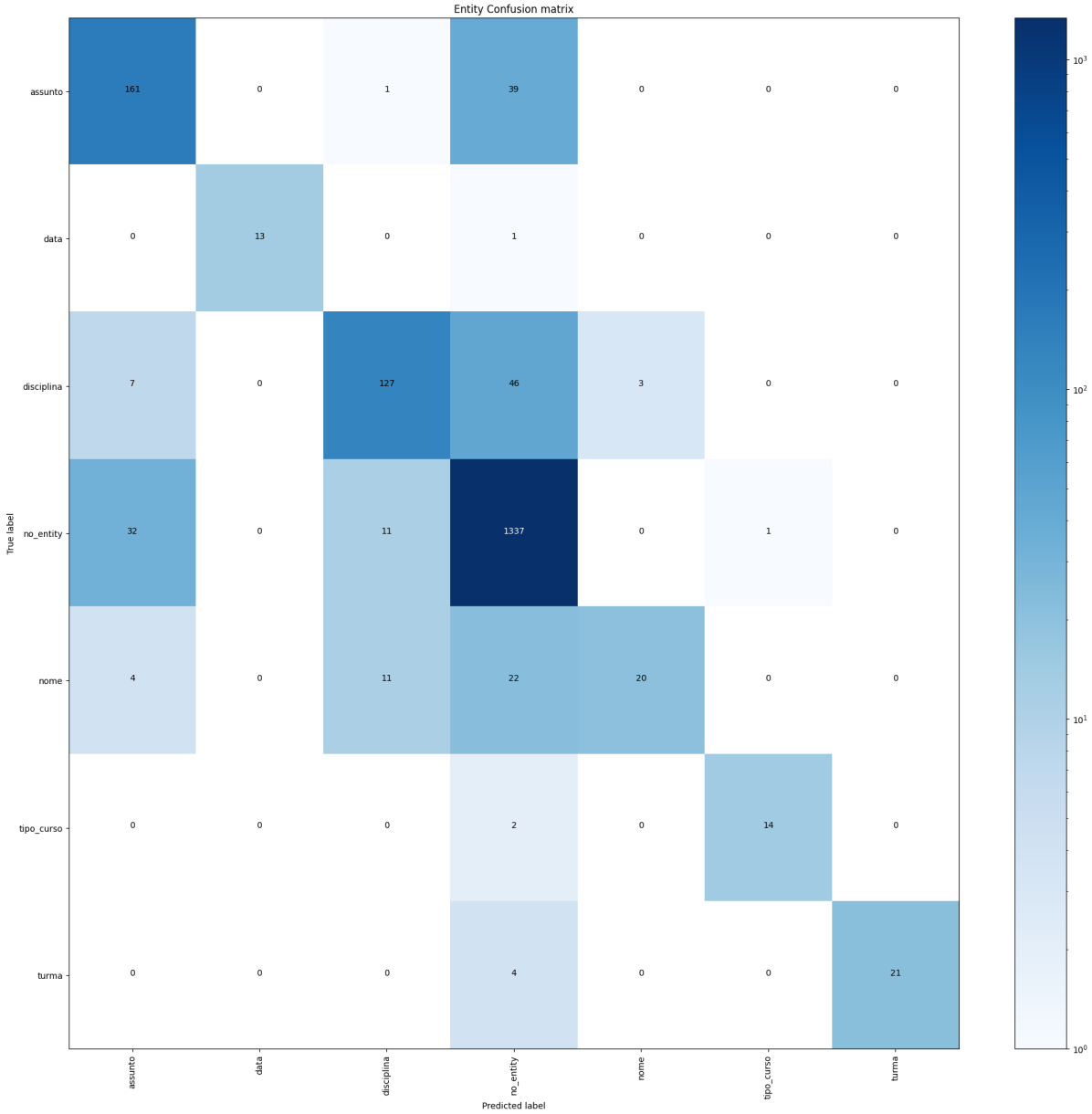

Figura 6.5: Matriz de confusão das entidades

7
Considerações finais

Através do protótipo final foi verificado que o *framework* Rasa oferece um bom apoio para desenvolvimento de sistemas conversacionais, com baixos requisitos mínimos e custo para ser feito o desenvolvimento. Destaca-se no reconhecimento das intenções e controle do diálogo com os usuários, sendo de fácil implementação em diferentes plataformas também. No entanto, existem melhorias a serem feitas em relação a parte de entidade. Seria válido para trabalhos futuros uma maior exploração do pipeline de NLU, como a inclusão de modelo pré-treinados como o SpaCy, visto que já foi feito o treinado em uma corpora de dados maior o que poderia fornecer *features* extras para o DIET que auxiliasse na identificação correta de cada entidade.

Considerando o escopo de consultas feitas em um ambiente de uma universidade, o *chatbot* poderia atender um número maior de questionamentos, como notícias recentes, localização de prédios e laboratórios no campus, indicação de disciplinas para matrícula e demais assuntos. É importante também coletar mais dados de perguntas de usuários de diferentes faixas etárias e cursos, de forma a alimentar o modelo com dados diversos. O Rasa possui uma funcionalidade que pode ser acessada através de rasa interactive na linha de comando, permitindo que a pessoa converse em tempo real com o *chatbot* e veja as escolhas que ele fez, conseguindo corrigir durante o processo caso ele comita algum erro, de forma que no final da sessão ele salva todos os dados gerados nos arquivos utilizados para treinamento. O trabalho teve um limitado número de funcionalidades então seria interessante aumentar esse número.

Para trabalhos futuros, planeja-se ampliar as funcionalidades que ficaram de fora do escopo atual, como permitir a pesquisa e localização de laboratórios e outros pontos de interesse dentro da universidade e criação de um FAQ mais robusto, incluindo as normas acadêmicas e outras dúvidas que os alunos podem ter em outras frentes, como a biblioteca e bolsas. Pretende-se explorar outras técnicas que permitam um resultado melhor em relação a extração de entidades das sentenças, melhorando a experiência do usuário e com isso o uso mais frequente da ferramenta.

Referências

[Adamopoulou e Moussiades, 2020] ADAMOPOULOU, E.; MOUSSIADES, L.. **Chatbots: History, technology, and applications**. Machine Learning with Applications, 2:100006, December 2020.

[Bunk et al., 2020] BUNK, T.; VARSHNEYA, D.; VLASOV, V. ; NICHOL, A.. **Diet: Lightweight language understanding for dialogue systems**. 2020.

[Caldarini et al., 2022] CALDARINI, G.; JAF, S. ; MCGARRY, K.. **A literature survey of recent advances in chatbots**. Information, 13(1):41, January 2022.

[Chowdhary, 2020] CHOWDHARY, K.. **Fundamentals of Artificial Intelligence**, p. 610. Springer New Delhi, 2020.

[Colby et al., 1972] COLBY, K. M.; HILF, F. D.; WEBER, S. ; KRAEMER, H. C.. **Turing-like indistinguishability tests for the validation of a computer simulation of paranoid processes**. Artificial Intelligence, 3:199–221, September 1972.

[ELIZAGEN, 2023] SHRAGER, J.. **Elizagen - the original eliza**, 2023. [Online; accessed 11-October-2023].

[Google, 2023] COLLINS, E.. **Lamda: our breakthrough conversation technology**, 2023. [Online; accessed 25-October-2023].

[Google, 2024] HSIAO, S.. **Bard becomes gemini: Try ultra 1.0 and a new mobile app today**, 2024. [Online; accessed 19-October-2023].

[Hussain et al., 2019] HUSSAIN, S.; AMERI SIANAKI, O. ; ABABNEH, N.. **A survey on conversational agents/chatbots classification and design techniques**. In: Barolli, L.; Takizawa, M.; Xhafa, F. ; Enokido, T., editors, WEB, ARTIFICIAL INTELLIGENCE AND NETWORK APPLICATIONS, p. 946–956, Cham, 2019. Springer International Publishing.

[JABBERWACKY, 2023] CARPENTER, R.. **jabberwacky**, 2023. [Online; accessed 20-October-2023].

[Luo et al., 2022] LUO, B.; LAU, R. Y. K.; LI, C. ; SI, Y.-W.. **A critical review of state-of-the-art chatbot designs and applications**. WIREs Data Mining and Knowledge Discovery, 12(1):e1434, January 2022.

[MarkTechPost, 2023] ISLAM, A.. **A history of generative ai: From gan to gpt-4**, 2023. [Online; accessed 24-October-2023].

[Mauldin, 1994] MAULDIN, M. L.. **Chatterbots, tinymuds, and the turing test entering the loebner prize competition**. In: PROCEEDINGS OF THE TWELFTH AAAI NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, AAAI'94, p. 16–21. AAAI Press, 1994.

[Meta, 2024] META. **Meta llama**, 2024. [Online; accessed 19-Jun-2024].

[OpenAI, 2023] **What is chatgpt**, 2023. [Online; accessed 24-October-2023].

[Pérez-Soler et al., 2021] PÉREZ-SOLER, S.; JUÁREZ-PUERTA, S.; GUERRA, E. ; DE LARA, J.. **Choosing a chatbot development tool**. IEEE Software, 38(4):94–103, June 2021.

[RASA, 2023] RASA. **Picpay case study: Conversational ai in financial services**, 2024. [Online; accessed 20-Apr-2024].

[RASA, 2024] RASA. **Policies**, 2024. [Online; accessed 06-Jun-2024].

[Sun et al., 2017] SUN, S.; LUO, C. ; CHEN, J.. **A review of natural language processing techniques for opinion mining systems**. Information Fusion, 36:10–25, 2017.

[Taori et al., 2024] TAORI, R.; GULRAJANI, I.; ZHANG, T.; DUBOIS, Y.; LI, X.; GUESTRIN, C.; LIANG, P. ; HASHIMOTO, T. B.. **Alpaca: A strong, replicable instruction-following model**, 2024. [Online; accessed 19-Jun-2024].

[Tero et al., 2016] TERO, P.; ZAITSEV, I. ; CARPENTER, R.. **Cleverbot data for machine learning**, 2024. [Online; accessed 19-May-2024].

[Tunstall, 2022] TUNSTALL, L.; VON WERRA, L. ; WOLF, T.. **Natural Language Processing with Transformers, Revised Edition**, p. 1,11,19,59. 1,59, Dordrecht, 2022.

[Vaswani et al., 2017] VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, Ł. ; POLOSUKHIN, I.. **Attention is all you need**. Advances in neural information processing systems, 30, 2017.

[Wallace, 2009] WALLACE, R. S.. **The Anatomy of A.L.I.C.E.**, p. 181–210. Springer Netherlands, Dordrecht, 2009.

[Warmerdam, 2021] WARMERDAM, V.. **Intents entities: Understanding the rasa nlu pipeline**, 2021. [Online; accessed 12-Feb-2024].

[Weizenbaum, 1966] WEIZENBAUM, J.. **Eliza—a computer program for the study of natural language communication between man and machine**. Commun. ACM, 9(1):36–45, January 1966.