UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

GABRIEL VIEIRA RODRIGUES

CHATBOT PARA PEDIDOS ONLINE EM RESTAURANTES BASEADO EM IDENTIFICAÇÃO DE INTENÇÕES E EXTRAÇÃO DE ENTIDADES UTILIZANDO A FERRAMENTA RASA OPEN SOURCE

CURITIBA

2022

GABRIEL VIEIRA RODRIGUES

CHATBOT PARA PEDIDOS ONLINE EM RESTAURANTES BASEADO EM IDENTIFICAÇÃO DE INTENÇÕES E EXTRAÇÃO DE ENTIDADES UTILIZANDO A FERRAMENTA RASA OPEN SOURCE

Chatbot for online ordering in restaurants based on identifying intentions and extracting entities using the Rasa Open Source Platform

Trabalho de Conclusão de Curso apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação do Curso de Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. João Alberto Fabro

CURITIBA

2022

GABRIEL VIEIRA RODRIGUES

CHATBOT PARA PEDIDOS ONLINE EM RESTAURANTES BASEADO EM IDENTIFICAÇÃO DE INTENÇÕES E EXTRAÇÃO DE ENTIDADES UTILIZANDO A FERRAMENTA RASA OPEN SOURCE

Trabalho de Conclusão de Curso apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação do Curso de Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 17/dezembro/2022

André Eugenio Lazzaretti, Dr.
Departamento Acadêmico de Eletrônica
Universidade Tecnológica Federal do Paraná

Paulo Cézar Stadzisz, Dr.
Departamento Acadêmico de Informática
Universidade Tecnológica Federal do Paraná

João Alberto Fabro, Dr.
Departamento Acadêmico de Informática
Universidade Tecnológica Federal do Paraná

CURITIBA

2022

Dedico este trabalho à minha família, pelos momentos de ausência.

AGRADECIMENTOS

Agradeço primeiramente à minha família. Meus pais, Walter e Alzeni e minha irmã Isabela, que me deram toda a base e suporte para que pudesse estar focado desenvolvendo esse trabalho.

Agradeço também à minha namorada Desireé, que sempre me apoiou e participou ativamente do projeto, contribuindo com os dados utilizados.

Ao meu orientador professor João Fabro também deixo o meu muito obrigado, pelos esforços em atender aos prazos apertados e pelas correções e conselhos sempre precisos.

E por fim, agradeço a todos os docentes e amigos da UTFPR que fizeram parte dessa longa jornada, possibilitando que pudesse me tornar um profissional de qualidade e realizar o sonho de utilizar a tecnologia para resolver problemas da nossa sociedade. Sem todos esses pilares, eu não teria chegado até aqui. Muito obrigado a todos(as).

RESUMO

Devido as medidas de isolamento social impostas com o avanço da pandemia da COVID-19, muitos estabelecimentos - incluindo restaurantes e lanchonetes, se viram obrigados a receber pedidos de forma online para *delivery* e retirada, para continuar o seu funcionamento. Uma parcela desses pedidos são feitos por aplicativos de mensagem, onde há uma interação direta via texto entre o cliente e o estabelecimento. Porém essa interação pode ser muitas vezes problemática: para o cliente pode haver uma experiência negativa devido a atrasos nas respostas do estabelecimento e ambiguidades nos pedidos. E do lado do estabelecimento, há a dificuldade de escalabilidade desse atendimento à medida que o número de pedidos cresce, a diminuição da produtividade de colaboradores que além das suas tarefas usuais, também precisam realizar o atendimento online, além da perda de pedidos. Visando solucionar ao menos uma parte desses problemas, esse projeto aplicou técnicas de processamento de linguagem natural - utilizando o *framework* Rasa Open Source, para criação de um *chatbot*, que seja capaz de interagir e entender as intenções de clientes, ao realizar pedidos via chat em restaurantes e lanchonetes. Foi desenvolvido um protótipo de *chatbot* que foi avaliado com base nas métricas de desempenho das tarefas de classificação - a partir de dados coletados de conversas de um experimento simulado, resultando em 94% de acurácia para a tarefa de identificação de intenções.

Palavras-chave: Processamento de Linguagem Natural; Chatbot; Rasa Framework.

ABSTRACT

Due to the social isolation measures imposed with the advance of the COVID-19 pandemic, many establishments - including restaurants and snack bars, were forced to process orders online for delivery and pick up, to keep the operation on. One portion of these requests are made through messaging applications, where there is direct interaction via text between the customer and the establishment. However, this interaction can be sometimes problematic: for the customer there may be a negative experience due to delays and order ambiguities. And on the establishment side, there is the difficulty of scaling this service as the number of orders grows, the decrease in the productivity of employees who have to perform online service beyond their other duties, in addition to the order loss. In order to solve some of these problems, this project have applied natural language processing techniques - using the Rasa Open Source framework, to create a chatbot that is able to interact and understand the intentions of customers, automatic processing orders via chat in restaurants and snack bars. A chatbot prototype was developed and evaluated based on the performance metrics of the classification tasks - from data collected from conversations in a simulated experiment, resulting in 94% accuracy for the intention identification task.

Keywords: Natural Language Processing; Chatbot; Rasa Framework.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTAGEM DE CÓDIGOS FONTE

LISTA DE ABREVIATURAS E SIGLAS

Siglas

API	*Application Programming Interface* - Interface de Programação de Aplicação
CRISP-DM	*Cross Industry Standard Process for Data Mining - Processo Padrão Inter-Indústrias para Mineração de Dados*
DIET	*Dual Intent Entity Transformer* - Transformador Duplo de Intenção e Entidade
HTTP	*Hypertext Transfer Protocol* - Protocolo de Transferência de Hipertexto
IA	Inteligência Artificial
NLP	*Natural Language Processing - Processamento de Linguagem Natural*
NLU	*Natural Language Understanding - Compreensão de Linguagem Natural*
RNN	*Recurrent Neural Network* - Rede Neural Recorrente

SUMÁRIO

1 INTRODUÇÃO

Nesta seção serão introduzidos a motivação e os objetivos do projeto.

1.1 Considerações iniciais

Alimentados pelo crescimento da internet móvel e dos *smartphones*, os aplicativos de mensagens instantâneas se tornaram essenciais na comunicação diária da população ao redor do mundo. De acordo com Wan *et al.* (2019), em 2019 mais de 66% da população mundial (4,92 bilhões) estava usando telefones celulares e mais da metade deles estava usando mídias sociais para se comunicar.

Com o avanço dessa tecnologia, e entendendo a necessidade de melhorar o diálogo com o consumidor, as empresas também aderiram ao uso dos aplicativos de mensagens instantâneas para atendimento. De acordo com estudo feito em 2021 por Zendesk[1], canais de mensagens como WhatsApp e Facebook estão se tornando novos favoritos do consumidor. Quase um terço dos clientes disse ter enviado mensagens para empresas por meio de aplicativos de comunicação pela primeira vez em 2020. E essa tendência deve se manter, já que 74% disseram que continuariam a se comunicar com as empresas por meio de mensagens.

Acompanhando essa tendência de crescimento, estabelecimentos como restaurantes e lanchonetes também estão utilzando aplicativos de mensagens para receber pedidos, em sua maioria para *delivery* e *take away*. De acordo com Resendes (2021), 60% dos consumidores dos EUA pediram entrega ou comida para viagem uma vez por semana e 59% dos pedidos de restaurantes da geração *millenium* foram para entrega. Entre os benefícios para os restaurantes de receber pedidos por aplicativos de mensagens, pode-se destacar a ausência de taxas impostas por *marketplaces*, e a criação de um canal de comunicação direta com o consumidor.

No entanto, os pedidos vindos através de mensagens exigem que um(a) colaborador(a) do estabelecimento esteja disponível para realizar o atendimento, tarefa que acaba tirando-o(a) de sua função principal e acarreta em outros problemas como perda de pedidos devido ao atraso nas respostas e possíveis imprecisões no registro dos pedidos. Este é um campo em que novas tecnologias de inteligência artificial têm muito a contribuir, e os chatbots representam uma mudança potencial na forma como os clientes interagem com o indústria de restaurantes digitalmente (BRANDTZAEG; FØLSTAD, 2017).

Segundo Shawar e Atwell (2005) “(...) Chatbots são sistemas de conversação por máquina que interagem com usuários humanos por meio de linguagem conversacional natural“. Os *chatbots* são desenvolvidos para evocar interações humanas por meio de texto, sendo capazes de identificar intenções do usuário (BRANDTZAEG; FØLSTAD, 2017). Com as habilidades selecionadas instaladas, os *chatbots* podem ajudar os clientes a encontrar restaurantes, fazer reservas e fazer pedidos de comida para viagem, autonomamente (BARACK, 2018).

[1] <https://www.zendesk.com/customer-experience-trends-2021/> - Acesso em 18 de out. 2022.

A partir dessas constatações foi identificada a oportunidade de desenvolver um *chatbot* para automatizar pedidos em restaurantes via troca de mensagens de texto. Para isso foi utilizado o *framework* de código livre Rasa[2], que utiliza técnicas e algoritmos de aprendizado de máquina para identificar intenções - que representam a intenção/propósito da mensagem do usuário, e extrair entidades de um texto - que representam peças importantes de informação do texto enviado. Para realizar tais identificações e extrações, os modelos de treinamento do Rasa necessitam um conjunto de dados de treinamento previamente classificados com suas respectivas intenções e entidades. Nesse projeto foi utilizado um conjunto de conversas de pedidos reais - realizados e cedidos por um restaurante voluntário, contendo vários exemplos do fluxo de conversa pretendido pelo *chatbot* desenvolvido.

1.2 Objetivos

Nesta seção serão apresentados o objetivo geral e os objetivos específicos do projeto.

1.2.1 Objetivo geral

Desenvolver um *chatbot* que seja capaz de identificar intenções e extrair entidades de mensagens de texto enviadas por clientes em pedidos online de restaurantes, utilizando algoritmos e técnicas de Processamento de Linguagem Natural (PLN).

1.2.2 Objetivos específicos

- Classificar as intenções de saudação, solicitação de cardápio, pedido de itens do cardápio, finalização de pedido e escolha de método de entrega e meio de pagamento dentro de uma base de conversas de pedidos realizados, fornecida por um restaurante voluntário.
- Classificar as entidades de sabor e quantidade de itens de um pedido, endereço informado para entrega e meio de pagamento escolhido dentro de uma base de conversas de pedidos realizados, fornecida por um restaurante voluntário. Alguns exemplos dos dados de treinamento podem ser visualizados no Anexo A.
- Alimentar e treinar os modelos do Rasa com os dados classificados.
- Integrar o *chatbot* com a *Application Programming Interface* (API) do Google Maps[3] para validação do endereço inserido pelo usuário, no caso de pedidos para *delivery*.

[2] <https://rasa.com/docs/rasa/> - Acesso em 16 de set. de 2022.

[3] <https://developers.google.com/maps/documentation/places/web-service/search> - Acesso em 25 de set. de 2022.

- Integrar o *chatbot* com o aplicativo Telegram[4], que será responsável pela interface de comunicação com o usuário.
- Realizar testes com um grupo de 10 usuários voluntários, que devem efetuar dois pedidos em um restaurante fictício - através do aplicativo Telegram.
- Mensurar a acurácia de classificação de cada intenção identificada nos testes pelo algoritmo do *chatbot*.

[4] <https://telegram.org/> - Acesso em 12 nov. 2022.

2 FUNDAMENTAÇÃO TEÓRICA E MATERIAIS

Este capítulo aborda os conceitos necessários para compreender esse projeto. Ele está dividido em cinco seções: aprendizado de máquina, processamento de linguagem natural, compreensão de linguagem natural, a ferramenta Rasa, e critérios de avaliação para sistemas de aprendizado de máquina.

2.1 Aprendizado de máquina

Aprendizado de máquina pode ser entendido como um conjunto de métodos que tem como objetivo detectar padrões nos dados e a partir destes padrões, prever dados futuros ou realizar outros tipos de tomada de decisão (MURPHY, 2013). Os algoritmos de aprendizado de máquina podem ser classificados em várias categorias amplas por seu estilo de aprendizado de acordo com Goodfellow, Bengio e Courville (2016). No aprendizado supervisionado, o algoritmo constrói um modelo matemático a partir de um conjunto de dados que contém tanto as entradas quanto as saídas desejadas. Algoritmos de classificação e algoritmos de regressão são tipos de aprendizado supervisionado. Existem também algoritmos de aprendizado semi-supervisionado, que utilizam uma combinação de dados rotulados e dados não rotulados para fazer previsões melhores para novos pontos de dados do que usando apenas os dados rotulados. No aprendizado não supervisionado, o algoritmo constrói um modelo matemático a partir de um conjunto de dados que contém apenas entradas e nenhum rótulo de saída desejado. Algoritmos de aprendizagem não supervisionada são usados para encontrar estrutura/padrões nos dados, como agrupar os pontos de dados em categorias. Já algoritmos de aprendizado por reforço recebem *feedback* em forma de reforço positivo ou negativo em um ambiente potencialmente incerto e complexo e são usados, por exemplo, em veículos autônomos (GOODFELLOW; BENGIO; COURVILLE, 2016).

2.2 Processamento de Linguagem Natural

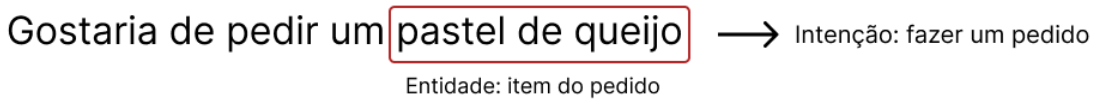
Processamento de Linguagem Natural (*Natural Language Processing* NLP) é um campo de pesquisa que emprega técnicas computacionais com o propósito de aprender, entender e produzir contéudo em linguagem natural humana (HIRSCHBERG; MANNING, 2015). O desenvolvimento de métodos de NLP depende cada vez mais de abordagens baseadas em dados que ajudam na construção de modelos de predição/classificação mais poderosos e robustos. Avanços recentes em poder computacional, bem como maior disponibilidade de dados (*big data*), permitem que a utilização de abordagens baseadas em aprendizado supervisionado tenha se tornado uma das mais atraentes no domínio da NLP (ZHANG; ZHAO; LECUN, 2015). Apesar desse desenvolvimento ter avançado nas últimas décadas, NLP ainda é um campo desafiador

pois quando se trata em linguagem natural, é importante levar em conta que o que está escrito não significa necessariamente o que deve ser interpretado. As pessoas podem escrever ou falar de maneiras diferentes para conseguir algo específico, podem cometer erros, digitar as palavras erradas, falar ou escrever frases fragmentadas (DINESH *et al.*, 2021).

2.3 Compreensão de Linguagem Natural

Segundo Abdellatif *et al.* (2022), o principal objetivo de um algortimo de Compreensão de Linguagem Natural (*Natural Language Understanding* NLU) é extrair dados estruturados de uma entrada de linguagem não-estruturada. Em particular, extrair intenções e entidades das entradas dos usuários. Um exemplo de intenção e entidade de uma frase pode ser visto na Figura 1. Para compreender a linguagem humana, a máquina precisa dividir a entrada do usuário em parágrafos, frases e palavras. Além disso, deve aprender a reconhecer as relações entre as diferentes palavras, extrair o significado exato do texto, compreender frases em diferentes situações e considerar o contexto do discurso anterior (TEMBHEKAR; KANOJIYA, 2017). O contexto é um componente essencial quando se trata de um algortimo de NLU, de acordo com McShane (2017) "(...) Como a robótica, NLU é naturalmente perseguido para realizar tarefas específicas em um domínio específico para o qual o agente é fornecido com o conhecimento e as capacidades de raciocínio necessárias".

Figura 1 – Exemplo intenção e entidade em uma frase



Fonte: Autoria própria.

2.4 *Framework* Rasa Open Source

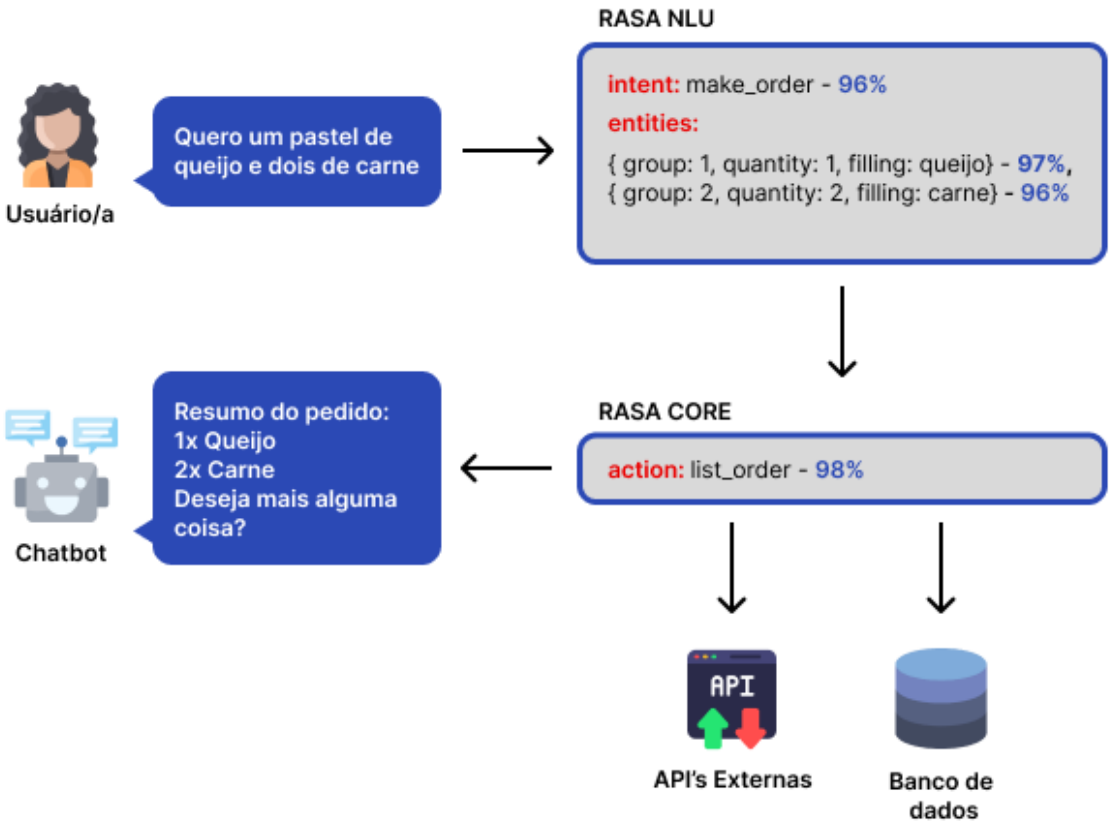
O Rasa Open Source é uma plataforma de código aberto que fornece os componentes necessários para a criação de assistentes virtuais ou *chatbots*. Utilizando Inteligência Artificial (IA), ele permite entender e manter conversas além de conectar-se a canais de mensagens e sistemas de terceiros por meio de um conjunto de APIs (RASA, 2022).

O *framework* é composto por dois componentes principais: Rasa NLU e Rasa Core. O componente Rasa NLU é responsável por interpretar as mensagens do usuário para obter a intenção e extrair entidades disponíveis na referida mensagem através de configurações definidas na chamada *pipelines*. Por outro lado, o Rasa Core é uma solução de gerenciamento de diálogo baseado em um modelo de probabilidade para decidir a resposta mais acertada em função das

mensagens anteriores do usuário. Ele usa modelos de aprendizado de máquina para treinar conversas para decidir o que fazer a seguir, o que no contexto do Rasa é chamado de *actions*.

A Figura 2 ilustra os principais componentes do Rasa, onde o Rasa NLU é responsável pela interpretação das mensagens do usuário e o Rasa Core decide a próxima ação a ser tomada.

Figura 2 – Exemplo de funcionamento dos componentes principais do Rasa



Fonte: Autoria própria.

2.4.1 Rasa NLU

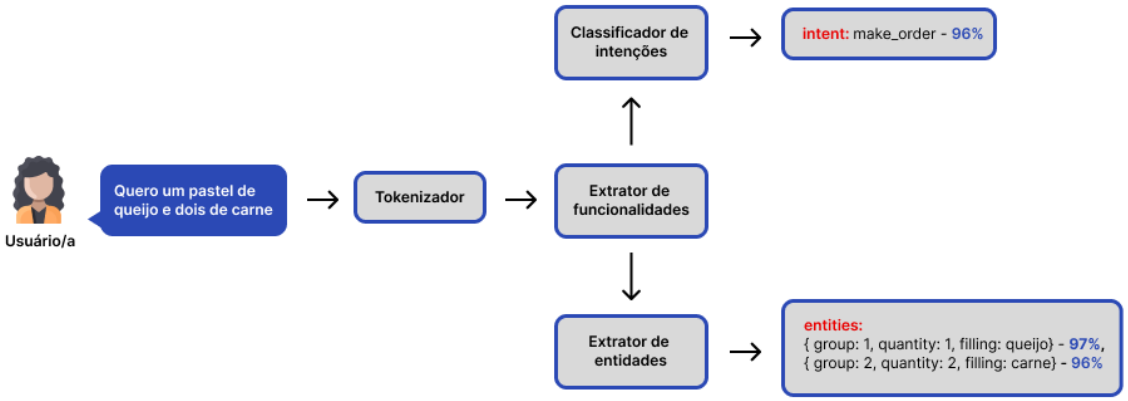
Ao projetar um assistente com Rasa, o processo chamado de *NLU Pipeline* deve ser definido. Este processo consiste em adicionar diferentes componentes pré-configurados para converter mensagens não estruturadas enviadas pelo usuário em uma saída estruturada em intenções e entidades. Os componentes que são adicionados ao processo de classificação de intenções e extração de entidades são os seguintes:

- Tokenizador.
- Extrator de características.

- Classificador de intenções.
- Extrator de entidades.

A Figura 3 mostra o processo pelo qual a mensagem é processada pelo Rasa NLU.

Figura 3 – Exemplo de funcionamento do componente Rasa NLU



Fonte: Autoria própria.

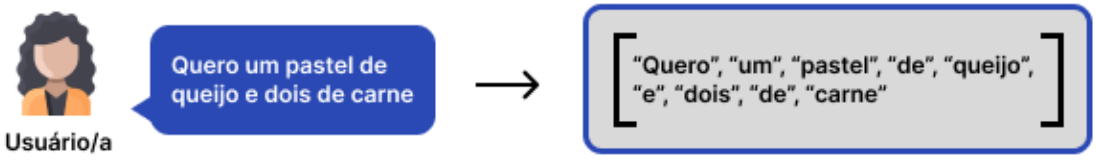
A seguir, é descrita a funcionalidade de cada componente dentro do processo de análise de mensagem do Rasa NLU.

2.4.1.1 Tokenizador

A primeira etapa do processo de análise das mensagens do usuário é a tokenização. Este componente é responsável por dividir a mensagem em pequenos fragmentos de texto, conhecidos como *tokens*. Cada *token* pode ser obtido através de uma técnica ou ferramenta de tokenização. Neste trabalho foi utilizado a técnica WhiteSpaceTokenizer, que separa palavras identificando espaços em branco (RASA, 2022).

A Figura 4 apresenta o resultado da técnica WhiteSpaceTokenizer para obter um lista de palavras da mensagem. Os tokens resultantes são utilizados para a extração de características.

Figura 4 – Exemplo de funcionamento do componente de Tokenização



Fonte: Autoria própria.

2.4.1.2 Extrator de características

O componente de extração de características, ou também conhecida como *Featurizer*, realiza o processo de transformar *tokens* (palavras) em números ou vetores representativos. O Rasa oferece dois tipos de *featurizers*: densos e esparços, podendo ser combinados entre si. Nesse projeto foram utilizados apenas *featurizers* esparços, pois segundo Rasa (2022) os *featurizers* esparsos armazenam apenas os valores diferentes de zero e suas posições no vetor e não os valores inválidos retornados pelo *featurizer*. Assim, economiza-se muita memória e pode-se treinar em conjuntos de dados maiores.

Para o contexto do projeto, foi utilizado o *featurizer* CountVectorsFeaturizer.

2.4.1.2.1 CountVectorsFeaturizer

CountVectorizer é uma técnica utilizada para transformar um determinado texto em uma forma vetorial com base na frequência que cada palavra ocorre em todo o texto (PEDREGOSA, 2019). Para isso, é criada uma matriz onde cada palavra única é representada por uma coluna da matriz e cada amostra de texto do documento é uma linha na matriz. O valor de cada célula é a frequência daquela palavra na amostra de texto informada. Na Listagem 1 é apresentado o seguinte dicionário de entrada, como exemplo:

Listagem 1 – Exemplo de dicionário para CountVectorizer

```
1 input = ["Quero um pastel de frango",
2          "Vou pedir um pastel de queijo e um de frango"]
```

Fonte: Autoria própria (2023).

Esse dicionário resulta no Quadro 1, após ser processado pelo CountVectorizer.

Quadro 1 – Exemplo de resultado de dados processados pelo CountVectorizer

	de	**e**	**frango**	**pastel**	**pedir**	**queijo**	**quero**	**um**	**vou**
input[0]	1	0	1	1	0	0	1	1	0
input[1]	2	1	1	1	1	1	0	2	1

Fonte: Autoria própria (2023).

Todas as características extraídas serão posteriormente alimentadas em um classificador de intenção/extrator de entidades.

2.4.1.3 Classificador de intenções e Extrator de entidades

Após a extração das funcionalidades dos *tokens*, elas passam para um modelo de classificação de intenção. Rasa recomenda usar o modelo criado pela própria organização chamado

de *Dual Intent Entity Transformer* - Transformador Duplo de Intenção e Entidade (DIET), um modelo capaz de extrair fragmentos de informações (entidades) da sentença e reconhecer intenções ao mesmo tempo. Utilizando uma arquitetura multitarefa, o modelo DIET é um tipo de rede neural *Transformer*.[1] Ele também fornece a capacidade de se conectar a várias redes neurais pré-treinadas, como BERT, GloVe, ConveRT e assim por diante (BUNK *et al.*, 2020). De acordo com Jiao (2020) a arquitetura DIET tem uma maior acurácia na extração de entidades, comparadas com um método de Rede Neural Recorrente[2].

2.4.2 Rasa Core

Rasa utiliza políticas em sua arquitetura para decidir qual ação o assistente deve tomar em todos os momentos da conversa. Existem políticas complexas baseadas em aprendizado de máquina e que levam em consideração o contexto da conversa, prevendo a próxima ação a ser tomada pelo *chatbot* com certo grau de confiança. Existem também políticas mais simples que usam apenas regras. Essas políticas podem também ser combinados entre si, se necessário. A seguir são apresentadas as características de cada política utilizada no projeto, segundo a documentação do Rasa[3]:

2.4.2.1 RulePolicy

A RulePolicy é uma política simples baseada em regras, que lida com as partes da conversa que seguem um comportamento fixo (por exemplo, regra de negócios). Ele faz previsões com base em quaisquer regras presentes nos dados de treinamento. Por exemplo, pode ser definida uma regra para que sempre que o usuário questione se ele está conversando com um humano, o *chatbot* responda que ele é um assistente virtual, independente do contexto que a conversa estava.

2.4.2.2 MemoizationPolicy

A MemoizationPolicy é uma política que utiliza aprendizado de máquina e tem como principal característica lembrar as histórias ou *stories* definidas nos dados de treinamento. As *stories* são os caminhos de conversação de exemplo que foram detalhados nos dados de trei-

[1] *Transfomers* são modelos de aprendizado de máquina baseados na auto-atenção, o que possibilita a aprendizagem de acordo com contextos. No cenário de NLP por exemplo, eles possibilitam uma análise semântica de palavras em uma frase, de acordo com o histórico da conversa (SINGH; MAHMOOD, 2021).

[2] Redes Neurais Recorrentes (*Recurrent Neural Network RNN*) são um grupo de redes neurais para processamento de dados sequenciais. No contexto de NLP, as RNN's entendem o texto como uma sequência de palavras e visam capturar dependências entre palavras e estruturas de texto (MINAEE *et al.*, 2020).

[3] <https://rasa.com/docs/rasa/policies> - Acesso em 1 dez. 2022.

namento e essa política garante que o assistente siga esses caminhos durante a conversa. Ela verifica se a discussão atual é semelhante a uma *story* nos dados de treinamento. Supondo que seja esse o caso, ela selecionará a ação correspondente à *story* definida nos dados de treinamento com um grau de confiança de 1.0. Caso nenhuma *story* de treinamento semelhante seja descoberta, a política retornará uma ação indefinida com grau de confiança 0.0.

2.4.2.3 TEDPolicy

A TEDPolicy (*Transformer Embedding Dialogue*) é uma arquitetura multitarefa para previsão da próxima ação a ser executada pelo *chatbot*. É uma política com base no aprendizado de máquina, que possui dentro de sua arquitetura uma rede neural do tipo *Transformer* e foi escolhida pois segundo Vlasov, Mosig e Nichol (2019) a TEDPolicy funciona particularmente bem em conversas não lineares, onde o usuário intervém com uma mensagem fora do tópico ou volta para modificar uma declaração anterior. Esses tipos de diálogos de vários turnos refletem a maneira como os usuários realmente falam e também são os tipos de diálogos particularmente complexos para tentar modelar com um conjunto de regras.

2.4.3 Critérios de avaliação

Os *chatbots* podem ser avaliados com relação ao desempenho das intenções classificadas. Para realizar essas avaliações pode-se utilizar as métricas comumente usadas em aprendizado de máquina para avaliar classificadores, entre as quais destaca-se: Acurácia, Precisão, Revocação (*Recall*) e F1-score (GRANDINI; BAGLI; VISANI, 2020). Para calcular essas métricas são necessários os valores de Verdadeiro Positivo (*True Positive* TP), Verdadeiro Negativo (*True Negative* TN), Falso Positivo (*False Positive* FP) e Falso Negativo (*False Negative* FN). Um *True Positive* (TP) é um caso classificado corretamente como verdadeiro. Da mesma forma, um TN também é um caso classificado corretamente, mas como Falso. Os erros de classificação então podem ser FP (um caso classificado como positivo que deveria ter sido classificado como negativo) e FN (um caso classificado como negativo que deveria ter classificado como positivo).

As métricas são baseadas na matriz de confusão, pois ela contém todas as informações relevantes sobre o algoritmo e desempenho da regra de classificação. A matriz de confusão é uma tabela cruzada que registra o número de ocorrências entre dois avaliadores, a classificação real e a classificação prevista, conforme mostrado na Figura 5. As classes são listadas na mesma ordem nas linhas e nas colunas, portanto os elementos corretamente classificados são localizados na diagonal principal da matriz (GRANDINI; BAGLI; VISANI, 2020).

Figura 5 – Exemplo de matriz de confusão

		Valor Verdadeiro	
		Classe Positiva	Classe Negativa
Valor previsto	Classe Positiva	**VP** Verdadeiro Positivo	**FP** Falso Positivo
	Classe Negativa	**FN** Falso Negativo	**VN** Verdadeiro Negativo

Fonte: Silva (2018).

2.4.3.1 Acurácia

O parâmetro de Acurácia é usado como medida de desempenho que mostra a taxa de acurácia de previsão criada pelo modelo. Esse parâmetro pode ser calculado conforme a equação 1:

$$Acurácia = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

2.4.3.2 Precisão

O parâmetro de Precisão é utilizado como a taxa de precisão dos resultados das amostras classificadas pelo modelo, informando quantas realmente pertencem à classe prevista. Esse parâmetro pode ser calculado conforme a equação 2:

$$Precisão = \frac{TP}{TP + FP} \tag{2}$$

2.4.3.3 *Recall*

O parâmetro de *Recall* ou Revocação, mostra a taxa de sucesso do sistema para reconhecer uma categoria. Esse parâmetro pode ser calculado conforme a equação 3:

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

2.4.3.4 *F1-score*

Essa métrica relaciona a precisão e o *recall* de modo a trazer um número único que indique a qualidade geral do modelo. Ele pode ser calculado conforme a equação 4:

$$F1 = \frac{2 * Precisão * Recall}{Precisão + Recall} \tag{4}$$

3 METODOLOGIA

Este projeto foi desenvolvido usando um compilado da adaptação do método *Cross Industry Standard Process for Data Mining - Processo Padrão Inter-Indústrias para Mineração de Dados* (CRISP-DM) proposta por Studer *et al.* (2020): "O método é um conjunto de processos ou sistemas de trabalho que são usados como diretrizes para a criação de um projeto completo de Inteligência Artificial e Aprendizado de Máquina". Esta metodologia consiste em seis fases principais:

- Entendimento do negócio e dos dados.
- Preparação dos dados.
- Modelagem.
- Avaliação.
- *Deployment*.
- Monitoramento e manutenção.

3.1 Entendimento do negócio e dos dados

O entendimento das necessidades enfrentadas por restaurantes se tornou objeto de análise após o desenvolvedor do projeto passar a frequentar os bastidores de um restaurante de um familiar. Essas necessidades se agravaram na época de pandemia de COVID-19, quando os estabelecimentos estavam impedidos de receber clientes presencialmente. Com isso, só havia uma forma de continuar a operação: receber pedidos *on-line*. Dessa maneira, foi definido que o desenvolvimento de um *chatbot* poderia auxiliar nesse fluxo de recebimento de pedidos - que chegavam via aplicativos de mensagens instantâneas, pois analisando os dados das conversas existentes, era possível identicar um fluxo relativamente bem definido. Dessa forma, foram definidos os requisitos mínimos para o *chatbot* atender os objetivos do negócio:

1. Identificar uma saudação e iniciar o fluxo do pedido.
2. Fornecer o cardápio ao cliente, quando solicitado.
3. Identificar a quantidade e sabor de um ou mais itens solicitados pelo cliente (com o limite de 2 itens por cada mensagem).
4. Calcular o valor total do pedido.
5. Identificar a forma de entrega do pedido: *delivery* ou retirada no balcão.

6. Identificar e validar o endereço de entrega e seus componentes como número e complemento, no caso de *delivery*.

7. Identificar o meio de pagamento solicitado pelo cliente.

8. Finalizar o pedido.

O processo de desenvolvimento de um *chatbot* contextual utilizando aprendizado de máquina, exige uma série de conjuntos de dados para treinar o modelo do *chatbot*. O conjunto de dados foi obtido de conversas reais fornecidas voluntariamente pelo restaurante em questão. Dentre os requisitos definidos, foram mapeadas e classificadas as intenções que o *chatbot* deveria reconhecer, com base no *dataset* de conversas. Essas intenções estão presentes na Tabela 1, totalizando 205 exemplos de mensagens e 13 intenções.

Tabela 1 – Classificação de intenções utilizadas no treinamento

Nº	Tipo de Intenção	Descrição	Nº de exemplos
1	*greet*	Saudação do cliente	19
2	*menu request*	Solicitação do cardápio	21
3	*order request*	Solicitação de um ou mais itens do cardápio	56
4	*delivery request*	Solicitação de *delivery*	8
5	*pick up request*	Solicitação de retirada no balcão	8
6	*delivery address*	Informação do endereço de entrega	24
7	*confirm address*	Confirmação do endereço	12
8	*reject*	Intenção usada para negações	14
9	*set street number*	Informação do número do endereço	9
10	*set complement*	Informação do complemento do endereço	9
11	*wrong address request*	Solicitação de correção do endereço	7
12	*set payment type*	Informação do meio de pagamento	15
13	*thanks*	Agradecimento	10

Fonte: Autoria própria (2023).

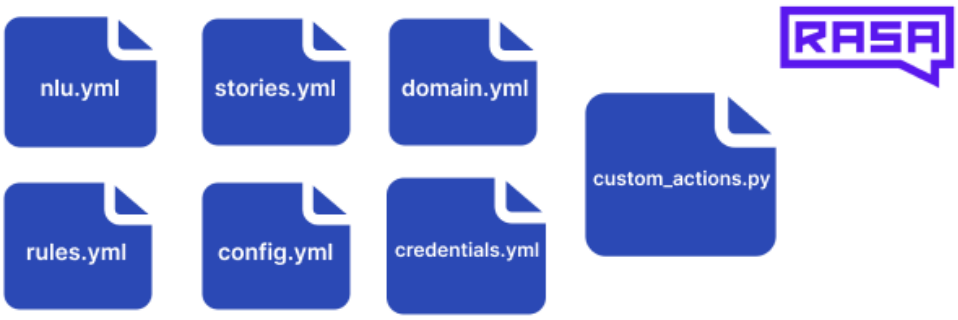
3.2 Preparação dos dados

Um assistente conversacional construído usando o *Framework* Rasa Open-Source requer vários arquivos de configuração, assim como um conjunto formatado de dados de treinamento. A Figura 6 mostra a estrutura de arquivos que foram configurados para o treinamento dos dados.

A explicação de cada arquivo de configuração é apresentada a seguir, segundo Rasa (2022).

1. *Natural Language Understanding* (NLU): Os dados do arquivo nlu.yml contém as mensagens dos usuários retiradas do *dataset* original e rotulados com a intenção, combinados com a adição de entidades, sinônimos e Tabelas de pesquisa ou *Lookup Tables*.

Figura 6 – Representação esquemática da estrutura de arquivos do Rasa



Fonte: Autoria própria (2023).

2. *Stories*: O arquivo stories.yml contém os exemplos de fluxos de conversas que o *chatbot* deve seguir durante a interação com o usuário. Nele é possível definir vários caminhos que a conversa deve tomar de acordo com o que o usuário envia, incluindo *happy paths* e *unhappy paths*[1]. No caso de um fluxo de um pedido de comida, essa configuração é essencial pois nela foi possível definir exatamente qual caminho a conversa deveria tomar desde a saudação até a finalização do pedido. A Figura 7 mostra o exemplo de uma *story* do projeto:

3. *Domain*: Os dados do arquivo domain.yml contém o universo de conversação como a definição das intenções, entidades, *slots*[2] e *actions*. As definições de ações são utilizadas como modelo de resposta para o *chatbot*.

4. *Rules*: No arquivo rules.yml é possível definir as regras que descrevem pequenos pedaços de conversas que devem sempre seguir o mesmo caminho. A Figura 8 mostra um exemplo de uma *rule* definida no projeto. Ela define que toda vez que for identificada a *intent* "*thanks*", o *chatbot* deve responder com a *action* "*utter_youre_welcome*".

5. *Config*: Os dados do arquivo config.yml contém as configurações de idioma, *pipeline* e configurações das políticas ou *policies* que são usadas para treinar o modelo. Essa etapa será discutida com mais detalhes na seção 3.3.

6. *Credentials*: No arquivo credentials.yml é possível configurar conectores externos para serem usados em conjunto com o *chatbot*. No projeto foram utilizados os conectores do

[1] Neste contexto, o termo *happy path* significa que, ao final do fluxo de conversação, o usuário tem seu pedido compreendido com sucesso. Desta forma, o termo *unhappy paths* significa que, ao final do fluxo, ocorreu algum problema que impediu a conclusão da conversação com sucesso.

[2] *Slots* são a memória do *chatbot*. Eles funcionam como um armazenamento de valor-chave que pode ser usado para armazenar informações fornecidas pelo usuário (por exemplo, seu pedido), bem como informações coletadas externamente (por exemplo, o resultado da busca pelo endereço do cliente) (RASA, 2022).

Figura 7 – Exemplo de *story* no Rasa

```
- story: interactive_story
  steps:
  - intent: greet
  - action: action_greet
  - intent: menu_request
  - action: action_list_menu
  - intent: order_request
    entities:
    - food_quantity: dois
    - filling: pizza
    - food_quantity: 1
    - filling: carne
  - action: action_utter_list_order
  - slot_was_set:
    - order:
      - food_quantity: dois
      - filling: pizza
      - food_quantity: 1
      - filling: carne
  - intent: reject
  - action: utter_delivery_type
  - intent: pickup_request
    entities:
    - delivery_type: retirada no local
  - action: utter_payment_type
  - intent: set_payment_type
    entities:
    - payment_type: cartao
  - action: action_confirm_order
```

Fonte: Autoria própria (2023).

aplicativo de mensagens Telegram para realizar a interface com o usuário e o conector de banco de dados, que é responsável por armazenar as conversas da fase de testes, para a posterior avaliação do modelo. O banco de dados contém uma única tabela que armazena o identificador da conversa, a mensagem enviada, as entidades e intenções classificadas e a ação (resposta) predita pelo modelo.

7. *Custom Actions*: Utilizando a linguagem Python, no arquivo custom_actions.py é possível definir customizações para as ações que o *chatbot* pode tomar. Por exemplo, para a *action* que trata a *intent* "order_request", foi implementado um método para calcular o valor total do pedido com base nos itens que já foram adicionados e estão armazenados na memória do *chatbot*. Com isso é possível montar a mensagem com o resumo e o valor do pedido. Nesse arquivo também foi definida a customização da *action* que trata a *intent* "delivery_address", na qual é realizada a integração com a API do Google

Figura 8 – Exemplo de *rule* no Rasa

Fonte: Autoria própria (2023).

Maps para a validação do endereço informado pelo cliente em um pedido para entrega. Para isso é feita uma requisição HTTP na API enviando como parâmetro o endereço fornecido pelo cliente, obtendo como resposta as informações detalhadas do local de entrega, como código postal, logradouro e coordenadas geográficas.

3.3 Modelagem

A escolha das técnicas e algoritmos utilizadas pelo Rasa para compreender a mensagem de entrada do usuário e determinar a resposta ou saída adequada, depende de três configurações principais, a escolha do idioma, *NLU pipeline* e políticas. A Figura 9 apresenta as configurações usadas no projeto.

O *framework* Rasa possui suporte para o idioma português, sendo esse o idioma escolhido. O modelo de *pipeline* consistiu no processo de tokenização, extração de características, classificação de intenção, extração de entidade e seletores de respostas - itens discutidos nas seções 2.4.1 e 2.4.2.

Além disso, as *Policies* visam definir o processo de gestão do diálogo e os modelos usados para definir as respostas com base na entrada da mensagem do usuário. Diferente do procedimento de *Pipeline* que processa cada componente em sequência, as *Policies* são executadas em paralelo. Cada *policy* definida na configuração irá prever uma próxima ação com um determinado nível de confiança. A *policy* que obtiver o maior grau de confiança decide a próxima ação a ser executada pelo *chatbot* (RASA, 2022).

Ademais foi configurado o número de épocas (*epochs*) para cada componente, que representa o número de vezes que o algoritmo vai recorrer aos dados de treinamento. Uma época consiste em uma passagem para frente e uma passagem para trás em todos os exemplos de treinamento (RASA, 2022).

Figura 9 – Arquivo de configuração do Rasa dos modelos utilizados no projeto

```yaml
language: pt

pipeline:
  - name: WhitespaceTokenizer
  - name: RegexFeaturizer
    "case_sensitive": False
  - name: CountVectorsFeaturizer
  - name: DIETClassifier
    epochs: 100
    constrain_similarities: True

policies:
  - name: MemoizationPolicy
  - name: TEDPolicy
    max_history: 20
    epochs: 200
  - name: RulePolicy
    max_history: 10
    epochs: 100
```

Fonte: Autoria própria (2023).

3.4 Avaliação

O *framework* Rasa também provê recursos para a avaliação das intenções classificadas pelos modelos de aprendizado de máquina configurados, produzindo automaticamente um relatório com as métricas discutidas na seção 2.4.3 e a matriz de confusão para a tarefa de classificação de intenções.

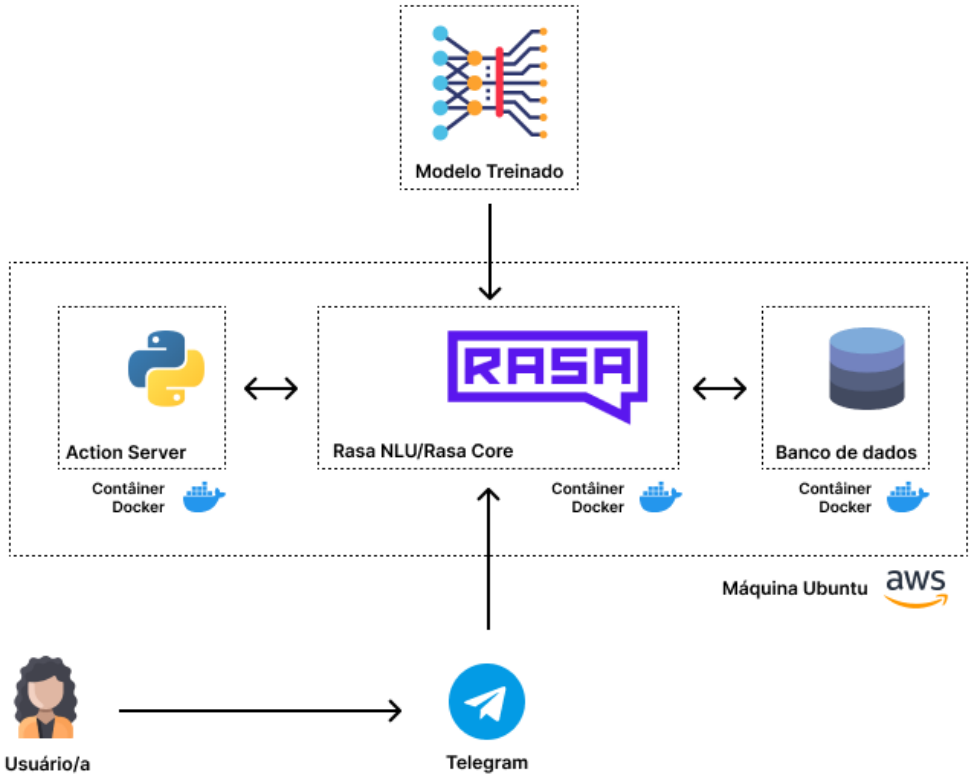
Para realizar a avaliação, foram geradas *stories* a partir dos dados levantados nos testes realizados por usuários voluntários, que foram todas armazenadas em um banco de dados. Estas *stories* são compostas pelas mensagem enviadas pelo usuário e a classificação manual de qual deveria ser a intenção identificada para aquela mensagem. Com base nisso, foi possível executar os testes automatizados providos pelo Rasa e gerar as métricas referentes ao classificador de inteções.

3.5 *Deployment*

Para o processo de *deployment* foi utilizada a estratégia de conteinerização[3]

Essa estratégia foi utilizada pois segundo Fernandez (2018) contêineres são rápidos, eficientes e compactos, através de suas funcionalidades e benefícios, trabalhando com múltiplas tarefas, além de poder ser reiniciados rapidamente, em casos de falhas. Para isso foi utilizado a plataforma Docker[4], em conjunto com infraestrutura baseada em nuvem, provida pela *Amazon Web Services*. A Figura 10 apresenta o esquema da arquitetura da infraestrutura do sistema.

Figura 10 – Esquema de arquitetura da infraestrutura do projeto



Fonte: Autoria própria (2023).

A fase de configuração e treinamento dos modelos de aprendizado de máquina foi concluída usando computadores locais e posteriormente enviados ao servidor, que utiliza o sistema operacional Ubuntu. Com o modelo treinado disponível, o Rasa se comunica diretamente com

[3] Contêineres contém todo o kit necessário para uma aplicação, de modo que a aplicação possa ser executada de forma isolada. Os contêineres geralmente são comparados às Máquinas Virtuais (VM) No caso de plataformas VM existentes, todo um sistema operacional convidado é virtualizado em um sistema operacional host. Este método tem a vantagem de poder virtualizar e usar vários SOs no sistema operacional host e relativamente simples de usar, mas há é uma limitação que é difícil de usar em ambientes de produção devido à desvantagem de ser relativamente pesado e lento (RAD; BHATTI; AHMADI, 2017).

[4] Docker é uma plataforma de código aberto para desenvolvimento, implantação e execução de aplicativos em contêiner (KWON; LEE, 2020).

um servidor em linguagem de programação Python para processar as ações customizadas, sendo possível realizar comunicações com API's externas. A integração com o banco de dados também é realizada, para armazenar os dados das conversas geradas no período de testes. Por fim, há a conexão com o aplicativo de mensagens Telegram que é facilmente integrado através do sistema de conectores externos do Rasa, que utiliza a tecnologia de *webhooks*[5].

3.6 Monitoramento e manutenção

A etapa de monitoramento e manutenção, está fora do escopo desse projeto. Porém uma estratégia que poderia ser adotada em trabalhos futuros é a de coletar e classificar os dados das conversas realizadas e gerar um novo modelo ou realizar um ajuste fino no modelo já existente (STUDER *et al.*, 2020).

Utilizando o *framework* Rasa, pode-se armazenar as mensagens enviadas e recebidas pelo *chatbot* definindo um *Tracker Store*, que é um local onde as mensagens podem ser armazenadas. Neste projeto foi utilizado como *Tracker Store* um banco de dados, de onde foi possível extrair as mensagens enviadas e recebidas durante os testes.

[5] *Webhook* é uma notificação de um evento enviada para uma determinado URL. Eles são úteis para comunicações entre aplicações, utilizando o protocolo HTTP (SITTAKUL; JUDPRASONG; SAENGMANEE, 2019).

4 RESULTADOS

Neste capítulo serão apresentados os resultados obtidos pelo projeto, incluindo seu escopo, limitações, demonstrações de uso do *chatbot* e a avaliação do modelo obtida através dos testes.

4.1 Escopo do sistema

O *chatbot* desenvolvido utilizando o *framework* Rasa deve ser capaz de conduzir o fluxo de conversas de pedidos em um restaurante, via mensagens de texto instantâneas recebidas e enviadas pelo aplicativo Telegram. Os clientes poderão solicitar que o pedido seja feito para entrega ou retirada e escolher também a forma de pagamento.

Inicialmente, com exceção do subfluxo de captura do endereço, o *chatbot* não permite a edição e remoção de informações já armazenadas pelo sistema, como por exemplo itens do pedido.

Para qualquer tema que fuja do escopo do fluxo definido na seção 4.2, o *chatbot* deve responder informando que não compreendeu a mensagem enviada, requisitando que o cliente se expresse de outra forma ou volte ao fluxo do pedido. Para que isso aconteça, foi definida uma *rule* para que quando o modelo classificar uma intenção com menos de 70% de nível de confiança, essa resposta padrão deve ser enviada.

4.2 Modelagem do sistema

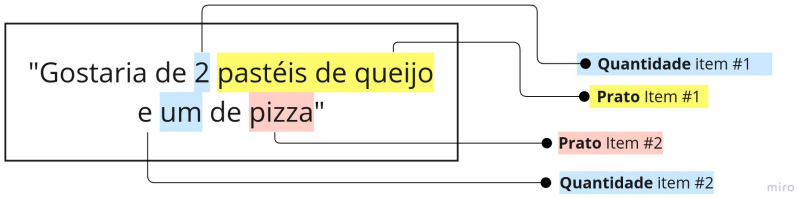
A modelagem do fluxo de conversa do pedido foi definido com base nas conversas analisadas e os requisitos mínimos que o *chatbot* deveria cumprir. A sequência dos passos é explicitada na lista de itens apresentados a seguir:

1. O *chatbot*, que nesse projeto foi denominado "Pep", deve ser capaz de entender quando a primeira mensagem é uma saudação e deverá responder à saudação com o nome do contato do cliente extraído dos metadados da mensagem, providos pelo *Telegram.* A resposta deve conter também uma pergunta questionando o que cliente deseja e como ele pode ajudá-lo. Nesse momento da conversa, ele deve ser capaz de responder aos seguintes interesses e dúvidas do cliente:
    - Solicitação do cardápio.
    - Realização de um pedido.
2. Ao receber uma solicitação de pedido ou cardápio do estabelecimento, o *chatbot* deve enviar os itens disponíveis e seus respectivos preços em forma de lista para o cliente. Junto com os itens do cardápio ele deve enviar uma instrução ao usuário requisitando

quais os itens ele deseja e em qual quantidade. O estabelecimento fictício utilizado no projeto foi o de uma pastelaria, chamada de Rei do Pastel - que tem 5 sabores de pastéis em seu cardápio: queijo, carne, frango, pizza e camarão.

3. Após o recebimento dos itens do pedido, o *chatbot* deve ser capaz de identificar na mensagem o que é referente ao prato e qual a quantidade desejada, como na Figura 11.

Figura 11 – Exemplo de mensagem com identificação de itens e quantidades



Fonte: Autoria própria.

Com a extração dessas informações o *chatbot* deve armazenar em memória (*slots)* o pedido com seus referentes itens e enviar uma mensagem com o resumo do pedido atual, com os itens selecionados e o valor total calculado.

4. Posteriormente ao armazenamento de um pedido, o *chatbot* deve perguntar se o cliente deseja mais algum item do cardápio. Se a resposta for positiva, ele deve retornar ao item 2. Se for negativa, o fluxo deve seguir para o item 5 para a finalização do pedido.

5. Caso o cliente deseje finalizar o pedido, o *chatbot* deve enviar uma mensagem perguntando a forma de entrega, com as opções disponíveis. Nesse projeto, serão definidas as opções *delivery* e retirada. Caso o cliente escolha pela opção *delivery*, o *chatbot* deve seguir para o item 6. Se a opção for para retirada, ele deve seguir para o item 8.

6. Ao receber a opção de entrega *delivery*, o *chatbot* deve perguntar ao cliente qual seu endereço de entrega. O endereço pode ser informado em três formatos diferentes: "logradouro", "logradouro e número", "logradouro, número e complemento". Caso o cliente não informe o número ou complemento inicialmente, o *chatbot* deve perguntar por essas informações separadamente.

7. Ao receber o endereço de entrega do cliente, o *chatbot* deve verificar se o endereço informado é válido, utilizando a API do Google Maps. Para realizar essa validação, é informado o endereço enviado pelo cliente como parâmetro da requisição e a API retorna todos os detalhes daquele endereço. Caso ele não exista, o *chatbot* deve pedir que o cliente informe o endereço novamente.

8. Nesse passo, o *chatbot* deve enviar para o cliente as opções de pagamento disponíveis, perguntando com qual ele deseja realizar o pagamento. Após receber a resposta, ele deve armazenar essa informação em memória com os dados do pedido.

9. O último passo, é o *chatbot* informar o resumo do pedido ao cliente com todos os itens solicitados, forma de entrega e forma de pagamento juntamente com uma mensagem de agradecimento.

A Figura 12 apresenta um fluxograma que representa os passos descritos acima, onde o lado esquerdo se refere às intenções do usuário e o lado direito às ações tomadas pelo *chatbot*.

4.3 Apresentação do sistema

Nesta seção serão apresentados alguns exemplos de funcionamento do *chatbot* desenvolvido, explorando diferentes caminhos e formas de realizar os pedidos. A Figura 13 apresenta o exemplo de um fluxo de um pedido para *delivery*.

Já a Figura 14 apresenta um exemplo de funcionamento para um pedido feito para retirada no local.

4.4 Avaliação e discussões

Esta seção detalha e analisa os resultados obtidos nos diferentes testes que foram feitos para o assistente desenvolvido. O tamanho do *dataset* de treinamento inicial tem cerca de 205 exemplos divididos em cada uma das intenções propostas (13 intenções). Esses resultados foram obtidos através de um experimento simulado com 10 voluntários que foram selecionados entre colegas do curso de Engenharia de Computação, familiares e amigos do proponente deste projeto, gerando 20 interações no total.

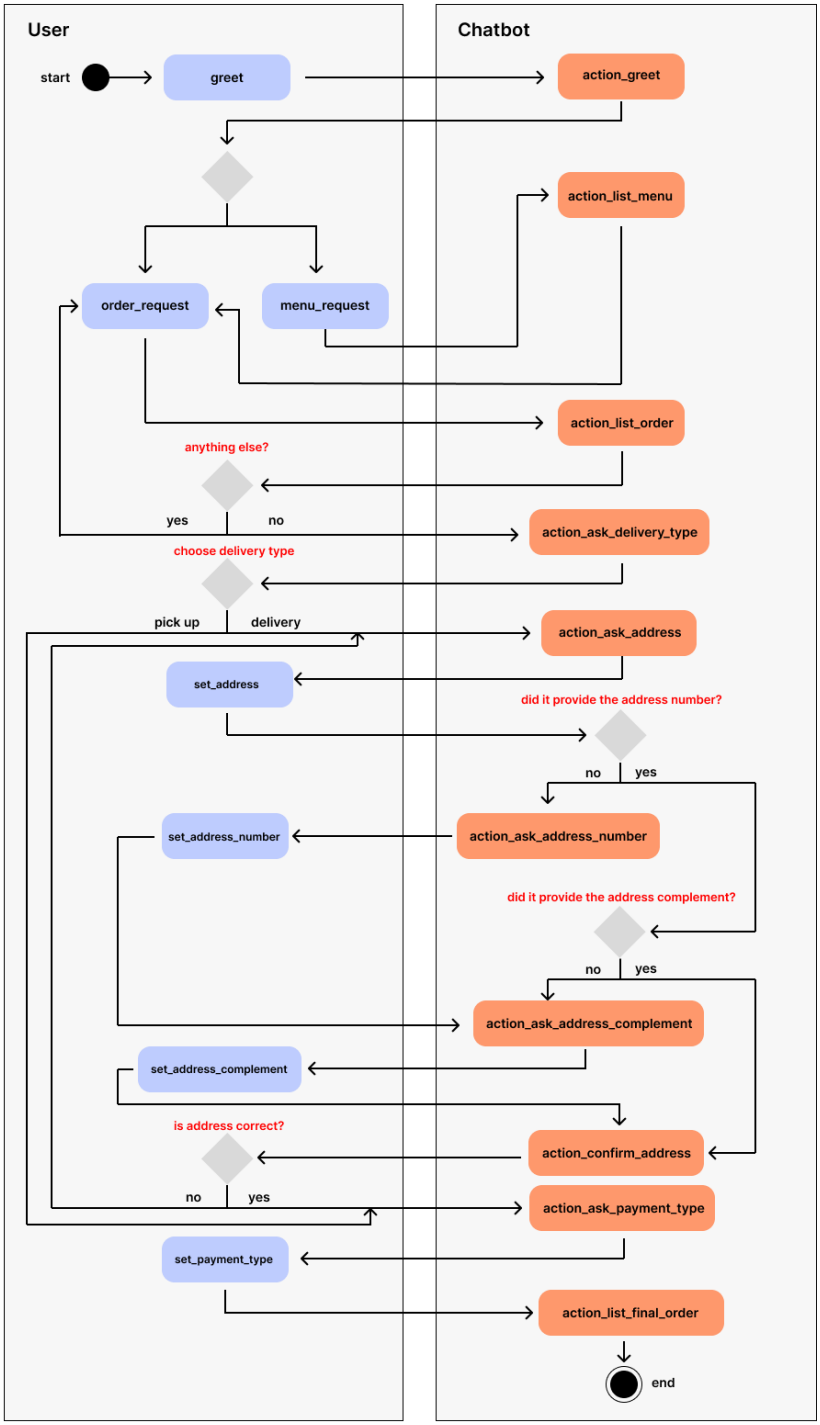
A Figura 15 mostra a matriz de confusão do modelo para a classificação de intenções.

A partir dos dados da matriz, pode-se calcular as métricas de avaliação definidas na seção 2.4.3. A Tabela 2 mostra os resultados das métricas para cada intenção mapeada.

Ao decorrer do desenvolvimento e aprofundamento no conhecimento do *framework* Rasa, identificou-se a possiblidade de utilizar botões (como é possível observar nas Figuras 13 e 14) para facilitar a interação com o usuário para perguntas fechadas e com respostas com alternativas. Como é o caso das perguntas que geram as intenções *"confirm address"*, *"delivery request"*, *"set payment type"*, *"pickup request"* e *"wrong address request"*. Isso explica os resultados com 100% de precisão para esses casos.

Em contrapartida, as intenções de *"delivery address"*, "*"set complement"* e *"set street number"*, tiveram precisões relativamente mais baixas que outras intenções. Alguns fatores explicam esse menor desempenho, como o menor número de exemplos presentes no *dataset* e

Figura 12 – Fluxograma do fluxo de pedidos



Fonte: Autoria própria.

a variedade de formas diferentes de informar um endereço. Na Figura 16 é possível observar alguns exemplos de um erros de classificação para a *intent* "delivery address".

Por fim, as métricas globais do modelo de classificação de intenções avaliado são dadas nas equações 5, 6, 7 e 8:

$$Acurácia = \frac{264}{280} = 0{,}94 \tag{5}$$

Figura 13 – Funcionamento do *chatbot* em um pedido de *delivery*

Fonte: Autoria própria.

Tabela 2 – Resultado da extração de intenções

Nº	Tipo de Intenção	Acurácia	Precisão	*Recall*
1	*confirm address*	1.00	1.00	1.00
2	*delivery address*	0.80	0.89	0.80
3	*delivery request*	1.00	1.00	1.00
4	*food request*	0.95	1.00	1.00
5	*greet*	0.90	0.9	0.90
6	*menu request*	1.00	1.00	1.00
7	*reject*	1.00	1.00	1.00
8	*set complement*	0.85	0.74	0.85
9	*set payment type*	1.00	1.00	1.00
10	*set street number*	0.85	0.89	0.85
11	*thanks*	0.95	0.95	0.95
12	*pickup request*	1.00	1.00	1.00
13	*wrong address request*	1.00	1.00	1.00

Fonte: Autoria própria (2023).

$$Precisão = \frac{264}{280} = 0{,}94 \tag{6}$$

$$Recall = \frac{264}{277} = 0{,}95 \tag{7}$$

Figura 14 – Funcionamento do *chatbot* em um pedido para retirada no local

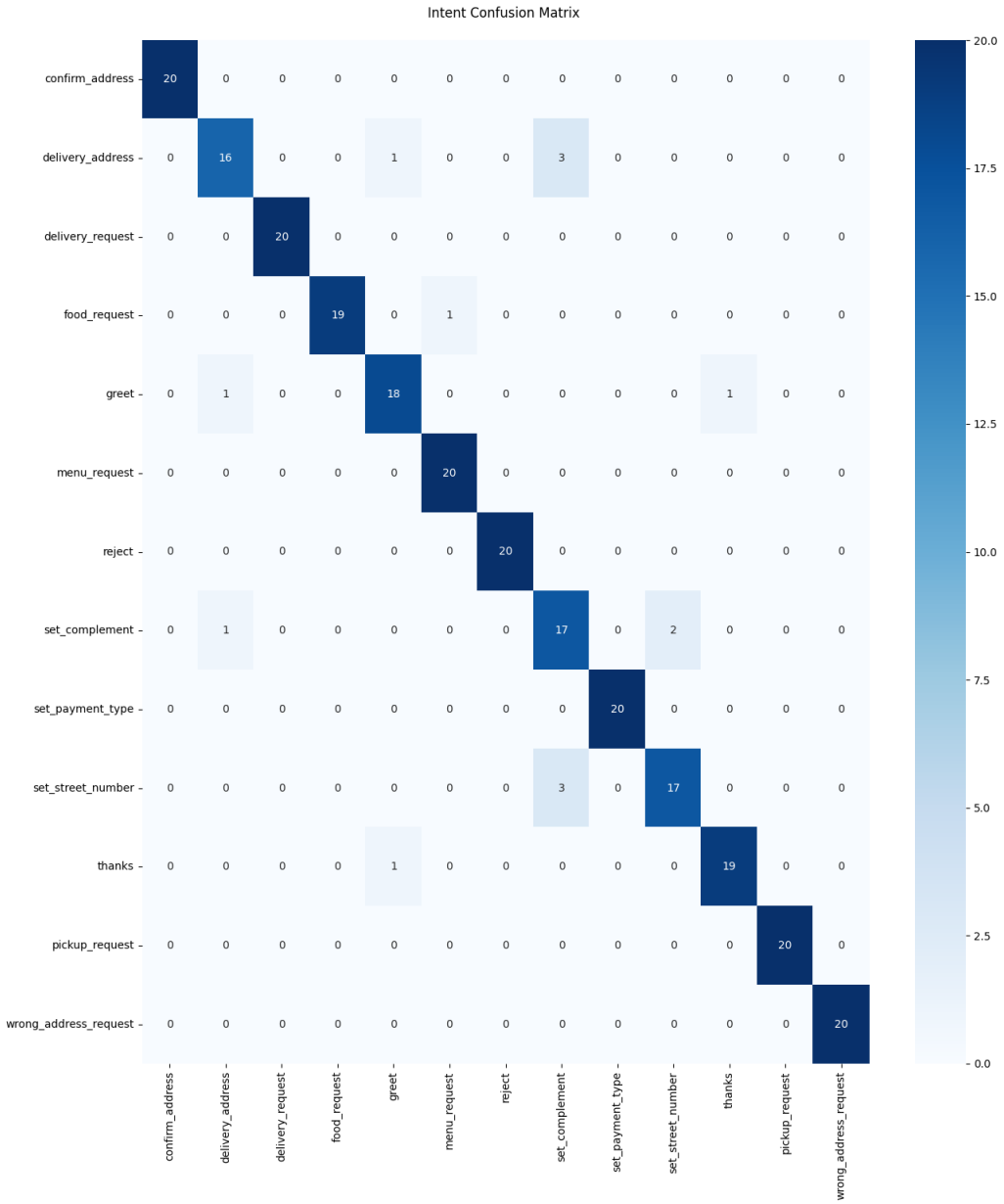
Fonte: Autoria própria.

$$F1\_score = \frac{1{,}79}{1{,}89} = 0{,}95 \tag{8}$$

Outra métrica relevante obtida através dos testes, é a diminuição do tempo médio de uma conversa quando se comparam as interações realizadas com o *chatbot* com as interações realizadas com o atendente humano. A duração média de uma conversa realizada pelo *chatbot* foi de 2 minutos e 30 segundos, enquanto a interação com o atendente humano dura em média 12 minutos. Portanto o uso do *chatbot* gerou uma diminuição de 79% do tempo médio para a realização de um pedido.

A intenção que gerou o maior número de erros foi a *"delivery address"* que é a intenção gerada quando o cliente informa o endereço de entrega de um pedido para *delivery*. Isso se deve ao baixo número de exemplos fornecidos aos modelos e a grande variação nos nomes dos logradouros. Uma possível solução para esse problema, seria a adição de extratores de características densos à *pipeline* do Rasa, que utilizam redes neurais pré-treinadas (como

Figura 15 – Matriz de confusão referente às intenções previstas

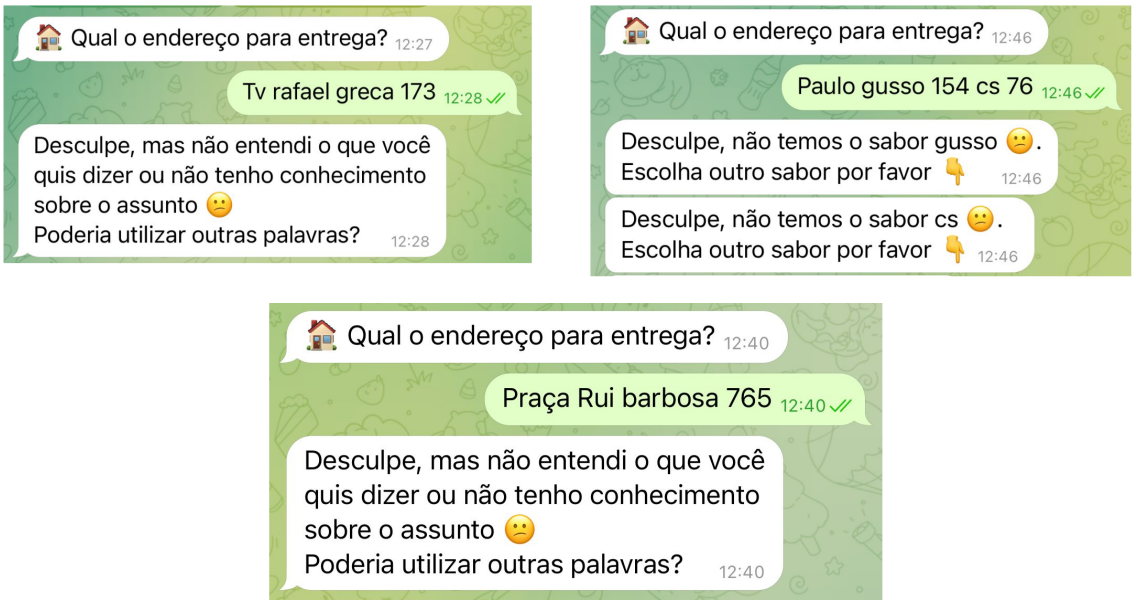


Fonte: Autoria própria.

GPT-3[1]), tendo acesso à uma vasta quantidade de dados, o que poderia aumentar acurácia a identificação dos endereços.

[1] <https://openai.com/api/> - Acesso em 8 de dez. 2022

Figura 16 – Examplos de erros de classificação para a *intent* "delivery address"



Fonte: Autoria própria.

5 CONCLUSÃO

O crescente desenvolvimento tecnológico, principalmente no campo da inteligência artificial - em especial na sub área de processamento de linguagem natural, abre um grande leque de opções para que as empresas também evoluam e tragam novas soluções para melhorar diversas frentes do seu negócio, entre elas a comunicação com seus clientes. E apesar dos desafios enfrentados no desenvolvimento dessas soluções, ferramentas como o Rasa tem papel importante para facilitar e diminuir a curva de aprendizado para implementação de projetos nessa esfera.

O presente projeto teve como objetivo o desenvolvimento de uma solução que possa atender restaurantes e lanchonetes na automação de pedidos realizados por aplicativos de mensagem instantâneas. Esse objetivo foi alcançado com 94% de acurácia geral do modelo de identificação de intenções e diminuição do tempo médio para a realização dos pedidos em 79%.

Com isso, o projeto possibilita que estabelecimentos possam oferecer uma alternativa rápida e inteligente para seus clientes realizarem pedidos via aplicativos de mensagem instantâneas e viabiliza a integração no seu fluxo de pedidos vindos de forma *online*, um assistente virtual que automatize parte desse processo, aumentando a produtividade da equipe e tornando facilitada a escalabilidade do negócio.

5.1 Trabalhos Futuros

Existem algumas opções como trabalhos futuros para esse projeto, entre elas destaca-se:

- Testar e comparar diferentes configurações de modelos de aprendizado de máquina disponíveis no Rasa, podendo assim encontrar uma configuração que melhor se aplique ao cenário do projeto.
- Aprimoramento do assistente: explorar as limitações do fluxo de pedidos do atual assistente desenvolvido, como a impossiblidade de remover ou editar um item escolhido. Nessa mesma linha de trabalho, pode-se explorar também outros ramos de estabelecimentos como pizzarias e hamburguerias, que também seguem o mesmo fluxo de pedidos, porém têm suas particularidades como a adição de itens adicionais ou personalizados e escolha de tamanho do item. Vale ressaltar que esse crescimento para outras aplicações fica dependente de um novo conjunto de dados, então uma etapa de coleta manual de dados reais para cada aplicação se faz necessária.
- Desenvolvimento de um sistema de gerenciamento de pedidos para os restaurantes: o atual projeto teve seu escopo definido até a finalização do pedido feito pelo cliente. Porém após isso, o restaurante tem a necessidade de gerenciar esses pedidos, abrindo

a possibilidade para o desenvolvimento de sistemas que façam esse gerenciamento ou possam se integrar com o atual sistema do restaurante, caso haja. Um diferencial desse sistema é que ele pode ser integrado também com o assistente virtual, melhorando ainda mais a interação com o cliente - avisando quando o pedido ficou pronto, saiu para entrega, etc.

REFERÊNCIAS

ABDELLATIF, A. *et al.* A comparison of natural language understanding platforms for chatbots in software engineering. **IEEE Transactions on Software Engineering**, v. 48, n. 8, p. 3087–3102, 2022.

BARACK, L. **Make your next restaurant reservation with Alexa and Google Assistant**. 2018. Disponível em: https://www.gearbrain.com/restaurant-ideas-alexa-google-assistant-2599259570.html.

BRANDTZAEG, P. B.; FØLSTAD, A. Why people use chatbots. *In*: SPRINGER. **International conference on internet science**. [*S.l.*], 2017. p. 377–392.

BUNK, T. *et al.* Diet: Lightweight language understanding for dialogue systems. **arXiv preprint arXiv:2004.09936**, 2020.

DINESH, T. *et al.* Ai bot for academic schedules using rasa. *In*: **2021 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES)**. [*S.l.*: *s.n.*], 2021.

FERNANDEZ, G. P. Orquestração de contêineres na nuvem: um modelo de segurança. 2018.

GOODFELLOW, I. J.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. Cambridge, MA, USA: MIT Press, 2016.

GRANDINI, M.; BAGLI, E.; VISANI, G. **Metrics for Multi-Class Classification: an Overview**. arXiv, 2020. Disponível em: https://arxiv.org/abs/2008.05756.

HIRSCHBERG, J.; MANNING, C. D. Advances in natural language processing. **Science**, v. 349, p. 261 – 266, 2015.

JIAO, A. An intelligent chatbot system based on entity extraction using rasa nlu and neural network. **Journal of Physics: Conference Series**, IOP Publishing, v. 1487, n. 1, p. 012014, mar 2020. Disponível em: https://dx.doi.org/10.1088/1742-6596/1487/1/012014.

KWON, S.; LEE, J.-H. Divds: Docker image vulnerability diagnostic system. **IEEE Access**, v. 8, p. 42666–42673, 2020.

MCSHANE, M. Natural language understanding (nlu, not nlp) in cognitive systems. **AI Magazine**, v. 38, n. 4, p. 43–56, Dec. 2017. Disponível em: https://ojs.aaai.org/index.php/aimagazine/article/view/2745.

MINAEE, S. *et al.* Deep learning based text classification: A comprehensive review. **CoRR**, abs/2004.03705, 2020. Disponível em: https://arxiv.org/abs/2004.03705.

MURPHY, K. P. **Machine learning : a probabilistic perspective**. Cambridge, Mass. [u.a.]: MIT Press, 2013. ISBN 9780262018029 0262018020.

PEDREGOSA, F. **scikit-learn user guide**. 2019. Disponível em: https://scikit-learn.org/0.21/_downloads/scikit-learn-docs.pdf.

RAD, B. B.; BHATTI, H. J.; AHMADI, M. An introduction to docker and analysis of its performance. **International Journal of Computer Science and Network Security (IJCSNS)**, International Journal of Computer Science and Network Security, v. 17, n. 3, p. 228, 2017.

RASA. **Rasa: Open source conversational AI**. 2022. Disponível em: https://rasa.com/docs/.

RESENDES, S. **26 Online Ordering Statistics Every Restaurateur Should Know in 2021**. 2021. Disponível em: https://upserve.com/restaurant-insider/online-ordering-statistics/.

SHAWAR, B. A.; ATWELL, E. S. Using corpora in machine-learning chatbot systems. **International journal of corpus linguistics**, John Benjamins, v. 10, n. 4, p. 489–516, 2005.

SILVA, F. Araujo da. **Detecção de Ironia e Sarcasmo em Língua Portuguesa: uma abordagem utilizando Deep Learning**. 02 2018. Tese (Doutorado), 02 2018.

SINGH, S.; MAHMOOD, A. The NLP cookbook: Modern recipes for transformer based deep learning architectures. **CoRR**, abs/2104.10640, 2021. Disponível em: https://arxiv.org/abs/2104.10640.

SITTAKUL, V.; JUDPRASONG, S.; SAENGMANEE, N. Smart quiz system for classroom via linebot. *In*: **2019 Research, Invention, and Innovation Congress (RI2C)**. [*S.l.*: *s.n.*], 2019. p. 1–4.

STUDER, S. *et al.* **Towards CRISP-ML(Q): A Machine Learning Process Model with Quality Assurance Methodology**. arXiv, 2020. Disponível em: https://arxiv.org/abs/2003.05155.

TEMBHEKAR, S.; KANOJIYA, M. A survey paper on approaches of natural language processing (nlp). **International Journal of Advance Research, Ideas and Innovations in Technology**, v. 3, n. 3, p. 1496–1498, 2017.

VLASOV, V.; MOSIG, J. E. M.; NICHOL, A. **Dialogue Transformers**. arXiv, 2019. Disponível em: https://arxiv.org/abs/1910.00486.

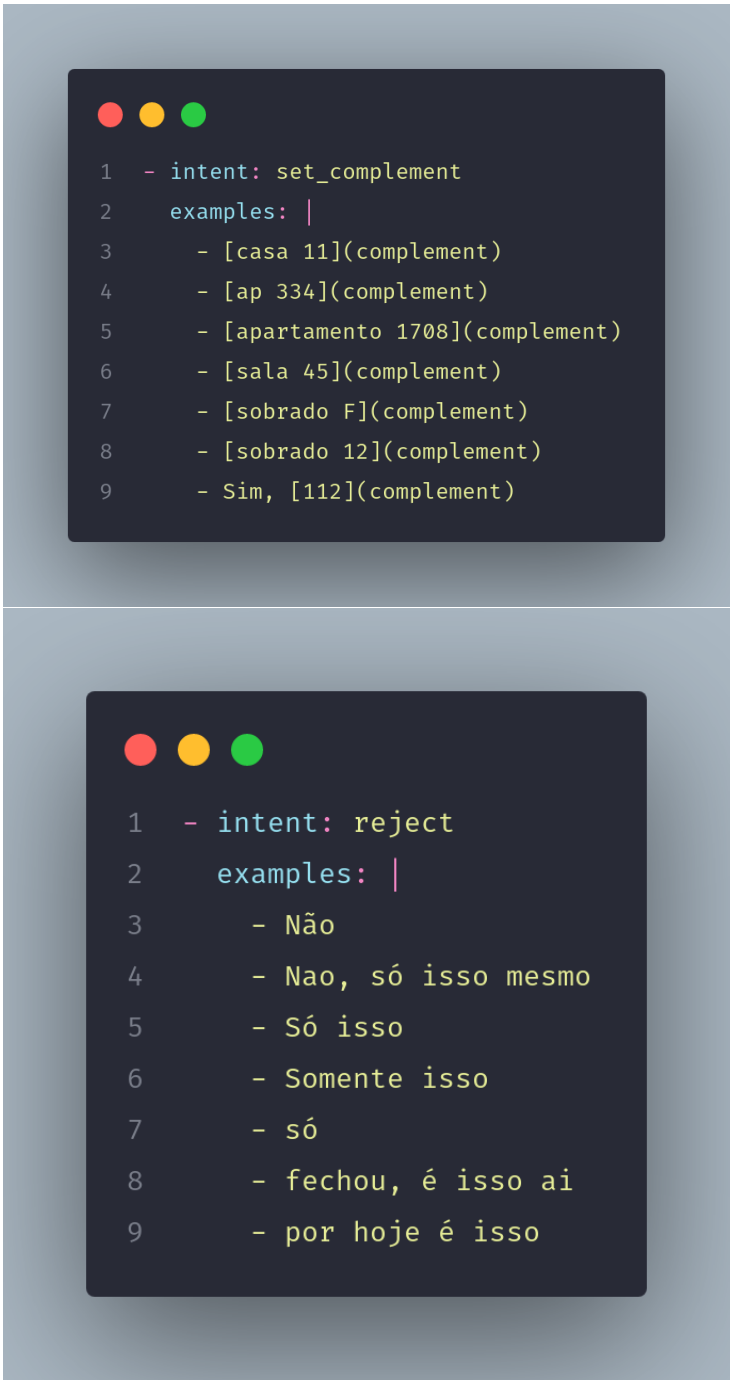
WAN, W. S. *et al.* What wechat can learn from whatsapp? customer value proposition development for mobile social networking (msn) apps: A case study approach. **Journal of Theoretical and Applied Information Technology**, v. 97, n. 4, p. 25, 2019.

ZHANG, X.; ZHAO, J.; LECUN, Y. **Character-level Convolutional Networks for Text Classification**. arXiv, 2015. Disponível em: https://arxiv.org/abs/1509.01626.

ANEXO A – Recorte de exemplos dos dados de treinamento

```yaml
- intent: greet
  examples: |
    - Ola
    - Boa tarde
    - Opa, bom dia
    - Olaaaa, bom diaaa
    - Boa tarde tudo bem?
    - Bom dia
    - Olá, bom dia
    - Bom dia
    - Bom dia. Tudo bem?
    - Oii Tudo bem?
    - Oie
```

```yaml
- intent: menu_request
  examples: |
    - Pode me mandar o cardápio?
    - quais as opções hoje?
    - Oq vc tem hoje?
    - Qual o cardápio?
    - quais os sabores de pastel?
    - qual o cardápio?
    - O q tem hj
    - Quais são às opções de cardápio?
    - Consegue me mandar o cardápio?
```



```
- intent: set_complement
  examples: |
    - [casa 11](complement)
    - [ap 334](complement)
    - [apartamento 1708](complement)
    - [sala 45](complement)
    - [sobrado F](complement)
    - [sobrado 12](complement)
    - Sim, [112](complement)
```

```
- intent: reject
  examples: |
    - Não
    - Nao, só isso mesmo
    - Só isso
    - Somente isso
    - só
    - fechou, é isso ai
    - por hoje é isso
```

```
- intent: thanks
  examples: |
    - Beleza, muito obrigado
    - Valeu
    - Obrigado
    - certinho, valeu!
    - obrigada!
    - fechou, tmj!
    - Valeu minha querida!
```