Ormazabal Lima do Nascimento

Estudo comparativo entre modelos *transformers* aplicados ao desenvolvimento de *chatbots*

Natal–RN

Dezembro de 2023

Ormazabal Lima do Nascimento

Estudo comparativo entre modelos *transformers* aplicados ao desenvolvimento de *chatbots*

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Tecnologia da Informação do Instituto Metrópole Digital da Universidade Federal do Rio Grande do Norte como requisito para a obtenção do grau de Mestre em Tecnologia da Informação.

Orientador: Dr. Daniel Sabino Amorim de Araújo

Universidade Federal do Rio Grande do Norte – UFRN

Instituto Metrópole Digital – IMD

Programa de Pós-Graduação em Tecnologia da Informação

Mestrado Profissional em Tecnologia da Informação

Natal–RN

Dezembro de 2023

Universidade Federal do Rio Grande do Norte - UFRN
Sistema de Bibliotecas - SISBI
Catalogação de Publicação na Fonte. UFRN - Biblioteca Central Zila Mamede

Nascimento, Ormazabal Lima do.
Estudo comparativo entre modelos transformers aplicados ao desenvolvimento de chatbots / Ormazabal Lima do Nascimento. - 2024.
74 f.: il.

Dissertação (mestrado) - Universidade Federal do Rio Grande do Norte, Instituto Metrópole Digital, Programa de Pós-Graduação em Tecnologia da Informação, Natal, RN, 2024.
Orientação: Prof. Dr. Daniel Sabino Amorim de Araújo.

1. Chatbot - Dissertação. 2. Transformers - Dissertação. 3. Inteligência artificial - Dissertação. I. Araújo, Daniel Sabino Amorim de. II. Título.

RN/UF/BCZM CDU 004

Elaborado por Ana Cristina Cavalcanti Tinoco - CRB-15/262

Ormazabal Lima do Nascimento

Estudo comparativo entre modelos *transformers* aplicados ao desenvolvimento de *chatbots*

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Tecnologia da Informação do Instituto Metrópole Digital da Universidade Federal do Rio Grande do Norte como requisito para a obtenção do grau de Mestre em Tecnologia da Informação.

Orientador: Dr. Daniel Sabino Amorim de Araújo

Trabalho aprovado. Natal–RN, 15 de Dezembro de 2023:

Prof. Dr. Daniel Sabino Amorim de Araújo
UFRN

Prof. Dr. João Carlos Xavier Júnior
UFRN

Profa. Dra. Thaís Gaudencio Do Rêgo
UFPB

Natal–RN
Dezembro de 2023

AGRADECIMENTOS

Agradeço a Deus por todas as bençãos que a mim foram concedidas.

Aos meus pais, em especial minha mãe Lenita, por estar sempre junto nos principais momentos de minha vida, me dando força e apoio.

Aos meus familiares, por entenderem meus momentos de ausência, mas principalmente por toda a torcida.

À minha namorada Mariana, por acreditar em mim e me apoiar nas minhas escolhas.

Ao meu orientador, professor Daniel Sabino, pela oportunidade e por todo o apoio prestado na elaboração deste trabalho.

Aos membros presentes nas bancas de qualificação e defesa, professores Eiji Adachi, João Carlos e Thaís Gaudencio, por toda a preocupação ao contribuir positivamente com este trabalho.

Aos meus amigos de curso, Yuri e Gabriel, pela troca de experiências, pelas risadas e pelas críticas construtivas.

À equipe da UFRN e do TCE, que fazem do Mestrado em TI uma experiência singular na vida das pessoas que participam dele.

A todos que de alguma forma fizeram parte dessa conquista.

RESUMO

Os *chatbots* são *softwares* que utilizam linguagem natural para se comunicar com seus usuários. Para a sua criação podem ser utilizados os modelos *transformers*, um tipo de rede neural que vêm apresentando resultados promissores em áreas de estudo como processamento de linguagem natural (NLP). A variedade de modelos existentes impõe desafios na seleção das melhores opções para o nicho de mercado em que o *chatbot* irá atuar. Este trabalho apresenta um estudo comparativo entre os modelos *transformers* DIET, LaBSE, BERTimbau, DistilBERT, RoBERTa, GPT, GPT-2 e XLNET aplicado ao desenvolvimento um *chatbot* submetido aos dados capturados nos atendimentos ao público realizados pelo Tribunal de Contas do Estado do Rio Grande do Norte (TCE/RN). O estudo considerou métricas como acurácia e precisão, além dos tempos de treinamento e de processamento de resposta. DIET foi considerado o melhor modelo por conseguir equilibrar bons resultados nas métricas de qualidade e tempo em conjunto, destacando-se no fator tempo por responder até 2,5 vezes mais rápido que os demais.

Palavras-chaves: *chatbot*, *transformers*, inteligência artificial.

ABSTRACT

Chatbots are software that use natural language to communicate with their users. Transformers models can be used to create them, a type of neural network that has been showing promising results in areas of study such as natural language processing (NLP). The variety of existing models poses challenges in selecting the best options for the niche market in which the chatbot will operate. This work presents a comparative study between the transformer models DIET, LaBSE, BERTimbau, DistilBERT, RoBERTa, GPT, GPT-2 and XLNET applied to the development of a chatbot submitted to data captured in public services carried out by the Court of Auditors of the State of Rio Grande do Norte (TCE/RN). The study considered metrics such as accuracy and precision, in addition to training and response processing times. DIET was considered the best model for managing to balance good results in quality and time metrics together, standing out in the time factor for responding up to 2.5 times faster than the others.

Keywords: chatbot, transformers, artificial inteligence.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AE	*Autoencoder (Auto-Codificado)*
AIML	*Artificial Intelligence Markup Language (Linguagem de Marcação de Inteligência Artificial)*
ALICE	*Artificial Linguistic Internet Computer Entity (Entidade Artificial Linguística de Computadores da Internet)*
AR	*Autoregressive (Auto-Regressivo)*
BERT	*Bidirectional Encoder Representations from Transformers (Representações de Codificadores Bidirecionais de Transformadores)*
BPE	*Byte-Pair Encoding (Codificação de pares de bytes)*
brWAC	*Brazilian Web as Corpus (Rede Brasileira como Corpus)*
CAJ	*Central de Atendimento ao Jurisdicionado*
CRF	*Conditional Random Field (Campo Aleatório Condicional)*
CV	*Cross Validation (Validação Cruzada)*
DIET	*Dual Intent and Entity Transformer (Dupla Intenção e Transformador de Entidade)*
DSRM	*Design Science Research Methodology (Metodologia de Pesquisa em Ciência do Projeto)*
FFN	*Feed Forward Network (Rede Olhar pra Frente)*
FN	*False Negative (Falso Negativo)*
FP	*False Positive (Falso Positivo)*
GPT	*Generative Pre-Trained Transformer (Transformador Generativo Pré-Treinado)*
HIST	*Hybrid Intent and Slots Transformers (Intenção Híbrida e Transformadores de Vagas)*
IA	*Inteligência Artificial*

LaBSE	*Language Agnostic BERT Sentence Embeddings (Incorporações de Frases BERT Independentes de Idioma)*
LLM	*Large Language Model (Modelo de Linguagem Grande)*
LSTM	*Long Short-Term Memory (Memória de Longo Prazo)*
mBERT	*Multilingual BERT (BERT Multilíngue)*
ML	*Machine Learning (Aprendizado de Máquina)*
MLM	*Masked Language Model (Modelo de Linguagem Mascarada*
MTL	*Multi-Task Learning (Aprendizado Multi-Tarefa)*
NER	*Named Entity Recognition (Reconhecimento de Entidade Nomeada)*
NLI	*Natural Language Inference (Inferência de Linguagem Natural)*
NLP	*Natural Language Processing (Processamento de Linguagem Natural)*
NLU	*Natural Language Understanding (Compreensão de Linguagem Natural)*
NSP	*Next Sentence Prediction (Previsão da Próxima Frase)*
QA	*Question Answering (Resposta a Pergunta)*
RNN	*Recurrent Neural Network (Rede Neural Recorrente)*
RoBERTa	*Robustly Optimized BERT Pretraining Approach (Abordagem de Pré-Treinamento BERT Robustamente Otimizada)*
RTE	*Recognizing Textual Entailment (Reconhecimento de Implicação Textual)*
SECEX	*Secretaria de Controle Externo*
seq2seq	*sequence-to-sequence (sequência-a-sequência)*
SQuAD	*Stanford Question Answering Dataset (Conjunto de dados de resposta a perguntas de Stanford)*
STS	*Sentence Textual Similarity (Semelhança Textual de Frases)*
TCE	*Tribunal de Contas do Estado*
TLM	*Translation Language Modeling (Modelagem de Linguagem de Tradução)*
TP	*True Positive (Verdadeiro Positivo)*

SUMÁRIO

1 INTRODUÇÃO

De acordo com o Art. 52 da Constituição Estadual (NORTE, 2019), compete à Assembleia Legislativa, através do Tribunal de Contas do Rio Grande do Norte (TCE/RN), a fiscalização contábil, financeira, orçamentária, operacional e patrimonial do Estado e de todas entidades da administração direta e indireta, denominada de controle externo.

Os jurisdicionados são os órgãos que devem essa prestação de contas ao TCE, como exemplo secretarias, prefeituras e câmaras municipais. Para realizá-la, eles se submetem às regras estabelecidas pelo tribunal, e fazem uso dos sistemas oferecidos por ele para esse fim. Como uma forma de centralizar a comunicação entre o órgão e os jurisdicionados, além de sanar as dúvidas a cerca do uso dos sistemas, em fevereiro de 2020, a Secretaria de Controle Externo (SECEX) do TCE, instituiu a criação da Central de Atendimento ao Jurisdicionado (CAJ).

De início, as formas de atendimento implementadas foram presencial, telefone e *e-mail*. Com o aumento gradual da informatização dos serviços do tribunal, e diante da limitação do número de servidores, o órgão se mostrou interessado em investir também na implementação de um *chatbot*, que entra como mais uma forma de atendimento, e visa diminuir a demanda de atendimentos em que é necessária atuação de um servidor.

Um *chatbot* é um *software* capaz de conversar com os usuários de maneira natural. Oferecem auxílio ao usuário em uma interação homem-máquina. Possuem capacidade de examinar e até mesmo influenciar o comportamento do seu usuário, perguntando e respondendo às suas perguntas (SAMEERA; JOHN, 2015).

Podem ser identificadas três gerações de *chatbot* (NEVES; BARROS; HODGES, 2006):

- A primeira geração, baseada em técnicas de casamento de padrões e regras gramaticais, tem como principal representante ELIZA (WEIZENBAUM, 1966), um robô de conversação que simula um psicólogo virtual, e utiliza a reformulação de trechos das frases capturados das entradas dos usuários, fazendo parecer que possui um vasto vocabulário (CURRY; O'SHEA; CROCKETT, 2012). Apesar do seu pioneirismo, ELIZA não possui memória, logo não se lembra de nada do que foi dito na interação com o usuário (JOHNSTON, 2008).
- A segunda geração, baseada em técnicas de Inteligência Artificial (IA), como as regras de produção e redes neurais, é lembrada pelo JULIA (MAULDIN, 1994). Sua função era ajudar e fornecer dicas para outros jogadores do jogo TinyMUD.

Diferentemente dos outros *chatbot*, JULIA possui bastante conhecimento sobre o mundo onde atua, uma vez que este tem um escopo reduzido, fornecendo um nível de conversação aceitável sobre o assunto (JUNIOR, 2008). Em sua primeira versão, JULIA apresentava um algoritmo simples, do tipo *if then else*, contudo, com o decorrer do tempo, suas habilidades na conversação foram alteradas para um modelo mais robusto, baseado em redes neurais (LEONHARDT et al., 2003).

- A terceira geração é baseada no uso de linguagens de marcação para a construção de bases de conhecimento. Utilizam a linguagem AIML (*Artificial Intelligence Markup Language*). Segundo MARIETTO et al. (2013), seu maior representante é ALICE (*Artificial Linguistic Internet Computer Entity*) (WALLACE, 2004). O projeto possui uma base de conhecimento constituída por centenas de fatos, citações e ideias de seu criador, apresentando um vocabulário de mais de 5.000 palavras. Foi vencedor três vezes do Loebner Prize (WALLACE, 2009).

Com base nos últimos avanços tecnológicos, as IAs Generativas podem ser consideradas parte de uma nova geração. Construídas a partir de modelos *transformers*, elas são capazes de criar respostas em tempo real e de manter longas conversas sem esquecer do seu histórico, ou seja, sem esquecer o que acabaram de falar. Dentre os modelos dessa categoria, o ChatGPT (BROWN et al., 2020) é um dos principais representantes.

Os *chatbots* por vezes são confundidos com os chamados *softwares* agentes. Esse tipo de *software* trabalha de forma autônoma e automatiza tarefas repetitivas, como buscar ofertas, sugerir produtos, etc., mas não podem ser considerados *chatbots*, pois não entendem linguagem natural.

O desenvolvimento de um *chatbot* envolve uma série de conhecimentos e tecnologias, como o uso de técnicas de Aprendizado de Máquina (ML - *Machine Learning*), para treinamento e validação de grandes quantidades de dados, e de Processamento de Linguagem Natural (NLP - *Natural Language Processing*), para garantir a melhor interpretação das mensagens trocadas durante a conversa.

Diversas são as tarefas de NLP em que podem ser aplicados modelos de ML, como tradução e resumo de texto (SUN et al., 2021). Nesse trabalho, nossa atenção se voltará para a tarefa de perguntas e respostas (QA - *Question Answering*), algo bastante relacionado ao uso de *chatbots*. Conforme CORTES (2019), QA é um campo de pesquisa das áreas de Recuperação de Informações e NLP que propõe, de forma autônoma, responder perguntas feitas por humanos em linguagem natural. Uma das principais etapas desses sistemas é a classificação de intenções, em que o sistema busca identificar o tipo de resposta que a pergunta se refere.

Na etapa de classificação de intenções, o modelo tentará associar a pergunta do

usuário a alguma intenção. As intenções são o que o próprio nome diz, e representam o objetivo do usuário ao fazer uma pergunta; elas também podem ser entendidas como classes, e cada uma delas armazena um conjunto de respostas que podem ser usadas durante o diálogo. Obter um modelo eficaz nessa classificação se traduz no sucesso de um *chatbot*, pois permite que a resposta seja escolhida corretamente e o diálogo possa prosseguir, aproximando o resultado ao de uma conversa entre seres humanos. A tarefa de classificação de intenções é um dos enfoques principais desse trabalho e será por meio dela que os modelos serão avaliados. Diante disso, optou-se por utilizar modelos *transformers* nesse trabalho, pois eles apresentam características interessantes quando comparados as Redes Neurais Recorrentes (RNN - *Recurrent Neural Network*), um outro tipo de rede que poderia ter sido utilizada nesse problema.

A arquitetura *transformer* (VASWANI et al., 2017) surge em 2017 e, dentre outras vantagens, facilita a paralelização durante a fase de treinamento. Isso possibilitou o uso de conjuntos de dados maiores do que era possível antes de seu surgimento. Com isso, apareceram os modelos pré-treinados baseados nessa arquitetura, que utilizam uma quantidade massiva de dados na etapa de treinamento, e que podem se ajustar a diferentes tarefas.

Dentre os modelos baseados na arquitetura *transformer*, o BERT (DEVLIN et al., 2019) é um dos mais importantes em QA. Ele possui a característica de processar o texto em ambos os sentidos, um fator bastante importante na captura de contexto entre os *tokens*. O modelo obteve resultados estado da arte em QA na época do lançamento de seu artigo (DEVLIN et al., 2019).

1.1 Justificativa

Criada em fevereiro de 2020, a CAJ faz parte da SECEX. Por meio dela os jurisdicionados, e a sociedade em geral, podem tirar dúvidas a cerca das normas as quais eles estão submetidos junto ao TCE, como também solicitar orientações sobre o uso dos sistemas desenvolvidos pelo tribunal. Somente por meio desses sistemas os jurisdicionados podem resolver suas pendências junto ao tribunal, e por esse uso possuir esse caráter de obrigatoriedade, os sistemas são lançados com um conjunto mínimo de documentos para dar apoio ao seu uso, como manual do sistema e as normas jurídicas que o regem.

No entanto, percebe-se ainda assim a necessidade de uma ajuda mais direcionada. No primeiro semestre de 2021, a CAJ realizou mais de 7800 atendimentos. Esses atendimentos são feitos de forma presencial, e através de e-mail ou telefone. Apesar de suprir bem a sua demanda, a tendência natural é que o órgão continue investindo na informatização de seus procedimentos e o número de sistemas lançados cresça, aumentando também a demanda da secretaria.

Com um número de servidores limitado, a secretaria já vem estudando formas de tornar esses atendimentos mais céleres. O WhatsApp é mais uma forma de atendimento que ela pretende implementar. Através desse trabalho, a Secretaria pretende dar um passo além do simples uso dessa ferramenta, permitindo que mais usuários possam ser atendidos, sem a necessidade do realocamento de servidores, já que se trata de um recurso limitado.

O *chatbot* pode trazer a eficiência que faltava ao setor, pois num primeiro momento, será possível atender bem mais usuários no mesmo tempo do que anteriormente. Entretanto, sem um modelo bem escolhido e treinado, ele terá dificuldade em entender as dúvidas do cliente, podendo resultar em frustração por parte deste público que não conseguirá se fazer entendido pela ferramenta. Existe também a questão do desperdício de recursos computacionais, visto que o tempo de atendimento poderá ser até piorado após a implantação da ferramenta, e também em uma má impressão já que o *chatbot* será o primeiro contato entre o jurisdicionado e o tribunal.

Diante das diversas opções de modelos que podem ser empregados no desenvolvimento de um *chatbot*, o objetivo deste trabalho será comparar e eleger a melhor opção pensada no atendimento ao público do tribunal, balanceando qualidade e a eficiência das respostas. Para esse fim, o comparativo usará diálogos reais disponibilizados pelo tribunal. Porém, mesmo focado em tentar resolver os problemas encontrados no TCE, esse trabalho pode se adequar a situação enfrentada por qualquer empresa que procura mais eficiência no seu setor de atendimento ao cliente, bastando que seus dados sejam colhidos e submetidos aos mesmos testes aqui presentes.

O DIET (BUNK et al., 2020) é um classificador desenvolvido pela equipe do Rasa (BOCKLISCH et al., 2017), plataforma que será usada para o desenvolvimento do *chatbot*. Ele pode atuar nas tarefas de classificação de intenções e reconhecimento de entidades, e além disso é um modelo *transformer*. Ele permite o acoplamento com outros modelos *transformers*, e essa característica será usada para conectarmos a sua rede aos modelos presentes no comparativo proposto neste trabalho. Devido ao seu módulo *transformer*, ele executa processamento similar aos outros modelos desse comparativo, na etapa pré-classificação, e pode solucionar o problema sozinho, sem necessidade de acoplamento com outros modelos. Devido a essa característica, ele também entrará no comparativo como uma alternativa aos outros modelos *transformers*.

O estudo também se preocupará em criar algoritmos automatizados para limpeza da base de dados. A etapa de limpeza transforma os dados brutos colhidos em atendimentos reais pelo TCE em textos que possuem a estrutura correta para ser entendida pela ferramenta. A automatização dessa fase se traduz em eficiência ao adicionar novos dados ao sistema.

1.2 Objetivo

O objetivo deste trabalho é realizar um estudo comparativo entre o modelo BERT (DEVLIN et al., 2019) e outros modelos *transformers* na tarefa de classificação de intenções. Cada modelo será treinado com um *corpus* que contém dados reais dos atendimentos da CAJ, e a sua saída será conectada ao classificador DIET (BUNK et al., 2020). A partir deste objetivo, se desdobram os seguintes objetivos específicos:

- Projetar e implementar no Rasa um *chatbot* com integração ao WhatsApp que se comunique usando apenas linguagem natural
- Avaliar a eficácia dos modelos *transformers* suportados pelo Rasa na tarefa de classificação de intenções
- Avaliar dentre esses modelos quais os mais eficientes no processamento de resposta
- Identificar padrões que possibilitem a limpeza e o processamento dos dados e resultem em um formato que possa ser interpretado pelo Rasa

Os atendimentos do setor envolvem certa complexidade devido ao grande número de temas que podem ser abordados. Eles vão desde dúvidas de usabilidade nos diversos sistemas do tribunal, como também informações e apoio jurídico nos trâmites que ainda não foram informatizados.

Como são diversos os meios de atendimento, a CAJ implementou um formulário a ser respondido no fim de cada atendimento pelo operador que atendeu ao chamado. Esses formulários preenchidos foram cedidos pelo setor para o desenvolvimento dessa ferramenta, um montante de quase quatro mil formulários.

Os dados contidos nos formulários serão utilizados em etapas de treinamento e validação do *chatbot*. Será feita a sua análise e catalogação, separando-os por temas e intenções. O texto descritivo será convertido em conjuntos de perguntas e respostas que formarão a base de dados inicial da ferramenta. Através dessa metodologia será possível realizar os testes que permitirão cumprir os objetivos propostos neste trabalho.

Se analisarmos o retrato dos tribunais de contas no país, veremos que cada tribunal faz a produção de seus próprios sistemas, entretanto o problema que cada uma dessas ferramentas vem resolver é praticamente idêntico. O êxito desse trabalho de pesquisa permitiria a aplicação dessa ferramenta, nos mesmos moldes utilizados aqui, em outros tribunais pelo país.

1.3 Estrutura do Trabalho

Esta dissertação está dividida da seguinte forma: no Capítulo 2 encontra-se uma introdução teórica sobre os modelos testados. O Capítulo 3 apresenta uma revisão de literatura sobre o tema. No Capítulo 4 será feito o detalhamento do funcionamento do *software* e dos experimentos. Por fim, o Capítulo 5 se debruçará sobre a experimentação e a análise desses experimentos, e no Capítulo 6, as considerações finais.

2 REFERENCIAL TEÓRICO

Neste capítulo serão descritas as ferramentas utilizadas na implementação do *chatbot* e o referencial teórico necessário para o entendimento do trabalho. Na Seção 2.1 será explicada a arquitetura *transformer*, que é inspiração para os modelos explicados das Seções 2.2 a 2.10; Na Seção 2.11 são apresentadas a plataforma Rasa, usada no desenvolvimento do *chatbot*, e a biblioteca Venom Bot, responsável por trocar as mensagens da aplicação com o WhatsApp.

2.1 *Transformer*

O *transformer* é uma arquitetura de rede neural apresentada em 2017 no trabalho de VASWANI et al. (2017). Ele traz soluções para problemas encontrados nas Redes Neurais Recorrentes (RNN - *Recurrent Neural Network*). A RNN tem uma arquitetura similar a rede *feedforward*, mas com a adição de um *loop* de *feedback*, que faz com que as decisões atuais dela sejam impactadas por decisões anteriores. Assim, costuma-se dizer que esse tipo de rede possui memória (BOOK, 2022).

A RNN sofria do problema do desaparecimento do gradiente, que causa perda de memória de longo prazo. A RNN processa o texto sequencialmente, o que significa que se houver frases muito longas, quando estiver processando informações no final da frase, ela esquecerá as informações encontradas no início. Esse problema foi resolvido em partes nas redes LSTM (*Long Short-Term Memory*), através da adição de mais algumas células de memória. Ainda assim, similarmente a RNN, o processamento da entrada era sequencial, em que as palavras eram processadas uma a uma, e as incorporações de palavras (*word embeddings*) não consideravam o contexto (LIMA, 2022).

Os *transformers* apresentam soluções para ambos os pontos. A arquitetura *transformer* é a primeira a utilizar apenas auto-atenção, que é um mecanismo que determina a relevância relativa de um *token* com relação aos demais *tokens* da sequência, ou seja, o contexto em que as palavras se inserem na frase estará presente nas incorporações realizadas. Além disso, permite a paralelização, podendo iniciar o processamento de qualquer ponto da frase. Ele segue uma estrutura codificador-decodificador como visto na Figura 1. O codificador recebe uma sequência de palavras na entrada, e as transforma numa sequência de valores, que serão aplicados na entrada do decodificador, gerando uma sequência de palavras na saída (VASWANI et al., 2017).

A codificação (*Input Embedding)* é aplicada na entrada do codificador, e converte cada *token* de entrada em um vetor de tamanho definido pelo desenvolvedor. Ao final do

processo, isso resulta em uma matriz de valores, onde cada linha representa um *token*. Como o modelo não contém recorrência e nem convolução, para que ele faça uso da ordem da sequência, devemos injetar alguma informação sobre a posição relativa ou absoluta dos *tokens* na sequência. Assim, é gerada uma matriz posicional (*Positional Encoding*), onde cada linha dela representa a posição de um *token* específico. Essa matriz é somada a matriz de codificação, e a matriz resultante é passada para o codificador (VASWANI et al., 2017).

Toda a explicação do parágrafo anterior se aplica também a entrada do decodificador (*Output Embedding*), mas aqui apenas os *tokens* já processados pelo decodificador voltam a ser injetados na entrada dele. Por exemplo, se o codificador está processando determinado *token* da sentença, na entrada do decodificador teremos apenas os *tokens* anteriores a esse, somados a uma matriz posicional (VASWANI et al., 2017).

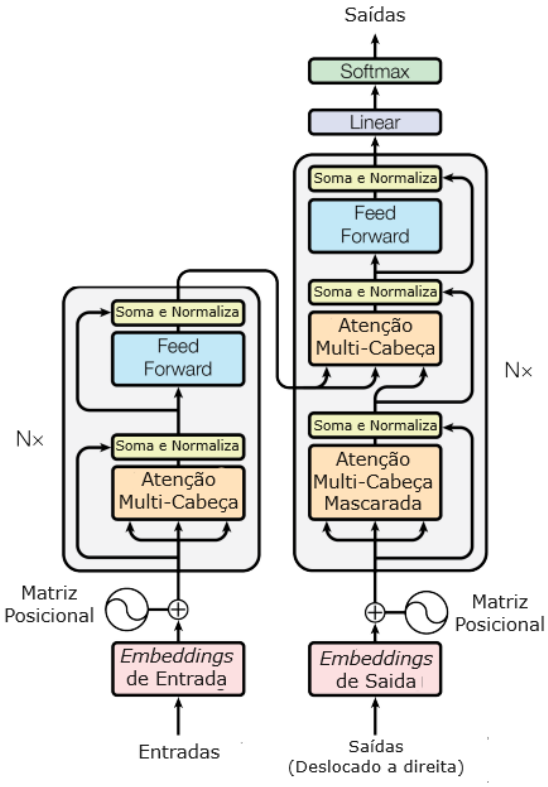

Figura 1 – Arquitetura de um *transformer*, com N codificadores à esquerda e N decodificadores à direita. Fonte: (VASWANI et al., 2017)

O mecanismo de atenção codifica os *tokens* em pontuação de importância. Dada uma sentença, é possível definir quais *tokens* são mais relevantes para resolver determinado problema. A função matemática da auto atenção está representada na Equação 2.1 e tem como parâmetros Sentenças (Q), Chaves (K) e Valores (V). Nela, Q é o vetor de um determinado *token* da sentença de entrada; K é o vetor de entrada de todos os outros *tokens* da sequência, incluindo o *token* de Q. O produto escalar entre Q com cada K

resulta em d_k pesos, onde d_k é a dimensão de K (VASWANI et al., 2017).

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.1}$$

Os pesos então passam por uma *softmax* que resultará em valores entre zero e um, a depender do grau de importância do *token* na sequência. Finalmente, o resultado é multiplicado por V, que são os vetores de cada palavra da entrada, que garante que palavras mais importantes serão pouco atingidas na pontuação, enquanto palavras menos importantes terão valores próximos a zero, e assim o modelo entenderá que elas são pouco relevantes (VASWANI et al., 2017). Seu esquema está na Figura 2.

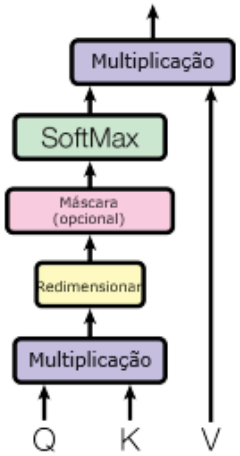

Figura 2 – Produto Escalar Normalizado. O primeiro bloco representa o produto entre os vetores Q e V. O resultado entra no bloco Redimensionar, que o divide por d_k. O bloco Máscara é opcional, e é usado quando houverem sequências de diferentes tamanhos. Nesse caso, elas são preenchidas com *tokens* mascarados até que fiquem com mesmo comprimento. O objetivo é se beneficiar de um recurso de paralelização do PyTorch. O bloco SoftMax traz todos os valores para o intervalo entre zero e um, e novamente o bloco de multiplicação representando a multiplicação pelo vetor V. Fonte: (VASWANI et al., 2017)

O mecanismo de atenção é repetido múltiplas vezes em paralelo. Cada repetição é denominada de cabeça de atenção (*attention head*). Q, K e V são divididos em n partes, sendo n o número de cabeças. Em cada uma delas é aplicado o produto escalar normalizado. Ao final, o resultado é concatenado, permitindo obter as informações que foram adquiridas em cada aplicação do produto escalar normalizado. Esse mecanismo é chamado de multi-cabeça (*multi-head attention*) e está representado na Figura 3 (VASWANI et al., 2017).

Como cada cabeça de atenção foca em uma parte dos dados, cada resultado permitirá ao modelo aprender determinados relacionamentos e detalhes sobre o comportamento das palavras dentro das frases. Assim, o resultado final permite ao modelo obter o máximo de informações sobre uma dada sentença (VASWANI et al., 2017).

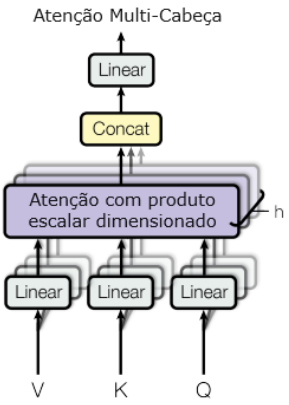

Figura 3 – Multi-Cabeça (*Multi-head attention*). Fonte: (VASWANI et al., 2017)

Enquanto o codificador possui apenas um mecanismo de atenção, com acesso a toda a sequência de entrada, no decodificador existem dois: um mais próximo da entrada calculando a auto-atenção dos *tokens* já processados; e outro chamado mecanismo de atenção codificador-decodificador, que se conecta tanto ao mecanismo de atenção anterior, como também leva em consideração o processamento da sentença completa que acontece no codificador (VASWANI et al., 2017).

Concluindo, *transformers* apresentam larga vantagem no tempo de treinamento, quando comparado a arquiteturas de redes usadas antes do seu surgimento. Isso se deve a auto-atenção atrelada ao uso de paralelização. Segundo MERRITT (2022), ao encontrar padrões entre os elementos matematicamente, os *transformers* também eliminam a necessidade de conjuntos de dados rotulados, que a depender do tamanho, eram caros e demorados de produzir, e colocam a disposição os trilhões de imagens e petabytes de dados de texto na web e em bancos de dados corporativos. Ele é usado em tarefas do tipo *sequence-to-sequence* (seq2seq), ou seja, aquelas que recebem uma sequência de dados e produzem outra sequência de dados, como tradução e resumo de texto.

2.2 BERT

Os modelos de linguagem pré-treinados têm se mostrado efetivos em diversas tarefas de NLP, incluindo QA. Ao usar modelos pré-treinados em outros problemas, duas estratégias podem ser adotadas: *feature-based*, que usa as representações pré-treinadas como atributos adicionais, e ajuste fino (*fine-tuning*), que é o ajuste dos parâmetros pré-treinados para a tarefa em questão. Ambas usam modelos de linguagem unidirecionais (DEVLIN et al., 2019).

O modelo unidirecional limita a quantidade de arquiteturas que podem ser usadas. Por exemplo, caso escolhida uma arquitetura esquerda para direita (*left-to-right*), cada *token* só teria acesso aos *tokens* anteriores nas camadas de auto-atenção do *transformer*.

Essas restrições podem ser bastante prejudiciais no caso de QA, onde é crucial incorporar o contexto de ambas as direções. Como solução, o BERT (*Bidirectional Encoder Representations from Transformers*) usa MLM (*Masked Language Model*), que permite que o treinamento ocorra em ambos os sentidos (DEVLIN et al., 2019).

O BERT utiliza o mesmo mecanismo de codificação do *transformer* em sua arquitetura. Ela está dividida em três camadas: uma para processar as entradas e transformá-las em uma matriz de codificações; uma que possui um conjunto de codificadores *transformer*, no qual cada codificador possui um módulo de auto-atenção, que levando em consideração o contexto, alterará a codificação (*word embedding*) de cada *token*, de acordo com a importância dele na sentença; e uma camada que permite realizar o ajuste das saídas, permitindo adaptar o BERT para resolver diversos problemas de NLP.

A pilha de codificadores é uma característica da aprendizagem profunda (*deep learning*). Ela permite capturar dependências mais complexas na sequência de entrada, semelhante às múltiplas camadas ocultas usadas em redes neurais tradicionais. Foram criados dois modelos: $BERT_{BASE}$ que possui 12 codificadores, uma matriz de codificação com 768 colunas e 12 cabeças de auto-atenção; $BERT_{LARGE}$ que possui 24 codificadores, matriz de codificação com 1024 colunas e 16 cabeças de auto-atenção. O modelo deve ser escolhido de acordo com a tarefa (DEVLIN et al., 2019).

O formato dos dados de entrada podem ser uma sentença ou um par de sentenças (Pergunta, Resposta). Na tokenização, é usado o *WordPiece embeddings*, que conta com um vocabulário de 30.000 *tokens*. Caso a palavra não seja encontrada no vocabulário, é verificado se parte dela faz parte, podendo implicar na divisão de uma palavra em vários *tokens*. Nessa fase, dois *tokens* especiais são adicionados as sentenças: [CLS] é adicionado no início de cada sequência e [SEP] é adicionado no fim, com objetivo de distinguir sequências distintas. É gerada uma matriz de codificação, composta por valores únicos para representar cada *token*. A representação da entrada pode ser vista na Figura 4. É a soma das matrizes de codificação, posicional e de segmentação, que é responsável por identificar a qual sentença o *token* pertence.

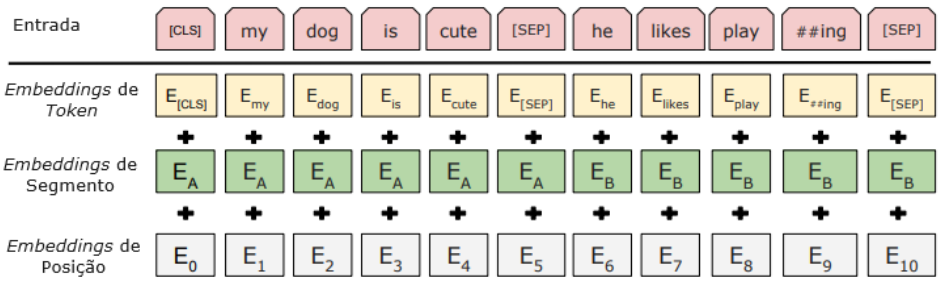

Figura 4 – Representação de entrada do BERT. Os *embeddings* de entrada são a soma dos *embeddings* de *token*, os *embeddings* de segmentação e os *embeddings* de posição. Fonte: (DEVLIN et al., 2019)

O codificador do BERT é uma pilha de codificadores similar aos da rede *transformer*. A distinção está na forma de processar a entrada, enquanto o *transformer* processa as entradas da esquerda para a direita, o BERT processa nos dois sentidos. A etapa de pré-treino é dividida em duas tarefas: MLM (*Masked Language Model*) e NSP (*Next Sentence Prediction*) (DEVLIN et al., 2019).

No MLM, uma porcentagem aleatória de *tokens* da sequência serão mascarados. Assim, partindo do conhecimento das outras palavras, o modelo tentará prever as palavras mascaradas. É um passo necessário, visto que na abordagem bidirecional, a falta dessa etapa faria com que um *token* visse a si próprio e o modelo poderia trivialmente prever a palavra alvo em um contexto de várias camadas. Ao final, ele obtém um vetor com as possíveis palavras e a probabilidade de cada uma ser a correta. Na NSP, a tarefa é prever se determinada sentença B é a sentença que vem após a sentença A em um texto. O objetivo é treinar um modelo que entenda a relação entre sentenças, fator importante para tarefas de QA e NLI (*Natural Language Inference*) (DEVLIN et al., 2019).

Na etapa de ajuste fino será feita a adequação de entradas e saídas ao problema que se deseja resolver. Em QA, por exemplo, a entrada pode ser um par de sentenças, composta pela pergunta e pelo texto que contém a resposta; Já em NLI, ela pode ser um par (hipótese, premissa) (DEVLIN et al., 2019). As etapas de pré-treino e ajuste-fino estão representadas na Figura 5.

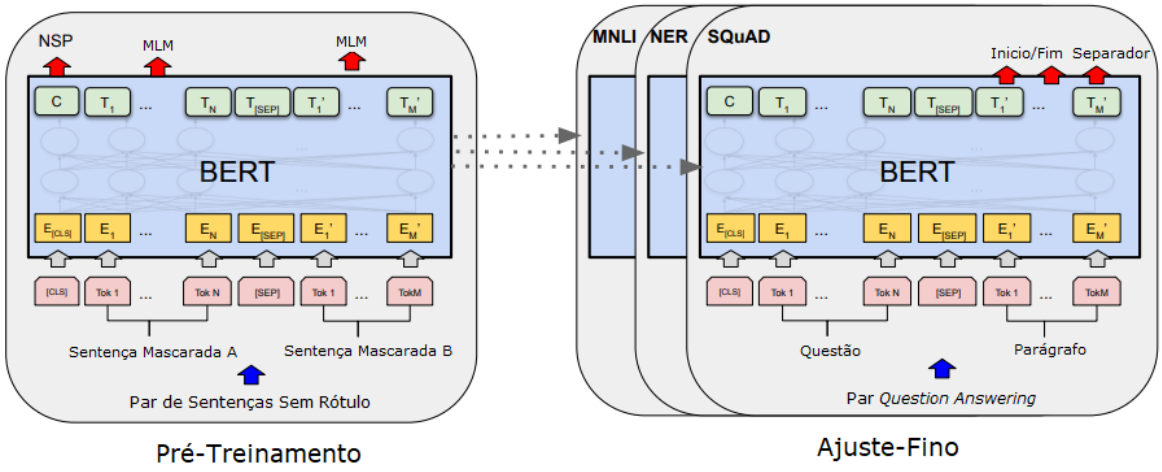

Figura 5 – A mesma arquitetura é usada no pré-treinamento e no ajuste-fino. Os mesmos parâmetros pré-treinados são usados na inicialização de diferentes tarefas de interesse. Durante o ajuste-fino, todos os parâmetros são ajustados. O formato de entrada dos dados depende da tarefa e do *dataset* escolhido. (DEVLIN et al., 2019)

Por ser pré-treinado com um *corpus* gigantesco e usar o mecanismo de auto-atenção em ambos os sentidos, que adiciona a característica de reconhecimento de contexto, o modelo BERT deve melhorar a capacidade dos *chatbots* no reconhecimento e classificação dos diálogos do usuário. O BERT obteve resultados estado da arte em onze tarefas de NLP,

incluindo QA, com um F1-*score* de 93,2 no SQuAD (*Stanford Question Answering Dataset*) v1 (DEVLIN et al., 2019). Uma vantagem de modelos pré-treinados é que devido a etapa de pré-treino, eles não precisam de muitos dados de exemplo no ajuste-fino para terem boa performance, ou até essa etapa pode ser pulada, dependendo da similaridade dos dados. Espera-se que os dados de ambas as etapas estejam no mesmo idioma. Para *chatbots* que não respondem em inglês, pode-se usar o modelo *Multilingual BERT* (*mBERT*).

2.3 DIET

O *Dual Intent and Entity Transformer* (DIET) é uma arquitetura multitarefa para classificação de intenções e reconhecimento de entidades. Sua característica chave é a habilidade de incorporar *embeddings* (representações em formato numérico de forma que um computador possa utilizá-los) de palavras de modelos pré-treinados e combiná-los com representações esparsas de palavras e atributos a nível de caractere (*character level n-gram features*) de uma maneira conecte e use (*plug-and-play*) (BUNK et al., 2020).

A arquitetura do DIET é modular e pode ser vista na Figura 6. A primeira tarefa é o pré-processamento dos dados de entrada. Nela são gerados os atributos necessários para o módulo *transformer*. O módulo *Embedding* Pré-treinado (*Pretrained Embedding*) fornece uma das entradas do *transformer*, que contém *tokens* de um determinado *embedding*. É um módulo *plug-and-play*, pois pode usar os *embeddings* pré-treinados de diversos modelos, como BERT, ConveRT, entre outros; e também pode ser removido da arquitetura, caso não deseje usá-lo.

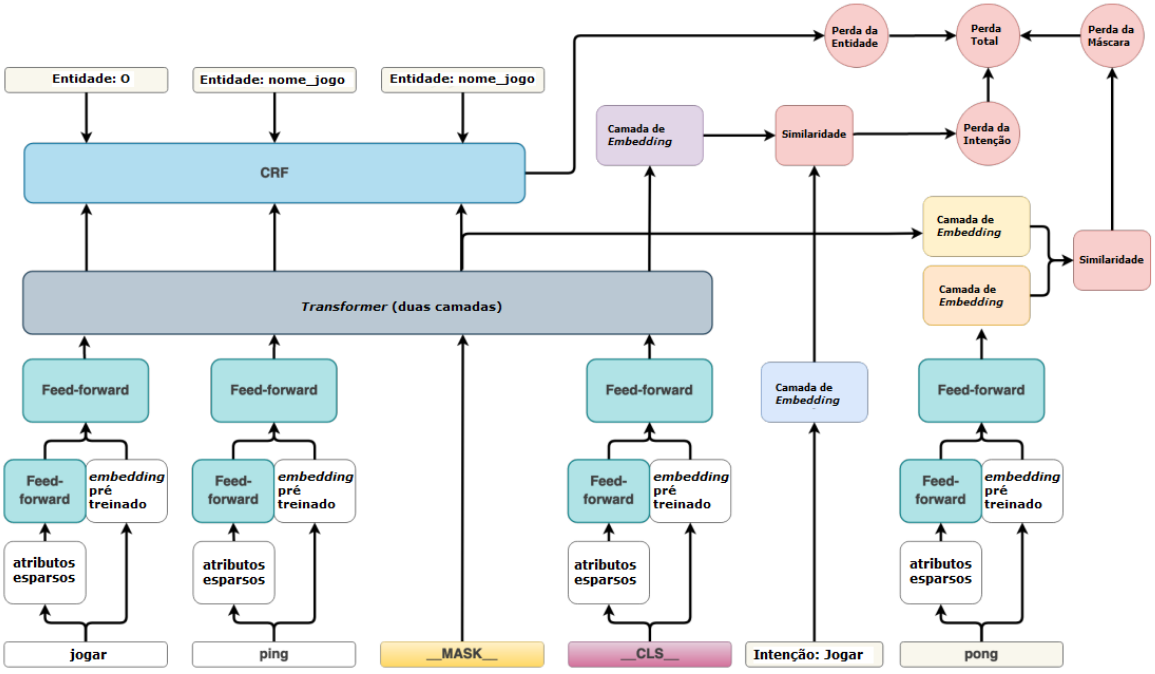

Figura 6 – Arquitetura DIET (BUNK et al., 2020)

O módulo atributos esparsos (*Sparse features*) é essencialmente uma rede *Feed Forward* com conexões esparsas, que gera atributos esparsos para os *tokens* em treinamento. A ideia é que, além dos *embeddings* gerados pelo Embedding Pré-treinado, mais atributos sejam adicionados, à medida que os dados de treinamento passam pelo modelo, de modo a fornecer mais informações do *token* ao *transformer*.

O módulo *Feed Forward Network* (FFN) é uma rede neural totalmente conectada (*fully connected neural network*) que aparece na arquitetura em dois momentos. Na primeira, a entrada são atributos esparsos de algum *token*; Na segunda, as entradas são atributos densos de *embeddings* de um modelo pré-treinado, além da saída da primeira FFN. Em ambos os casos, o objetivo é aprender atributos sobre *tokens* que serão usados posteriormente como entradas do módulo *Transformer*. As entradas de atributos esparsos são matrizes esparsas, e para tornar o modelo leve, 80% das suas conexões são desligadas por padrão.

O *token* mascarado é aplicado aleatoriamente a uma palavra da sentença. Ele é conectado ao *transformer*, que permite prever qual *token* foi mascarado e, em seguida, permitir o aprendizado por meio do gradiente descendente. O *token* CLS é uma representação de toda a sentença, similar a um *embedding* de toda a sentença, e provê a capacidade de aprender a intenção da sentença ao *transformer*.

Para codificar o contexto em toda a frase, usa-se um *transformer* de duas camadas com auto-atenção de posição relativa, uma maneira de usar as distâncias entre pares de *tokens* como forma de criar codificações posicionais. Ele recebe os atributos processados dos *tokens* e como saída, para cada *token* diz se é uma entidade ou não, e qual é a palavra mascarada. A tarefa de classificação de intenções é feita durante o treinamento por basicamente dois módulos: Camada de *Embedding* (*Embedding Layer*) e Perda de Intenção (*Intent Loss*). O Camada de *Embedding* é usado no treinamento, para gerar a Perda de Máscara (*Mask Loss*), associada à máscara gerada pelo modelo. Isso basicamente gera uma medição de perda entre o *token* previsto pela saída dos *transformers* e o gerado pelo FFN, que usa entradas de *embeddings* pré-treinados e os módulos de atributos esparsos do *token*. O módulo Perda de Intenção (*Intent Loss*) calcula a perda da saída do *transformer* e a classificação de intenção da sentença, dada pelo módulo Similaridade (*Similarity*). Sua saída conecta a entrada do módulo Perda Total (*Total Loss*), que fornecerá uma medida de aprendizado (SELLANES, 2022).

A tarefa de classificação de entidades é feita por dois módulos durante o treinamento: *Conditional Random Field* (CRF) e Perda de Entidade (*Entity Loss*). O CRF calcula a perda entre as entidades reconhecidas pela *Natural Language Understanding* (NLU) e a saída do *transformer*. Essa perda mede a capacidade do *transformer* de reconhecer entidades relacionadas à sentença de entrada. A saída do CRF conecta-se à Perda de Entidade, que calculará a medida de quão bom o *transformer* está identificando entidades

na frase.

Existem quatro módulos relacionados ao cálculo de perdas na arquitetura DIET: Perda de Entidade (*Entity Loss*), Perda de Máscara (*Mask Loss*), Perda de Intenção (*Intent Loss*) e Perda Total (*Total Loss*). O objetivo é o mesmo: calcular a fórmula para cada perda específica as suas entradas e gerar o valor da perda. A ideia é que essas diferentes perdas desencadeiem modificações de pesos por meio de gradiente descendente em diferentes etapas ou iterações, quando alguma parte do *pipeline* comete um erro, a fim de corrigir e melhorar seu desempenho. Esses módulos são essenciais para o treinamento e fornecem uma das características mais importantes do modelo DIET: a capacidade de inferir intenções e reconhecer entidades ao mesmo tempo.

Por exemplo, se o *transformer* não conseguiu reconhecer alguma entidade, isso será refletido na Perda de Entidade gerada a partir da saída do módulo CRF, o que levará a um ajuste dos pesos do *transformer* e das camadas anteriores (FFN, *Embedding* Pré-Treinado). Como outro exemplo, se o *transformer* não conseguir prever corretamente a palavra mascarada, isso será refletido na Perda de Máscara calculada a partir do módulo Similaridade. Como resultado, à semelhança do exemplo anterior, o gradiente descendente cuidará disso, ajustando os pesos dos módulos envolvidos (SELLANES, 2022).

Uma etapa de ajuste fino é feita com exemplos de frases relacionados as intenções correspondentes. Finalizado esse passo, têm-se um modelo treinado, que receberá como entrada uma sentença, e como saída a intenção da sentença (uma classe dentre as listadas durante o ajuste fino) e as entidades reconhecidas nela.

Como visto, uma das vantagens do DIET é o de resolver de uma só vez duas tarefas de bastante importância no desempenho de um *chatbot*: classificação de intenções e reconhecimento de entidades. Além disso, possibilita a incorporação de palavras (*embeddings*) de diversos modelos pré-treinados; como também possui uma camada *transformer* na sua arquitetura, dispensando o uso desses modelos, caso a economia de processamento seja um fator relevante. Sem usar nenhum *embedding* pré-treinado, o DIET ainda pode alcançar desempenho competitivo, superando o estado da arte nos testes com *NLU-Benchmark dataset* (BUNK et al., 2020).

2.4 LaBSE

O *Language Agnostic BERT Sentence Embeddings* (LaBSE) (FENG et al., 2022) possui suporte a 112 idiomas, oito a mais que o M-BERT. No LaBSE, o M-BERT foi adaptado para produzir incorporações de sentenças agnósticas de linguagem. Em modelos desse tipo, as representações vetoriais de sentenças semanticamente semelhantes estarão mais próximas, mesmo que em idiomas diferentes. Isso significa que a representação vetorial

de uma determinada frase em inglês estaria mais próxima da representação vetorial de sua tradução em hindi, do que a representação de qualquer outra frase diferente em inglês/hindi. Através dessa abordagem pode-se treinar um classificador com dados de treinamento em inglês e usá-lo diretamente para outro idioma como o alemão (MALHOTRA, 2021).

Na Figura 7 é apresentada a arquitetura do modelo. Além do uso do $BERT_{BASE}$, ela combina os melhores métodos para aprendizagem de representações monolíngue e multilíngue incluindo:

- MLM e *Translation Language Modeling* (TLM): São combinados na etapa de pré-treino. TLM é uma extensão do MLM para problemas multilíngues, que modifica o treinamento MLM ao incluir pares de tradução concatenados. Como visto na Figura 8, as palavras são mascaradas aleatoriamente, tanto na sentença fonte, quanto na sentença alvo. Para prever uma palavra mascarada em uma frase em inglês, o modelo pode se atentar às palavras em inglês ao redor ou à tradução em francês, incentivando o modelo a alinhar as representações em inglês e francês. Em particular, o modelo pode aproveitar o contexto francês, se o inglês não for suficiente para inferir as palavras inglesas mascaradas. Para facilitar o alinhamento, também foram redefinidas as posições das frases-alvo (CONNEAU; LAMPLE, 2019).

- *Dual Encoder Translation Ranking* (GUO et al., 2018): Os modelos de codificador duplo são uma abordagem eficaz no aprendizado de incorporações multilíngues. Consistem de codificadores emparelhados que alimentam uma função de pontuação. As sentenças fonte e alvo são codificadas separadamente. As incorporações multilíngues são treinadas usando uma tarefa de classificação de tradução com amostragem negativa em lote. Nessa tarefa, são usadas frases negativas aleatórias com frases negativas cuidadosamente selecionados que desafiam o modelo a distinguir entre pares de tradução verdadeiros e pares sem tradução que exibem algum grau de semelhança semântica, como exemplificado na Figura 9. O treinamento maximiza a pontuação do produto escalar de pares de sentenças, que são traduções umas das outras em detrimento dos negativos amostrados.

- Margem Aditiva *Softmax* (YANG et al., 2019a): estende a função de pontuação introduzindo margem em torno de pares positivos. Como visto na Figura 10, a margem melhora a separação entre traduções e não-traduções próximas.

- *Cross-Accelerator Negative Sampling*: Modelos de incorporação multilíngue treinados com amostras negativas em lote se beneficiam de treinamento com grandes tamanhos de lote. Modelos como BERT são limitados a pequenos tamanhos de lote devido a restrições de memória. Enquanto o paralelismo de dados nos permite aumentar o tamanho do lote global usando vários aceleradores, o tamanho do lote em núcleos

individuais permanecem pequenos. Nessa estratégia, cada núcleo codifica suas sentenças atribuídas e, em seguida, as representações de sentenças codificadas de todos os núcleos são transmitidos como negativos para outro núcleo. Isso nos permite perceber plenamente os benefícios de tamanhos de lote maiores, enquanto ainda distribui o trabalho de codificação computacionalmente intensivo em vários núcleos (FENG et al., 2022).

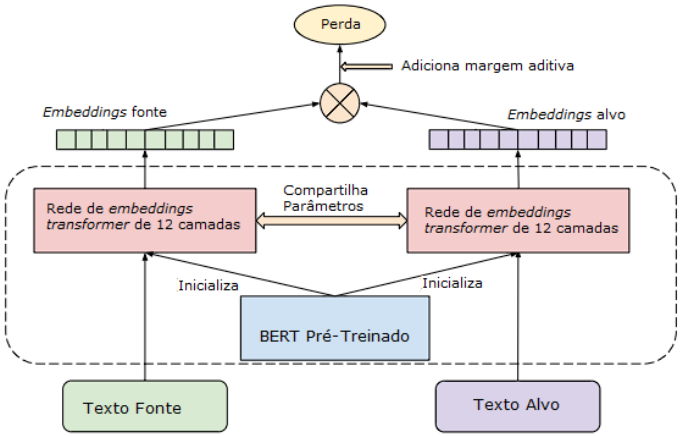

Figura 7 – Arquitetura LaBSE. Fonte: (FENG et al., 2022)

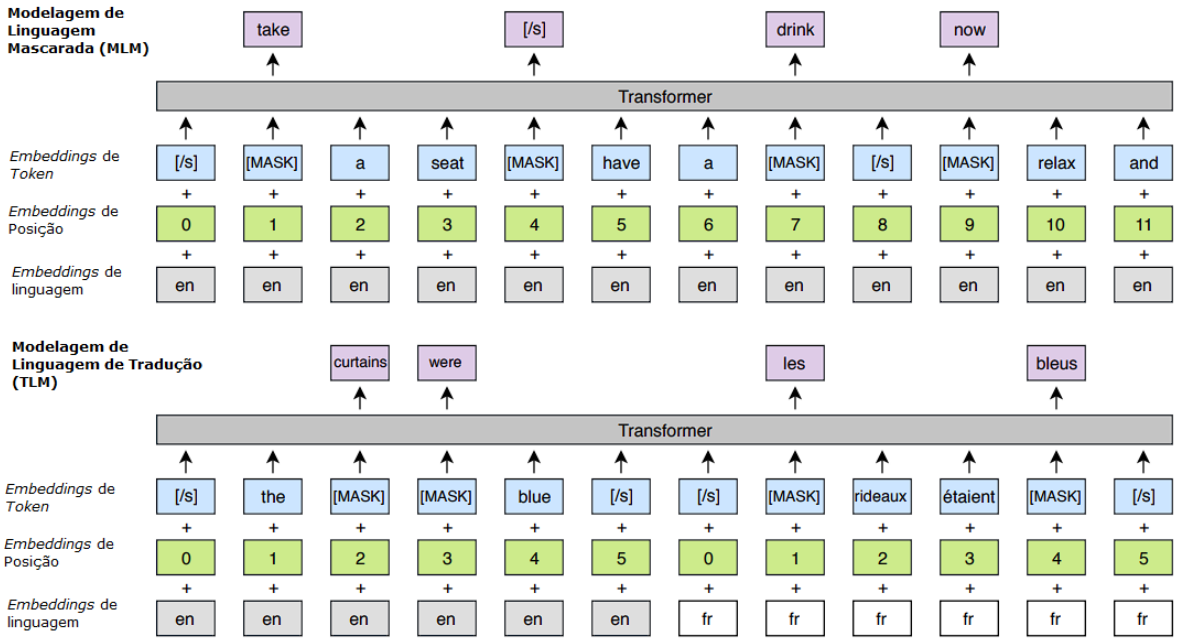

Figura 8 – *Masked and Translation Language Modeling*. Fonte: (CONNEAU; LAMPLE, 2019)

O LaBSE estabelece um novo estado da arte em tarefas de recuperação de texto paralelo múltiplo (também conhecido como bi-texto). Nos testes, o modelo BASE e LARGE

en-es	Oil and gas investments (Inversiones en petróleo y gas)	**Aleatória**	Alquiler mensual desde : 890 USD ¿Qué más se deja para preguntar?
		Difícil	Petróleo y gas Petróleo y Gas Petroquímica página
	In Spain, it has clearly chosen the gratuity (En España, se ha elegido claramente la gratuidad)	**Aleatória**	Ve el perfil completo de Fleishman León de montaña en roca
		Difícil	Dejar propina es una costumbre chilena Este es un típico restaurante español de España

Figura 9 – Tarefa de classificação de tradução com amostragem negativa em lote. Fonte: (GUO et al., 2018)

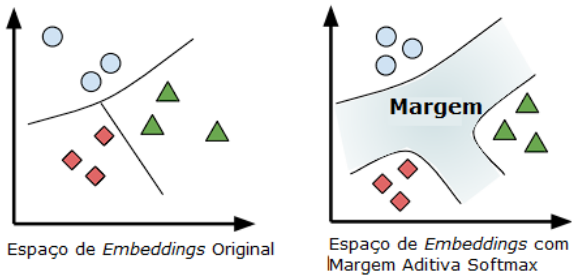

Figura 10 – Margem aditiva *softmax*. Fonte: (YANG et al., 2019a)

obtiveram pontuações bastante semelhantes, e devido a isso apenas a versão BASE foi liberada ao público.

2.5 BERTimbau

O BERTimbau possui mesma arquitetura e parametrização dos modelos $BERT_{BASE}$ e $BERT_{LARGE}$. Seu diferencial é ser um modelo treinado com uma base de dados totalmente em português, a base brWAC (*Brazilian Web as Corpus*) (FILHO et al., 2018), composta por 2,7 bilhões de *tokens*, é o maior *corpus* em língua portuguesa disponível até então. O modelo foi avaliado em três tarefas:

- Similaridade Textual de Sentença (*Sentence Textual Similarity* - STS): tarefa de regressão que mede o grau de equivalência semântica entre duas sentenças em uma escala numérica.
- Correlação Textual (*Recognizing Textual Entailment* - RTE): tarefa de classificação com objetivo de prever se uma determinada sentença de premissa implica em uma sentença de hipótese.
- Reconhecimento de Entidades (*Named Entity Recognition* - NER): consiste em identificar trechos de texto que mencionem entidades nomeadas e classificá-los em categorias pré-definidas.

O BERTimbau melhora o estado da arte nessas tarefas em relação ao BERT Multilíngue (mBERT) e abordagens monolíngues anteriores sob o Coeficiente de correlação

de Pearson para STS e F1 para RTE e NER, confirmando a eficácia de modelos pré-treinados para o português (SOUZA; NOGUEIRA; LOTUFO, 2020).

2.6 DistilBERT

A Destilação (*Knowledge distillation*) (HINTON; VINYALS; DEAN, 2015) é uma técnica de compressão na qual um modelo menor, denominado aluno, é treinado para reproduzir o comportamento de um modelo maior, denominado professor. O DistilBERT é uma versão destilada do BERT, sendo menor, mais leve, mais rápido e menos custoso que ele (SANH et al., 2019). Ele tem 40% menos parâmetros que o $BERT_{BASE}$, executa 60% mais rápido, enquanto preserva 97% do desempenho da capacidade de entendimento de linguagem do BERT, medido no teste de compreensão de linguagem GLUE.

O modelo segue a arquitetura original do BERT, porém com número de camadas reduzido por um fator de dois, além da remoção do *pooler*, a última camada usada nas tarefas de interesse (*downstream*), e dos *embeddings* de segmentação, usando apenas o *token* [SEP] para indicar a separação entre sentenças. Para aproveitar os vieses indutivos aprendidos por modelos maiores durante o pré-treinamento, foi introduzida uma perda tripla que combina perdas da MLM, da destilação e da distância do cosseno (SANH et al., 2019).

O DistilBERT segue as melhores práticas propostas em LIU et al. (2019): ele é destilado em lotes muito grandes de sentenças, usando mascaramento dinâmico e sem o treinamento de NSP. O treinamento usando lotes maiores melhora a perplexidade na tarefa de MLM, bem como a acurácia nas tarefas de interesse. Grandes lotes também são mais fáceis de paralelizar por meio de treinamento paralelo de dados distribuídos. No mascaramento dinâmico, o padrão de mascaramento é alterado toda vez que uma sequência é alimentada ao modelo, diferente do estático que executa o mascaramento apenas no pré-processamento dos dados. O mascaramento dinâmico é comparável ou ligeiramente melhor que o mascaramento estático (SANH et al., 2019).

2.7 RoBERTa

O *Robustly Optimized BERT Pretraining Approach* (RoBERTa) é uma replicação do estudo de pré-treinamento do BERT, porém com uma avaliação mais cuidadosa sobre o impacto da escolha dos hiper-parâmetros e do tamanho dos dados de treinamento. "Descobrimos que o BERT foi significativamente sub-treinado e que ele pode igualar ou ultrapassar o desempenho de todos os modelos publicados depois dele" (LIU et al., 2019).

As mudanças incluem treinar o modelo por mais tempo, com lotes maiores e sob mais dados; remover a tarefa de NSP; treinar com sequências mais longas e usar mascaramento

dinâmico. O RoBERTa também usou um *corpus* maior no pré-treinamento e confirmou que o uso de mais dados melhora ainda mais o desempenho em tarefas de interesse. Na *tokenização*, ele usa um *Byte-Pair Encoding* (BPE) a nível de *byte*, um híbrido entre representações a nível de caractere e de palavra. Em vez de palavras completas, o BPE conta com unidades de sub-palavras, que são extraídas por meio da análise estatística do *corpus* de treinamento. O BERT original usa BPE a nível de caractere. O RoBERTa superou resultados estado da arte em diversas tarefas dos *benchmarks* GLUE, SQuAD e RACE (LIU et al., 2019).

2.8 GPT

No artigo RADFORD et al. (2018) observa que apesar de que muitos trabalhos tenham aplicado pré-treino e ajuste-fino a diversos problemas de NLP, ainda não existe consenso na maneira mais efetiva de se transferir as representações aprendidas às tarefas alvo (*transfer learning*). O *Generative Pre-Trained Transformer* (GPT) (RADFORD et al., 2018) explora uma abordagem semi-supervisionada para tarefas de NLU, combinando pré-treino não-supervisionado, usando pra isso grandes *corpus* de texto, com ajuste-fino supervisionado. Nesse último caso, ao invés de modificar a arquitetura para resolver determinada tarefa, ele opta por adaptar a estrutura dos dados de entrada dessas tarefas a arquitetura, chamado *traversal-style*.

No pré-treino, o GPT segue a arquitetura *transformer* original, mas ao contrário do BERT, que usa apenas codificadores, aqui são usados apenas decodificadores. São 12 decodificadores, matriz de codificação com 768 colunas e 12 cabeças de auto-atenção. Na Figura 11 temos uma representação visual da arquitetura do modelo, à esquerda, a fase de pré-treino representada pela arquitetura *transformer* multi-camadas de codificadores; à direita, as transformações de entrada para ajuste-fino de diferentes tarefas.

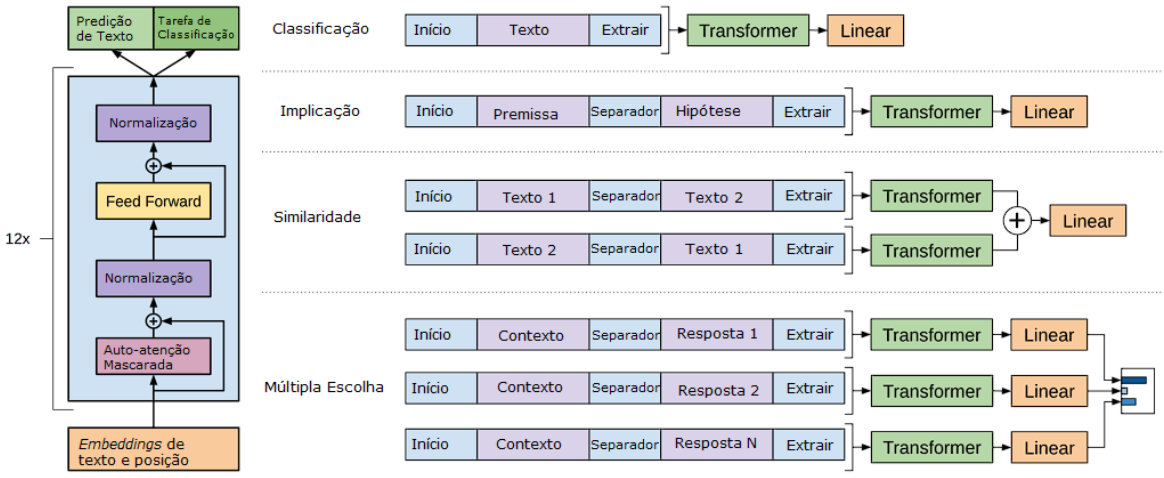

Figura 11 – Arquitetura GPT. Fonte: (RADFORD et al., 2018)

O modelo geral, independente de tarefa, supera os modelos que empregam arquiteturas especificamente criadas para cada tarefa, melhorando significativamente o estado da arte em 9 das 12 tarefas testadas.

2.9 GPT-2

O GPT-2 (RADFORD et al., 2019) quando comparado ao GPT possui as seguintes diferenças: inicialização de peso diferente; dez vezes mais parâmetros; maior vocabulário; sequência de entrada mais longa; sem necessidade de ajuste-fino. O modelo melhorou o estado da arte para 7 de 8 conjuntos de dados de modelagem de linguagem na configuração de *zero shot*. *Zero-shot learning* é quando os dados da etapa de validação não estavam presentes na etapa de treinamento.

O GPT-2 mostrou que o treinamento em um conjunto de dados maior e com mais parâmetros melhorou a capacidade do modelo ao entender tarefas e superar o estado da arte de muitas delas em configurações *zero shot* (RADFORD et al., 2019).

Apesar de já existirem versões mais novas do GPT, o Rasa ainda não possui suporte à elas. Por isso, esse trabalho se limitará a usar apenas os modelos GPT e GPT-2 nos seus experimentos.

2.10 XLNet

O XLNet (YANG et al., 2019b) combina o modelo estado da arte de linguagem auto-regressivo, Transformer-XL (DAI et al., 2019), com a capacidade bidirecional do BERT. Ele aproveita o melhor dos modelos de linguagem auto-regressivos (AR - *autoregressive*) e auto-codificados (AE - *autoencoder*), enquanto evita as limitações de ambos.

Nos modelos auto-regressivos, os *tokens* são lidos em uma única direção, seguindo a ideia de adivinhar o próximo *token* depois de ler todos os anteriores. Como exemplo, os modelos GPT e GPT-2; Nos modelos auto-codificados, os *tokens* de entrada são alterados, e eles tentam reconstruir a sentença original. Por exemplo, o BERT que mascara *tokens* aleatórios na entrada (BI et al., 2020).

Tanto BERT quanto XLNet realizam predição parcial, ou seja, preveem apenas um subconjunto de *tokens* na sequência. A previsão parcial desempenha um papel na redução da dificuldade de otimização, prevendo apenas *tokens* com contexto suficiente. Segundo DAI et al. (2019), para características de dependência de alta ordem e longo alcance em linguagem natural, o BERT simplifica demais o problema, assumindo que os *tokens* mascarados são independentes uns dos outros. Já o XLNet é capaz de capturar as dependências entre *tokens* mascarados (YANG et al., 2019b).

O XLNet é um Modelo de Linguagem de Permutação. Em vez de usar uma ordem fixa de fatoração direta, ou reversa, como nos modelos AR convencionais, ele realiza todas as permutações possíveis de determinada ordem de fatoração. Graças à operação de permutação, o contexto de cada posição captura *tokens* da esquerda e da direita. Assim, cada posição aprende a utilizar informações contextuais de todas as outras posições, capturando o contexto bidirecional (YANG et al., 2019b).

Sua arquitetura é a *Two-Stream Self-Attention for Target-Aware Representations*, que conta com dois diferentes tipos de auto-atenção: *Content stream*, similar a auto-atenção tradicional, codifica contexto e o *token* alvo; e a *Query stream*, que codifica apenas a posição. O XLNet também incorpora duas importantes técnicas do Transformer-XL no pré-treinamento: o esquema de codificação posicional relativa e o mecanismo de recorrência de segmento. Elas melhoram empiricamente o desempenho do modelo especialmente para tarefas que envolvem sequências de texto mais longas (YANG et al., 2019b).

Segundo YANG et al. (2019b), existem dois benefícios em usar codificações de segmento relativas: melhorar a generalização e abrir a possibilidade de ajuste fino em tarefas que possuem mais de dois segmentos de entrada, o que não é possível usando codificação de segmentos absoluta. Já quanto ao mecanismo de recorrência, a sequência de estados ocultos calculada para o segmento anterior é armazenada em *cache* para ser reutilizada como um contexto estendido quando o modelo processar o próximo novo segmento.

O modelo XLNet-Large possui os mesmos hiperparâmetros de arquitetura do BERT-Large, o que resulta em um tamanho de modelo semelhante. Ele alcança melhorias substanciais, e empiricamente, supera o BERT em 20 tarefas, muitas vezes por uma grande margem, incluindo QA, inferência de linguagem natural, análise de sentimento e classificação de documentos (YANG et al., 2019b).

2.11 Ferramentas

2.11.1 Rasa

O Rasa é uma ferramenta *open-source* para desenvolvimento de *chatbots*. Sua arquitetura modular permite integração de seus componentes com outros sistemas. Ele pode ser dividido em dois componentes principais: Rasa NLU (*Natural Language Understanding*), para compreensão de linguagem natural; e Rasa Core, para o gerenciamento de diálogo (BOCKLISCH et al., 2017).

A ferramenta atende ao público mais iniciante, que apenas querem desenvolver um *chatbot* funcional, e ao mais especializado, já que o Rasa possui uma documentação bastante explicativa de como seus componentes funcionam e das técnicas que cada um deles usa. Cada componente possui parâmetros configuráveis. Assim, o desenvolvedor

pode definir quais componentes utilizar, em que momento serão utilizados, e como estarão configurados no seu *chatbot*. Alguns desses componentes são apresentados na Seção 4.3.

O gerenciamento de diálogo pode ser encarado como um problema de classificação. A cada iteração, Rasa Core prevê qual ação tomar a partir de uma lista de ações predefinida. Uma ação pode ser um simples enunciado, ou seja, enviando uma mensagem para o usuário, ou pode ser uma função arbitrária a ser executada. O estado da conversa é salvo durante a sessão, e uma ação pode fazer uso de qualquer informação coletada durante a sessão, como entidades, enunciados anteriores e resultados de ações anteriores (BOCKLISCH et al., 2017).

A classificação de intenção do Rasa NLU é feita a partir de algoritmos de aprendizado de máquina, consistindo de métodos de classificação de NLP e pode ser auxiliado por métodos de modelagem e tokenização.

Tanto o Rasa NLU quanto o Core trabalham com formatos de dados de treinamento legíveis por humanos. Rasa NLU requer um lista de enunciados anotados com intenções e entidades. Estes podem ser especificados como uma estrutura *json* ou em formato de marcação. Seguem alguns conceitos:

- *Intent*: Intenção da mensagem. O desenvolvedor cadastra uma lista de intenções com exemplos, e o Rasa NLU tentará classificar a intenção da mensagem do usuário, associando-a a alguma intenção predefinida. São definidas no arquivo nlu.yml e podem ser vistas na Figura 12.
- *Entity*: Cada entidade engloba exemplos de palavras que são importantes no reconhecimento do contexto da conversa, como visto na Figura 13.
- *Stories*: caminhos de diálogo já pré-definidos pelo desenvolvedor, como mostrado na Figura 14.
- *Rules*: Como o *bot* deve responder ao usuário após identificar a intenção. Dentro do arquivo domain.yml são definidas quais intenções estão ativas na sessão, as respostas pra cada uma delas, além de configurações do tempo de duração máximo da sessão. Dentro do arquivo rules.yml ficam definidas as *actions*, que é o relacionamento entre a intenção e sua resposta, e pode ser visto na Figura 15.
- *Pipeline*: Define quais componentes serão usados e a função em que eles serão aplicados. Fica dentro do arquivo config.yml e um exemplo da sua estrutura é mostrado na Figura 16.

```
intent: int6
examples: |
  - qual a ferramenta para solicitação informação
  - qual a ferramenta para pedido conhecimento
  - qual a ferramenta para súplica referência
  - qual a ferramenta para rogo noção
  - qual a ferramenta para imploração ideia
  - qual a ferramenta para insistência base

intent: int7
examples: |
  - Xavier - e-sic - - 27764435000169 , usuário está tentando acesso e não consegue .
  - Xavier - e-sic - - 27764435000169 , usuário está procurando acesso e não atinge .
  - Xavier - e-sic - - 27764435000169 , usuário está buscando acesso e não chega .
  - Xavier - e-sic - - 27764435000169 , usuário está intentando acesso e não conquista .
  - Xavier - e-sic - - 27764435000169 , usuário está pretendendo acesso e não realiza .
  - Xavier - e-sic - - 27764435000169 , usuário está realizando acesso e não cumpre .
```

Figura 12 – Lista de Intenções

```
nlu:
- intent: localizacao_setor
  examples: |
    - Qual o endereço da [CAJ](setor)

responses:
  utter_localizacao_setor:
  - text: "O endereço da (setor) é na Av. Presidente Getúlio Vargas"
```

Figura 13 – Lista de Entidades. No exemplo, a CAJ é um exemplo da entidade setor. Ao detectar a entidade, o *chatbot* responde recuperando certas palavras da entrada do usuário, tornando o diálogo mais amigável.

```
stories:
- story: quero o numero de telefone
  steps:
  - user: |
      bom dia
    intent: greet
  - action: utter_greet
  - user: |
      gostaria de saber o telefone do gabinete do senhor Thompson
    intent: contato_carlos_thompson
  - action: utter_contato_carlos_thompson
  - user: |
      também preciso do número da CAJ
    intent: contato_caj
  - action: utter_contato_caj
  - user: |
      obrigado
    intent: goodbye
  - action: utter_goodbye
```

Figura 14 – Lista de *stories*

```
rules:

- rule: int0
  steps:
  - intent: int0
  - action: utter_int0

- rule: int1
  steps:
  - intent: int1
  - action: utter_int1
```

Figura 15 – Lista de *rules*

```yaml
pipeline:
- name: WhitespaceTokenizer
- name: RegexFeaturizer
- name: CountVectorsFeaturizer
- name: CountVectorsFeaturizer
  analyzer: char_wb
  min_ngram: 1
  max_ngram: 4
- name: DIETClassifier
  epochs: 100
  constrain_similarities: true
- name: EntitySynonymMapper
- name: FallbackClassifier
  threshold: 0.7
  ambiguity_threshold: 0.1
```

Figura 16 – Configuração do *pipeline*

2.11.2 Venom Bot

O Venom Bot é uma biblioteca *open-source* desenvolvida em JavaScript com foco na criação de *chatbots* para WhatsApp (ORKESTRAL, 2022). Nesse trabalho, o Rasa é a ferramenta de desenvolvimento da inteligência do *bot*, porém a comunicação entre o Rasa e o WhatsApp é gerenciado por meio de uma aplicação NodeJS usando o Venom Bot. Ao iniciar, o Venom Bot apresenta um *QR Code* que deve ser escaneado a partir de um celular logado no Whatsapp. A partir desse momento, toda a troca de mensagens feita nesse celular será interceptada pelo Venom Bot.

A biblioteca possui suporte a diversos recursos do WhatsApp, como envio de textos, áudio, vídeo, documentos, localização, botões, *stickers*, contatos, entre outros. Por si só, ela não possui os recursos para desenvolvimento de um *chatbot*, mas o seu objetivo não está na criação do *bot*, mas sim em ser uma intermediadora da troca de mensagens entre o WhatsApp e o *bot*.

2.12 Considerações Finais

Nesse capítulo foi apresentado uma visão geral de todos os modelos *transformers* suportados pelo Rasa até o momento. O objetivo é destacar as características de cada um, para que na análise dos resultados possamos tentar compreender quais os motivos que poderiam levar um modelo a se sobressair em comparação aos outros. Como são muitas características a considerar, foi feita uma síntese dos principais pontos de atenção de cada modelo na Tabela 1.

Modelo	Características	Resultados
Transformer	Arquitetura base de todos os outros modelos; Unidirecional; Mais eficiente que as RNN's; primeiro modelo baseado unicamente em auto-atenção	Para tarefas de tradução, o *Transformer* pôde ser treinado significativamente mais rápido do que arquiteturas baseadas em recorrência ou convolução
BERT	Bidirecional; Pré-treinado com dois objetivos: MLM e NSP; Usa apenas codificadores	Supera o estado da arte em onze tarefas de NLP
DIET	Arquitetura multitarefa para *chatbots*; Possui seu módulo *transformer*, mas também pode incorporar outros modelos *transformers* em sua rede	Supera o estado da arte nos testes com o *dataset NLU-Benchmark*
LaBSE	Modelo agnóstico; Possui suporte a mais idiomas que o mBERT	Supera estado da arte em tarefas de bi-texto
BERTimbau	Treinado com o maior *corpus* em língua portuguesa	Supera estado da arte nas tarefas STS, RTE e NER
DistilBERT	Menor, mais leve, mais rápido e menos custoso que o BERT	Alcança em média 97% das pontuações do BERT
RoBERTa	Melhor calibrado que o BERT com mais dados e lotes maiores; Usa mascaramento dinâmico	Supera estado da arte em tarefas como QA
GPT	Usa apenas decodificadores	Melhora o estado da arte em 9 de 12 tarefas testadas
GPT-2	Mais parâmetros, maior vocabulário, maior sequência de entrada que o GPT	Melhora o estado da arte para 7 de 8 conjuntos de dados de modelagem de linguagem na configuração *zero shot*
XLNET	Introduz PLM, uma misto de MLM e auto-regressão; Captura dependências entre *tokens* mascarados; Bidirecional, através da permutação; Arquitetura com dois tipos de auto-atenção	Supera o BERT em 20 tarefas

Tabela 1 – Resumo dos modelos *transformers* apresentados.

3 REVISÃO DA LITERATURA

Este capítulo será dedicado a uma revisão da literatura sobre o tema, e apresentará os *chatbots* aplicados a diferentes contextos. Para a sua escrita, os artigos foram obtidos na base Scopus da Elsevier. As palavras chaves escolhidas foram *chatbot*, *transformers* e *question answering*, filtrando apenas artigos, de 2018 em diante. A busca retornou 16 resultados, que foram ordenados dos mais aos menos citados. Foram lidos os resumos de cada um, e destes, 6 artigos foram lidos totalmente, como detalhado na Tabela 2.

Chave de busca	*chatbot AND transformers AND question AND answering*
Ano	a partir de 2018
Tipo de documento	apenas artigos
Total de artigos	16
Total de resumos lidos	16
Total de artigos lidos	6

Tabela 2 – Revisão bibliográfica.

Em PALASUNDRAM et al. (2021), é apresentada uma nova abordagem de aprendizagem multi-tarefa (MTL - *Multi-task learning)* para o modelo de aprendizagem *seq2seq* denominado SEQ2SEQ++, que compreende um codificador multifuncional, um decodificador de resposta, um codificador de resposta e um classificador ternário. SEQ2SEQ++ utiliza um mecanismo de peso de perda de tarefas dinâmicas para cálculo de perda de MTL e um novo mecanismo de atenção, denominado mecanismo de atenção abrangente. Experimentos com as bases de dados NarrativeQA e SQuAD foram conduzidos para avaliar o desempenho do modelo proposto em comparação com os modelos STL e MTL-BC. Os resultados experimentais mostraram que o SEQ2SEQ++ produz melhorias notáveis em relação aos dois modelos no substituto de avaliação bilíngue (*bilingual evaluation understudy*), taxa de erro das palavras e métricas Distinct-2.

Em GILSON et al. (2022), o ChatGPT é submetido ao exame de admissão de médicos nos EUA. O objetivo é analisar as respostas quanto à interpretabilidade do usuário. O desempenho do ChatGPT foi comparado com outros dois modelos de linguagem: GPT-3 e InstructGPT. A saída de texto de cada resposta foi avaliada através de 3 métricas qualitativas: justificativa lógica da resposta selecionada, presença de informações internas à pergunta e presença de informações externas à pergunta. O ChatGPT superou o InstructGPT em média em 8,15% em todos os conjuntos de dados, enquanto o GPT-3 teve um desempenho semelhante. O modelo demonstrou uma diminuição significativa no desempenho à medida que a dificuldade das perguntas aumentava. A justificativa lógica para a seleção de respostas do ChatGPT esteve presente em 100% dos resultados. A

informação interna à pergunta esteve presente em 96,8% de todas as respostas. A presença de informações externas à pergunta foi 44,5% no total e 27% menor para respostas incorretas em relação às respostas corretas. O autor conclui que o ChatGPT marca uma melhoria significativa nos modelos de processamento de linguagem natural nas tarefas de resposta a perguntas médicas. Ao atingir um limite superior a 60% de acertos, mostra-se que o modelo atinge o equivalente a uma pontuação de aprovação para um estudante de medicina do terceiro ano.

Em KOCOŃ et al. (2023), o autor examina as capacidades do ChatGPT em 25 diversas tarefas analíticas de NLP, a maioria delas subjetivas até mesmo para humanos, como análise de sentimento, reconhecimento de emoções, ofensiva e detecção de postura. Em contrapartida, as outras tarefas exigem raciocínio como desambiguação do sentido das palavras, aceitabilidade linguística e QA. As soluções mostraram que a perda média de qualidade do modelo ChatGPT foi de cerca de 25% para avaliação do tipo *zero-shot* (previsões para classes distintas das que estavam presentes na base de dados de treinamento) e *few-shot* (previsões para novas classes com base em poucos exemplos). Para o modelo GPT-4, a perda de tarefas semânticas é significativamente menor do que para ChatGPT. Viu-se que quanto mais difícil a tarefa, nesse caso as que tiveram menor pontuação estado da arte, maior será a perda do ChatGPT, dentre elas, problemas pragmáticos de NLP, como reconhecimento de emoções.

Em AU-YEUNG et al. (2023), é discutido o potencial e comparado o desempenho de duas diferentes abordagens para modelos *transformers* generativos: ChatGPT, o *Large Language Model* (LLM) de conversação geral mais utilizado, e o Foresight, um modelo baseado em GPT, focado na modelagem de pacientes e distúrbios. A comparação é realizada com a tarefa de prever diagnósticos relevantes com base em pequenos textos clínicos. Também são discutidos considerações e limitações importantes dos *chatbots* baseados em *transformers* para uso clínico. Ambos os modelos tiveram alto desempenho qualitativo, com desempenho ligeiramente superior do Foresight. Os médicos relataram que 21 das 35 respostas do ChatGPT apresentavam diagnósticos cruciais que estavam faltando, sendo mais superficial na predição de doenças de alto nível em vez de doenças específicas, algo que o Foresight consegue se sair melhor, produzindo sugestões mais específicas.

Em AHMED; KHAN; MUNIR (2023), o objetivo do artigo é apresentar uma arquitetura de *chatbot*, concentrando-se em problemas de aprendizagem *few-shot* e gerenciamento de contexto de conversas baseadas em diálogo. Primeiro, para mitigar o problema de aprendizado *few-shot*, é proposto um modelo híbrido de intenção e de *slots transformer* (HIST - *Hybrid Intent and Slots Transformers*). A arquitetura do *chatbot* HIST utiliza *transformers* e mecanismos de autoatenção junto com uma *Bigated Recurrent Unit* e combina algoritmos *Conditional Random Field* (CRF) para classificação de intenção e extração de entidades. Segundo, para abordar o gerenciamento de diálogo, é introduzida

uma estratégia de interação híbrida para mapeamento de *slots* e gerenciamento eficaz de contexto conversacional. Para validar a eficácia do modelo proposto, é realizada uma análise empírica abrangente utilizando três conjuntos de dados de referência, incluindo sistema de informação de viagens aéreas (ATIS), serviços bancários e interface de linguagem conversacional para conversação natural (CLINC150). Os resultados mostram que o HIST supera os métodos existentes de última geração com uma margem clara e obteve uma precisão de 94,89% e 96,17% para classificação de intenções e extração de *slots*, respectivamente. Os resultados empíricos confirmam a eficácia do *chatbot* HIST para resolver o problema de aprendizagem de poucas tentativas, com gerenciamento eficaz de diálogo em sistemas de *chatbot*.

Em Qiu et al. (2023), é proposta uma abordagem robusta de ponta a ponta que pode melhorar a eficiência e a eficácia da recuperação de consultas relacionadas aos termos de exploração mineral. Primeiro é construído um sistema automático de perguntas e respostas no domínio das geociências, que pode fornecer declarações relevantes que contenham respostas baseadas nas perguntas inseridas pelo usuário. O modelo BERT é treinado para testar as respostas geradas a partir da pergunta de entrada do usuário. Depois é desenvolvido um *chatbot* usando a plataforma *WeChat* usado nos testes. O BERT obteve melhores resultados que os outros modelos do comparativo, como QANET e SAN, obtendo uma pontuação F1 de 71,89.

Dos trabalhos listados, metade foca apenas em modelos *transformers* generativos, enquanto a outra metade usa tanto modelos *transformers* quanto modelos baseados em RNN. Neste trabalho optou-se por usar apenas modelos *transformers* generativos e discriminativos, e comparado aos trabalhos relacionados, foi o que envolveu o maior número de modelos no comparativo. Metade dos trabalhos avaliaram os modelos em diferentes tarefas NLP, enquanto na outra metade houve o enfoque em uma tarefa somente. Em grande parte deles o *chatbot* é apenas um meio usado na fase de testes e obtenção dos resultados, sendo descartado ao final; Neste trabalho, o enfoque foi na capacidade de entendimento dos modelos, e o *chatbot* além de ser utilizado nos testes, também é o produto final deste projeto.

3.1 Considerações Finais

Ao analisar os trabalhos apresentados, é notória a utilização de modelos *transformers* em diferentes áreas e temáticas no cenário acadêmico, como medicina, geografia, serviços bancários. Existe um grande interesse da comunidade no uso desses modelos, que têm obtido bons resultados inclusive em comparação com as RNNs. Porém, é visível que poucos modelos são envolvidos nesses comparativos, e o *chatbot* normalmente é somente um meio usado para testes. O presente trabalho envolve 9 modelos de diferentes arquiteturas

transformers, treinados sob a mesma base de dados e que serão avaliados na tarefa de classificação de intenções, tarefa chave no desenvolvimento de *chatbots*. O objetivo desse trabalho, além de avaliar os modelos, está no uso desse *software* no TCE, ou seja, aqui o desenvolvimento do *chatbot* está entre um dos principais objetivos dessa pesquisa. Na Tabela 3 é apresentado um resumo dos trabalhos presentes nesse capítulo.

Trabalho	Modelos	Base de Dados	Objetivo
(PALASUNDRAM et al., 2021)	SEQ2SEQ++, STL e MTL-BC	NarrativeQA e SQuAD	Avalia a qualidade da geração das respostas baseadas na diversidade (Distinct-2), taxa de erro (similaridade) das palavras e BLEU (*bilingual evaluation understudy*) que mede a qualidade de tradução da máquina quando comparado ao humano.
(GILSON et al., 2022)	ChatGPT, GPT-3 e InstructGPT	AMBOSS-Step1, AMBOSS-Step2, NBME-Free-Step1 e NBMEFree-Step2	Analisar a interpretabilidade das respostas.
(KOCOŃ et al., 2023)	ChatGPT e GPT-4	25 *datasets* públicos como CoLA, SQuAD e RACE	Avaliação dos modelos em 25 tarefas de NLP.
(AU-YEUNG et al., 2023)	ChatGPT e Foresight	MultiMedQA	Avaliar a previsão de diagnósticos a partir de textos clínicos.
(AHMED; KHAN; MUNIR, 2023)	HIST e LSTM	ATIS, CLINC150 e *datasets* de serviços bancários	Propõe uma arquitetura para problemas *few-shot*. Avaliação entre um modelo *transformer* e uma RNN através da classificação de intenções e reconhecimento de entidades.
(Qiu et al., 2023)	BERT, BiDAF, Match-LSTM, SAN, QANet	GeoQA2021	A base de dados é multilíngue e inclui pergunta, resposta e contexto. A avaliação dos modelos está na geração de respostas a partir do par (pergunta, contexto).
Trabalho proposto	DIET, LaBSE, BERTimbau, DistilBERT, RoBERTa, GPT, GPT-2 e XLNET	CAJ	Avaliação dos modelos quanto a classificação das perguntas do usuário.

Tabela 3 – Comparativo entre trabalhos da revisão da literatura.

4 DETALHAMENTO DO DESIGN E MATERIAL EXPERIMENTAL

Neste capítulo serão descritos o conjunto de dados de treinamento e as formas de validação a qual a solução será submetida, além da metodologia usada nos experimentos.

O presente trabalho é de pesquisa empírica e tem por finalidade produzir como artefato um *chatbot* que atuará no atendimento ao jurisdicionado dentro da CAJ. A pesquisa empírica, também chamada de pesquisa de campo, pode ser entendida como aquela em que é necessária comprovação prática de algo, seja através de experimentos ou observação de determinado contexto para coleta de dados em campo (ACADEMY, 2020).

O trabalho seguirá a metodologia *Design Science Research* (DSRM), proposta em PEFFERS et al. (2007), e apresentada visualmente na Figura 17. Nessa metodologia é definido um fluxo de seis fases pelas quais o trabalho deve se preocupar:

1. Identificação do problema e justificativa

   Implementar uma solução de atendimento tecnológica na CAJ se torna imprescindível sob a justificativa de que o aumento do número de sistemas do TCE implicará no aumento de atendimentos ao jurisdicionado, e o quadro de pessoal praticamente estacionado no órgão futuramente provocará um perda na qualidade desses atendimentos.

2. Definição dos objetivos

   - Analisar a viabilidade de implementação de um *chatbot* com integração ao WhatsApp, usando para isso somente tecnologias *open-source*.
   - Através da arquitetura DIET, promover um estudo comparativo na tarefa de classificação de intenções entre seu uso puro, sem *embeddings* pré-treinados, e seu uso com *embeddings* pré-treinados do BERT e outros modelos *transformers* estado da arte. Os modelos serão comparados seguindo as métricas descritas na Seção 4.1.

3. Design e desenvolvimento do artefato

   O *chatbot* será desenvolvido usando a plataforma Rasa. Em seu núcleo, um modelo *transformer* estado da arte em QA será treinado com dados capturados de atendimentos reais feitos pelo TCE. A sua integração com o WhatsApp será feito com o auxílio da biblioteca Venom Bot versão 4.2. Mais detalhes da solução estão descritas na Seção 4.4.

4. Demonstração e testes

   Nessa etapa, os dados serão submetidos a Validação Cruzada (CV - *Cross Validation*). A técnica será mais detalhada no Capítulo 5. Os resultados colhidos serão usados na etapa de Avaliação.

5. Avaliação

   Na avaliação, os dados obtidos na fase de testes serão usados no cálculo das métricas da subseção 4.1. Esta etapa estará concluída com a definição do modelo que será implantado no tribunal.

6. Compartilhamento da pesquisa

   Segundo PEFFERS et al. (2007), a pesquisa e o artefato precisam ser apresentados e comunicados de forma que possam servir a comunidade científica, interessada no assunto como um todo. A conclusão do presente trabalho, assim como sua divulgação nos meios acadêmicos serão suficientes para cumprir esse último requisito.

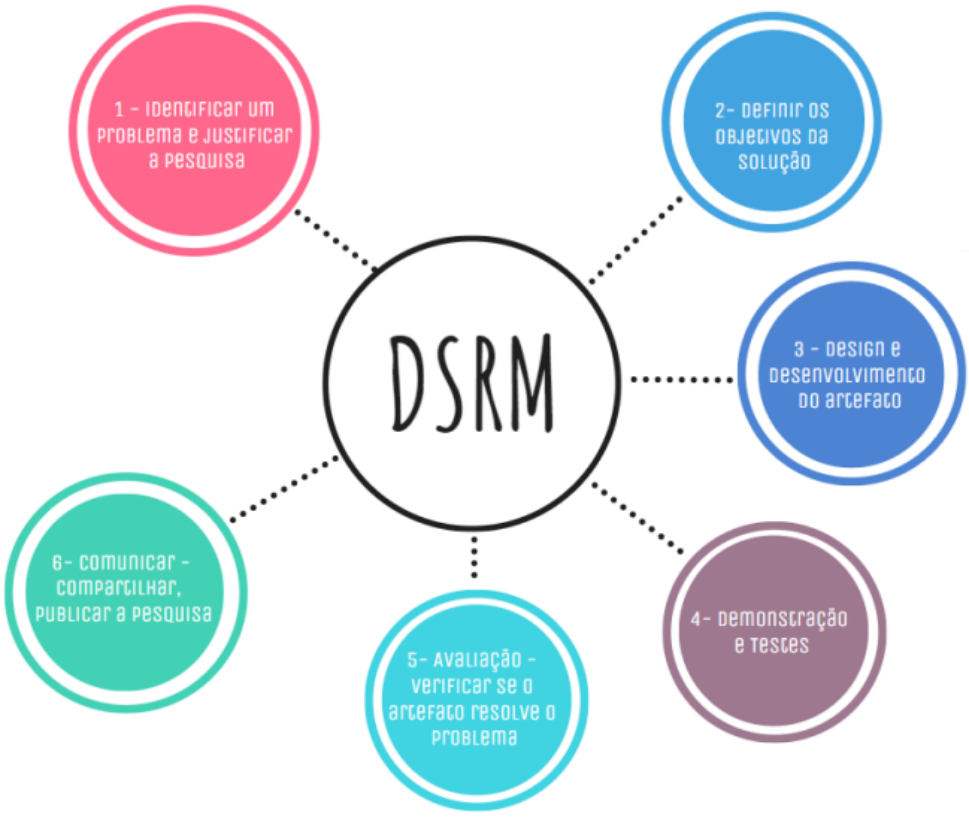

Figura 17 – Metodologia DSRM. Fonte: (SILVA, 2021)

4.1 Métricas

Em modelos classificatórios, as principais métricas de avaliação são acurácia, precisão, F1 e revocação. Nas fórmulas 4.2 e 4.3, para uma intenção, *tp* são os casos em que

o texto foi classificado corretamente como essa intenção; *fp* são os casos em que o texto foi classificado incorretamente como essa intenção; e *fn* são os casos em que o texto foi classificado incorretamente como não sendo essa intenção (PAULA, 2021).

$$acuracia = \frac{Quantidade\ de\ respostas\ preditas\ corretamente}{Quantidade\ de\ respostas\ totais} \tag{4.1}$$

$$precisao = \frac{tp}{tp + fp} \tag{4.2}$$

$$revocacao = \frac{tp}{tp + fn} \tag{4.3}$$

$$F1 = \frac{2 \times precisao \times revocacao}{precisao + revocacao} \tag{4.4}$$

A parir da acurácia, temos quantas amostras foram preditas corretamente no total, enquanto que a precisão e a revocação nos permite analisar se o modelo tende a predizer corretamente mais uma classe que outra. Já o F1 é uma combinação de precisão e revocação. Assim, quanto maior o valor de precisão, revocação e F1, maior a garantia de que o modelo não está classificando corretamente apenas determinadas classes, mas que de fato está classificando corretamente para a maioria das classes presentes na base. (PAULA, 2021)

4.2 Dados

A base de dados desse trabalho foi obtida através de contato com a SECEX, e consiste nas perguntas recebidas pela CAJ e das respostas a essas perguntas escritas pelos servidores do órgão. Somente os atendimentos realizados até maio de 2022 estarão no conjunto de dados. Os dados foram colhidos através dos formulários de atendimento.

Os formulários de atendimento são respondidos pelos servidores envolvidos no chamado, e possuem a seguinte estrutura: nome do servidor, sistema ou funcionalidade de que trata o atendimento, e problema/resolução. A coluna Sistema por vezes trata de temas que vão além de nomes de sistemas, por isso nos próximos parágrafos, me referirei a ela com o nome de assuntos. O documento cedido pela CAJ possui um total de 3700 atendimentos, divididos em 82 assuntos. Os assuntos com maior quantidade de registros são referentes a nomes de sistemas ou suas funcionalidades, sendo SISPATRI, SIAI OBRAS e Gerenciamento de Usuários do Portal do Gestor, os assuntos com maior número de ocorrências: 601, 436 e 346 ocorrências, respectivamente.

Os assuntos com menor quantidade de ocorrências tratam de temas avulsos, que não estão relacionados a sistemas ou a assuntos que possam interessar aos jurisdicionados como

um todo, e muitas vezes tratam de processos relacionados aos usuários que estão entrando em contato com o TCE. Como são campos livres a serem preenchidos pelo atendentes, diversos assuntos possuem apenas um registro, como exemplo: recibo entrega, contato da DDP, e Ligação ao usuário. Na Tabela 4 são apresentados alguns registros contidos nesse documento. Os nomes próprios presentes na amostra foram trocados por nomes fictícios.

Data/Hora	Atendente	Sistema	Problema/Resolução	Setor
5/19/2021 14:51:03	LUIS	[PdG] Novo SIAIDP	Cm montanhas - Monica - Siai dp – usuária gostaria de saber como acessar as tabelas auxiliares ..... foi informado a usuária que ela deveria acessar o novo siai dp e fazer logoff pra ter aceso ao link.	COEX/CAJ
5/19/2021 15:05:26	LUIS	[PdG] [Site] E-CONSULTA	Carlos Alberto -Consultar processo – 002067/2018 . problemas para visualizar processo. Foi informado ao usuário que é possível realizar e visualizar todo o processo via portal do gestor através do link consulta processos.	COEX/CAJ
5/19/2021 15:12:15	LUIS	contato	Maria - usuária gostaria de falar com o gabinete de Sr Thompson. foi solicitado a mesma pra entrar em contato com o telefone 3642-7262	COEX/CAJ
5/31/2021 16:45:08	LUIS	[PdG] LEGIS	Legis – Angélica – edição de norma como fazer? Usuária informa que a norma foi revogada e que ela não consegue editar ..... Foi informado a usuária que a mesma pode solicitar a exclusão	COEX/CAJ
6/30/2021 11:42:02	LUIS	[PdG] Anexo 13	Luciana – anexo 13 - Bom tarde,Considerando que o anexo SIAI 13, trata do Contrato e dos Aditivos, e outras alterações, solicitamos por orientação do controle interno desta Autarquia a inclusão da base legal no respectivo anexo, conforme Parágrafo 8º do Artigo 65 da Lei nº 8.666 de 21 de Junho de 1993, que trata de apostilamento. Luciana Eustáquio	COEX/CAJ

Tabela 4 – Exemplo de Estrutura dos Dados.

Como já mencionado, a primeira dificuldade enfrentada é a grande quantidade de temas que podem ser abordados pelo jurisdicionado. Eles podem solicitar informações

simples como números de telefone, falar sobre erros ou tirar dúvidas sobre funcionalidades do sistema, além de tratar de trâmites do meio jurídico.

A segunda dificuldade é que não é seguida uma estrutura rígida nos textos. Todos os exemplos apresentados na Tabela 4 pertencem ao mesmo servidor, e apesar disso, percebe-se que de um texto para o outro a estrutura muda. Alguns itens são totalmente descritivos, outros apresentam pergunta e resposta, e alguns textos falham em descrever o passo-a-passo como o item 4. Apesar disso, na tarefa de QA, é uma característica bastante comum que dados reais não sigam uma estrutura fixa.

Na Tabela 5 tem-se uma base de dados criada de forma automatizada a partir dos textos da Tabela 4. As duas colunas Pergunta e Resposta são preenchidas a partir da separação dos textos, e a esse par [Pergunta, Resposta] é formada uma intenção, e a ela deve ser dado um nome, que está representado na última coluna da tabela. A definição da quantidade de exemplos de perguntas e respostas de uma mesma intenção ficam a cargo do desenvolvedor, e essas quantidades são independentes, podendo haver N perguntas e M respostas para uma mesma intenção. Destaco o último item, pois nele não foi possível separar os textos de pergunta e resposta, e com isso ele é descartado da base de dados final, que ao fim dessa etapa possui um total de 1.727 registros. Não é possível declarar uma intenção sem pelo menos uma pergunta e uma resposta. O algoritmo usado na automatização será explicado no Capítulo 5.

O TCE deverá fazer uma análise qualitativa dos dados gerados e ponderar se eles devem ser usados no treinamento do *chatbot*, ou se a geração manual de dados é o mais adequado ao resultado que se deseja obter com a ferramenta, embora seja mais demorado. Neste trabalho foi escolhida a abordagem automatizada, pois o objetivo do trabalho está em comparar os modelos *transformers* à disposição, não sendo determinante cobrir toda a base de dados para realizar esse estudo. Entretanto, recomenda-se que os dados sejam analisados em sua totalidade numa etapa pós-implantação. A limpeza e a geração de novos dados está detalhada no Capítulo 5.

4.3 *Pipeline*

O *pipeline* pode ser descrito como a sequência de passos necessários para treinar um modelo. Para cada um deles existe um leque de componentes que podem ser empregados.

4.3.1 *Tokenizers*

Os *tokenizers* quebram a entrada de texto em pedaços chamados *tokens*. É a primeira etapa que deve ser empregada em qualquer *pipeline*, e o idioma do texto é um fator que limita a quantidade de componentes a disposição do desenvolvedor. Alguns componentes:

Pergunta	Resposta	Intenção
Cm montanhas - Monica - Siai dp – usuária gostaria de saber como acessar as tabelas auxiliares	foi informado a usuária que ela deveria acessar o novo siai dp e fazer *logoff* pra ter aceso ao *link*	siaidp_tabela_auxiliar
Carlos Alberto -Consultar processo – 002067/2018 . problemas para visualizar processo	Foi informado ao usuário que é possível realizar e visualizar todo o processo via portal do gestor através do link consulta processos	portalgestor_processo_acesso
Maria - usuária gostaria de falar com o gabinete de Sr Thompson	foi solicitado a mesma pra entrar em contato com o telefone 3642-7262	contato_thompson
Legis – Angélica – edição de norma como fazer	Foi informado a usuária que a mesma pode solicitar a exclusão	legis_norma_editar
Luciana – anexo 13 - Bom tarde,Considerando que o anexo SIAI 13, trata do Contrato e dos Aditivos, e outras alterações, soli- citamos por orientação do controle interno desta Autarquia a inclusão da base legal no respectivo anexo, conforme Parágrafo 8º do Artigo 65 da Lei nº 8.666 de 21 de Junho de 1993, que trata de apostilamento. Luciana Eustáquio	-	-

Tabela 5 – Intenções correspondentes aos dados da Tabela 4

- *WhitespaceTokenizer*: *Tokeniza* a entrada de texto bruto usando espaços em branco como separador (RASA, 2022).
- *SpacyTokenizer*: Oferece suporte a mais de 60 idiomas, incluindo o português (SILVA, 2021). Usando esse componente, a entrada será quebrada em *tokens* correspondentes ao seu vocabulário (RASA, 2022).

4.3.2 *Featurizers*

Os *featurizers* transformam a entrada textual em um vetor de atributos, representação legível por uma máquina, que é usado como entrada do modelo de aprendizado. Podem ser divididos em dois tipos:

- *Sparse Featurizer*: vetor de atributos que pode conter grandes quantidades de zeros.

Uma alternativa para economizar memória é armazená-lo como atributos esparsos. Nesse caso, são armazenados apenas os valores diferentes de zero e a posição desses valores no vetor. Essa é a forma usada pelo Rasa (RASA, 2022).

- *Dense Featurizer*: vetor de atributos que contém *embedding* pré-treinado. Eles funcionam melhor em qualquer problema NLP, comparado aos vetores esparsos. Nele, palavras similares têm representações similares.

Por exemplo, "casa" e "lar" significam coisas diferentes em representações vetoriais esparsas; já a representação densa captura a semelhança entre essas palavras. É possível usar mais de um *featurizer*, inclusive de tipos diferentes, e concatená-los de modo que a entrada do *featurizer* seguinte depende da saída do anterior. Segue alguns componentes que executam essa etapa:

- *RegexFeaturizer*: Cria um vetor esparso usando expressões regulares (RASA, 2022).
- *LexicalSyntacticFeaturizer*: cria atributos léxicos e sintáticos para dar suporte à etapa de extração de entidade (RASA, 2022).
- *CountVectorsFeaturizer*: cria uma representação *bag-of-words* de textos, intenções e respostas (RASA, 2022). O *bag-of-words* é uma lista que contém todas as palavras que estão nos textos de maneira não repetida. Essa lista é usada no cálculo da frequência de cada palavra.

4.3.3 Classificadores de Intenção (*Intent Classifiers*)

Após gerar atributos para todos os *tokens* e para a sentença completa, essas informações são passadas para um modelo de classificação de intenções. Os classificadores de intenção atribuem a entrada do usuário a uma das intenções disponíveis no domínio. Alguns componentes dessa categoria são:

- *DIETClassifier*: lida tanto com classificação de intenções, quanto com extração de entidades.
- *FallbackClassifier*: quando o classificador de intenções não consegue classificar uma intenção com uma confiança maior ou igual ao valor limite, o *FallbackClassifier* classifica o texto de entrada com a intenção denominada *nlu_fallback* (RASA, 2022).

4.3.4 Extratores de Entidade (*Entity Extractors*)

Os extratores de entidade extraem entidades do texto, como nomes de pessoas ou locais. Além do modelo de ML DIET, outros componentes podem ser usados quando se

pensa em propósitos mais específicos. Segue algumas opções de componentes para essa etapa:

- *EntitySynonymMapper*: mapeia as entidades que forem consideradas sinônimos para o mesmo valor. Por exemplo, caso "Reino Unido" e "UK" sejam considerados sinônimos, o componente poderá mapear ambas as palavras quando apareceram para "UK".
- *DucklingEntityExtractor*: modelo pré-construído usado na extração de números, datas, urls, endereços de e-mail. Modelos pré-construídos não precisam de dados de treinamento.
- *SpacyEntityExtractor*: modelo pré-construído usado na extração de nomes, locais, nomes de produtos.
- *Regex*: para detecção de padrões específicos, como números de telefone ou CEP.
- *DIETClassifier*: modelo de ML. Necessário se preocupar em gerar dados para a etapa de treinamento.

4.3.5 Seletores (*Selectors*)

Para o caso de intenções que tenham mais de um exemplo de resposta, após o modelo definir a intenção com maior índice de confiança, fica a cargo dos seletores escolherem uma resposta dentre esse conjunto de respostas predefinidas. Um componente para esse passo é o *ResponseSelector*.

4.4 Detalhamento do *Design*

A solução foi desenvolvida usando NodeJS. Essa tecnologia permite a criação de aplicações *standalone* escritas em Javascript. Utilizou-se a biblioteca Venom Bot pra a integração da aplicação com o Whatsapp. A partir do momento que já era possível a troca de mensagens com o WhatsApp, o passo seguinte foi a integração da aplicação com a API do Rasa. Através de rotas específicas, foi possível enviar e receber mensagens do modelo treinado. O fluxo de mensagens está representado na Figura 18.

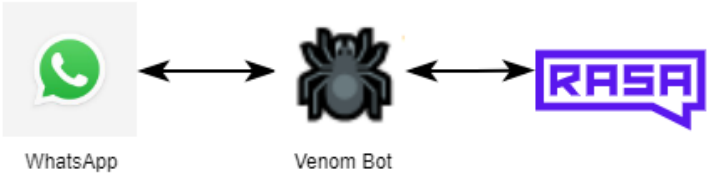

Figura 18 – Fluxo de mensagens da aplicação

No Rasa, foram criadas oito configurações de *pipeline*. Todas usam o classificador DIET, e com exceção do módulo *Pretrained Embedding*, todos os outros componentes são similares, e já vieram pré-configurados pelo Rasa. No Apêndice B são mostradas duas configurações: a primeira representa apenas a atuação do modelo DIET, por isso seu módulo *Pretrained Embedding* está desligado; a segunda representa o acoplamento de um modelo *transformer*, especificamente nesse exemplo o modelo BERTimbau, uma versão do modelo BERT treinado em português. De forma similar, pode-se acoplar diferentes modelos alterando-se apenas os parâmetros *model_name* e *model_weights* do componente *LanguageModelFeaturizer*.

O componente *LanguageModelFeaturizer* usa um dos modelos de linguagem *transformer* pré-treinados descritos no Capítulo 2 para calcular representações vetoriais do texto de entrada. Também cria recursos para extração de entidade, classificação de intenção e seleção de resposta. Os pesos são inicializados a partir do modelo pré-treinado escolhido, no nosso exemplo BERTimbau (neuralmind/bert-base-portuguese-cased). WARMERDAM (2022) enfatiza que, dessa forma, os parâmetros do modelo não são reajustados durante o treinamento usando os dados disponibilizados no Rasa, ou seja, não houve ajuste-fino. Isso economiza tempo de processamento, e os modelos de aprendizado de máquina no *pipeline* geralmente podem compensar a falta dessa etapa.

No *pipeline*, cada componente é declarado na sequência real em que vão ser utilizados. No caso do *LanguageModelFeaturizer*, a *tokenização* é originada da base brWAC, entretanto novos *tokens* podem ser adicionados ao modelo em decorrência da base de dados adicionada ao Rasa. Para esses dados, a *tokenização* será feita pelo componente *WhitespaceTokenizer*, adicionado um passo antes da declaração do *LanguageModelFeaturizer* no *pipeline*.

Sobre os componentes *RegexFeaturizer* e *CountVectorsFeaturizer* que aparecem nas declarações seguintes, apenas os dados de treinamento inseridos no Rasa passaram por esses componentes. Revisitando a arquitetura DIET na Figura 6, esses componentes geram os chamados atributos esparsos (*sparse features*), e eles estão em um caminho paralelo ao dos atributos densos (*dense features*), gerados pelo módulo de *embeddings* pré-treinados (*pretrained embedding*). O *CountVectorsFeaturizer* é usado duas vezes: na primeira, um modelo *bag-of-words* a nível de palavra, e no segundo usando *bag-of-words n-grams*, onde n é quantidade de letras em cada conjunto de cada palavra.

Nos experimentos deste trabalho, o modelo DIET refere-se ao classificador DIET com o módulo de *embedding* pré-treinado (*Pretrained embedding*) desligado, ou seja, estão sendo usadas apenas os atributos esparsos dos dados de treinamento; nos outros modelos têm-se o classificador DIET com o módulo de *embedding* pré-treinado conectado ao modelo referenciado, ou seja, os atributos densos do modelo pré-treinado se somarão aos atributos esparsos dos dados de treinamento, e elas serão as entradas no módulo *transformer*, como visto na Figura 6. O *hardware* usado nos testes é um desktop com processador Intel Core

I5 3330 com frequência de 3Ghz, 8gb de ram, e placa de vídeo integrada HD Intel 2500.

4.5 Considerações Finais

Nesse capítulo foi apresentada a metodologia que será seguida, um pouco da estrutura dos dados e as métricas que serão aplicadas na análise dos resultados. Também foi feita uma breve descrição dos componentes da plataforma Rasa, sua integração e como elas poderão auxiliar no desenvolvimento do *chatbot*.

5 EXPERIMENTAÇÃO E ANÁLISE DE RESULTADOS

Neste capítulo será detalhada a experimentação, e a fase de análise, em que serão comparados os resultados obtidos por meio desses testes.

5.1 Experimentação

Na fase de experimentação, todos os modelos serão submetidos aos testes de Validação Cruzada (CV - *Cross Validation*), que será explicado em detalhes mais a diante. Através desses testes serão calculadas as métricas definidas na Seção 4.1 e os resultados serão comparados.

A base de dados disponibilizada possui um total de 3.709 registros. A primeira dificuldade para utilizá-la foi separar o texto da coluna PROBLEMA/RESOLUÇÃO, que contém a descrição do atendimento, em duas colunas: pergunta e resposta. É através do treinamento, usando as perguntas e respostas inseridas na sua base de dados, que o *chatbot* consegue identificar a intenção de uma pergunta do usuário, e responder da forma esperada.

Para fazer essa separação de forma automatizada, os dados foram analisados com o objetivo de identificar padrões nesses textos que indicassem quando essa separação poderia acontecer. Nessa análise, ficou claro que não havia uma regra estipulada dentro da CAJ de como os textos deveriam estar estruturados. Os padrões mudam de acordo com o atendente, e por vezes o mesmo atendente altera a forma de escrita. Alguns textos foram descartados porque continham apenas pergunta ou resposta, ou porque não resolviam o atendimento, solicitando mais dados a quem requisitava.

Foram identificados delimitadores para a separação desses textos, como: quebras de linhas, espaçamentos, pontos de interrogação, entre outros. Do montante inicial de 3.709 registros, essa etapa resultou em 1.727 registros. Os registros resultantes foram revisados, e quase metade foi descartado. Uma explicação para esse descarte tão alto é que muitos atendimentos constavam apenas da resposta ou da pergunta, enquanto outros solicitavam informações muito específicas ao atendente do setor, como solicitar o andamento de um processo de seu interesse. Apenas 903 registros possuíam perguntas e respostas de caráter genérico, que poderiam se aplicar a qualquer usuário dos serviços do TCE.

Outra preocupação eram os erros de português que constantemente estavam presentes nos textos. Apesar do componente *WhitespaceTokenizer*, presente em nosso *pipeline*,

gerar novos *tokens* para palavras que ainda não estavam presentes no vocabulário do modelo, palavras escritas incorretamente afetam negativamente no cálculo de similaridade entre sentenças e na aprendizagem da relação entre as palavras nas sentenças.

Para a etapa de correção ortográfica, cada palavra foi cruzada com os *corpus* brWAC (FILHO et al., 2018) e ITD (SOUSA; FABRO, 2019). A ITD é uma base de acórdãos decididos pelo STF entre 2010 e 2018. Essa base serviu para expandir o vocabulário, acrescentando palavras do meio jurídico. Assim, caso a palavra não estivesse presente em nenhuma das duas bases, provavelmente ela estava mal-escrita. Ainda assim, ela também era cruzada com a base Mac-Morpho (ALUíSIO et al., 2003), para verificar se ela poderia ser um nome próprio, como também se ela pertencia a um conjunto de siglas dos setores do TCE. Se ainda assim ela não estivesse presente em nenhum dos conjuntos descritos, ela finalmente passa pela etapa de correção gramatical, a cargo da biblioteca *python pyspellchecker* (*Pure Python Spell Checking*). Ela usa o algoritmo da Distância de Levenshtein para encontrar permutações dentro de uma distância de edição de dois da palavra original. Em seguida, ele compara todas as permutações (inserções, exclusões, substituições e transposições) com palavras conhecidas em uma lista de frequência de palavras. As palavras que são encontradas com mais frequência na lista de frequência são as mais prováveis de serem as corretas (BARRUS, 2023).

Uma das formas de avaliar o *chatbot* está na criação de diálogos de teste. Cada diálogo de teste contém as entradas do usuário e as intenções que estão associadas a essas entradas. Esses diálogos contarão uma estória, desde a saudação até o desfecho da conversa. Esses testes têm importância quando representam situações reais, e é importante que o sistema já esteja em uso, pra que sejam armazenados os diálogos com usuários reais, e a partir deles seja feita a confecção dos diálogos de teste. Por isso, esse tipo de teste não será implementado nesse trabalho, mas poderá ser posto em prática posteriormente, durante testes com usuários reais, ou após a ferramenta entrar em produção.

A forma de avaliação que será usada é o da Validação Cruzada (CV - *Cross Validation*) (LACHENBRUCH; MICKEY, 1968; LUNTZ; BRAILOVSKY, 1969). Na CV *k-fold*, o conjunto de dados é dividido em k partes de mesmo tamanho, chamados de *folds*. O valor de k também define quantas iterações serão feitas na execução. Na CV, $k - 1$ conjuntos serão utilizados como dados de treinamento e o conjunto que sobrou será o conjunto de testes, demonstrado na Figura 19.

Os dados são divididos uma única vez. Contudo, a cada iteração, o conjunto de treinamento e o de teste mudam. Ao final das iterações, todos os conjuntos de treinamento, em algum momento, foram conjuntos de teste. A cada iteração, um novo modelo de treinamento é gerado, de acordo com o conjunto de dados de treinamento. O desempenho do modelo é avaliado a partir da comparação entre o conjunto de dados de treinamento e o conjunto de testes. Após a comparação, o modelo de treinamento é descartado. Os

valores de Acurácia, F1 e Precisão são somados a cada iteração e o resultado final é um valor médio de cada métrica (CAMPOS, 2022).

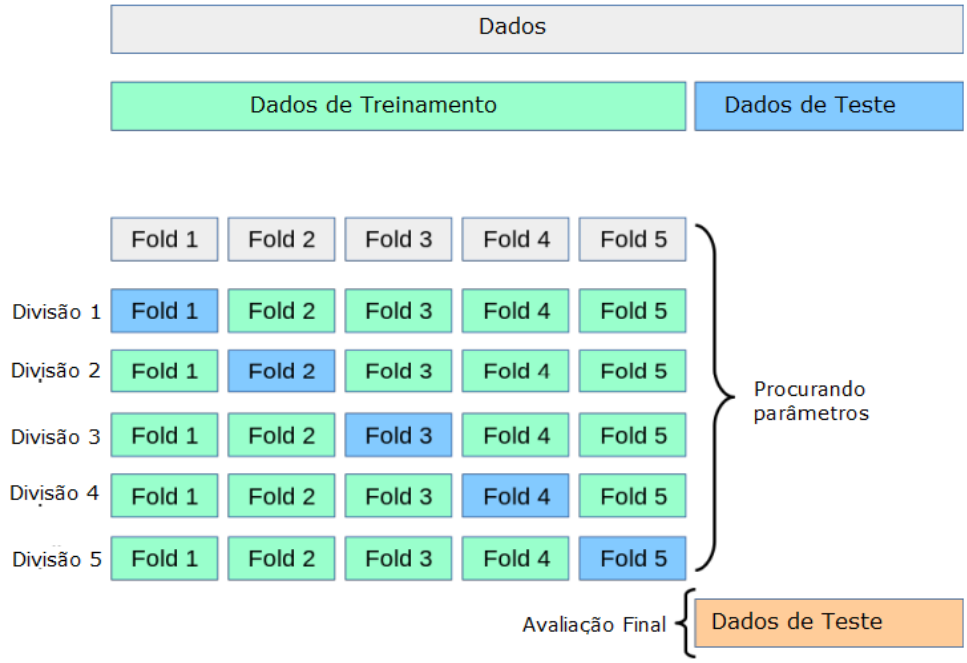

Figura 19 – Funcionamento da validação cruzada para 5-*folds*. (PEDREGOSA et al., 2011)

O teste de CV avalia o desempenho do NLU em garantir que as entradas do usuário sejam reconhecidas. Os testes se limitarão a valores de k = {2,3}, e serão feitos de forma que cada intenção em um subconjunto de dados possua pelo menos dois exemplos. Esse é um requisito do próprio Rasa, que descarta do teste intenções que possuem apenas um exemplo. Para seguirmos essa metodologia, cada intenção deve ter pelo menos seis exemplos.

Os exemplos que fazem parte de uma mesma intenção não podem se repetir, assim como o nome de cada intenção deve ser único. A etapa de limpeza resultou numa base de dados de 903 intenções, cada uma com apenas uma pergunta e uma resposta. Para executarmos a CV, será preciso gerar mais cinco perguntas para cada intenção, totalizando 5.418 perguntas e 903 respostas.

Como solução, foi desenvolvido um algoritmo que percorre o código *html* do *site* https://sinonimos.com.br/ com o objetivo de capturar a lista de sinônimos de uma palavra de interesse. O algoritmo desenvolvido percorre toda a lista de intenções, e em cada uma analisa a pergunta que tem a disposição. Na pergunta, seleciona aleatoriamente duas ou mais palavras que deverão ser substituídas por seus sinônimos. Através da lista de sinônimos obtida no *site*, são gerados os próximos cinco exemplos que vão compor as perguntas da intenção. Caso essa lista de sinônimos tenha menos que cinco itens, outra palavra da sentença será sorteada, e o processo se repete, até que cada intenção da base

tenha seis exemplos não repetidos de perguntas.

No Rasa, os exemplos de cada intenção são escritos no arquivo *nlu.yml* (Figura 21), enquanto as respostas de cada uma delas são escritos no arquivo *domain.yml* (Figura 22), e no arquivo *rules.yml* (Figura 22) é onde se declara o relacionamento entre a resposta e a intenção correspondente. Na Figura 20 está a sentença original que consta na base do TCE. Após passar pela etapa de correção gramatical, ela foi alocada como o primeiro exemplo de pergunta de determinada intenção, como visto na Figura 21, enquanto as outras perguntas são geradas a partir dos sinônimos das palavras "solicitação" e "informação".

```
qual a ferramenta para solicitação infomração? E-sic
```

Figura 20 – Frase original da intenção int6.

```
- intent: int6
  examples: |
    - qual a ferramenta para solicitação informação
    - qual a ferramenta para pedido conhecimento
    - qual a ferramenta para súplica referência
    - qual a ferramenta para rogo noção
    - qual a ferramenta para imploração ideia
    - qual a ferramenta para insistência base
```

Figura 21 – Exemplos de frases associadas a intenção int6.

```
utter_int6:
- text: "E-sic"
```

Figura 22 – Exemplos de respostas associados a intenção int6.

```
- rule: int6
  steps:
  - intent: int6
  - action: utter_int6
```

Figura 23 – Regra que associa intenções a respostas.

5.2 Análise de Resultados

Nesta seção será feito um estudo comparativo dos resultados obtidos durante os testes de CV. Além das pontuações de qualidade, foram colhidos também tempos de treinamento e inferência de cada modelo. Cada tabela estará ordenada do melhor ao pior resultado, com base nas métricas apresentadas em cada uma. A análise dos resultados levará em consideração os melhores modelos em cada estágio de teste, e apesar de não existir um modelo que será o melhor em todos os quesitos, essa análise tentará extrair o modelo que represente o melhor custo-benefício a instituição.

Nas Tabelas 6 e 7 são apresentados os resultados obtidos, e nela as execuções estão divididas em número de *folds*, modelo, métrica e pontuação. A pontuação consiste

da média das execuções para cada *fold*, acrescido do desvio padrão. Nos testes, cada exemplo é classificado com base numa pontuação de confiança denominada similaridade. A similaridade é calculada a partir do produto escalar entre a sentença de teste e as sentenças de treinamento. Quanto mais similares as sentenças, maior a pontuação, de tal modo que a intenção inferida pelo modelo será a da sentença de treinamento com maior pontuação de similaridade (RASA, 2023).

A sentença predita então será contabilizada como Verdadeiro Positivo (TP - *True Positive*), Falso Positivo (FP - *False Positive*) ou Falso Negativo (FN - *False Negative*), e impactará no cálculo das métricas apresentadas na Seção 4.1. A matriz de confusão na Figura 24 exemplifica esses intervalos de dados na predição da classe *siaidp_tabela_auxiliar* encontrada na Tabela 5 e permite compreender melhor essa divisão.

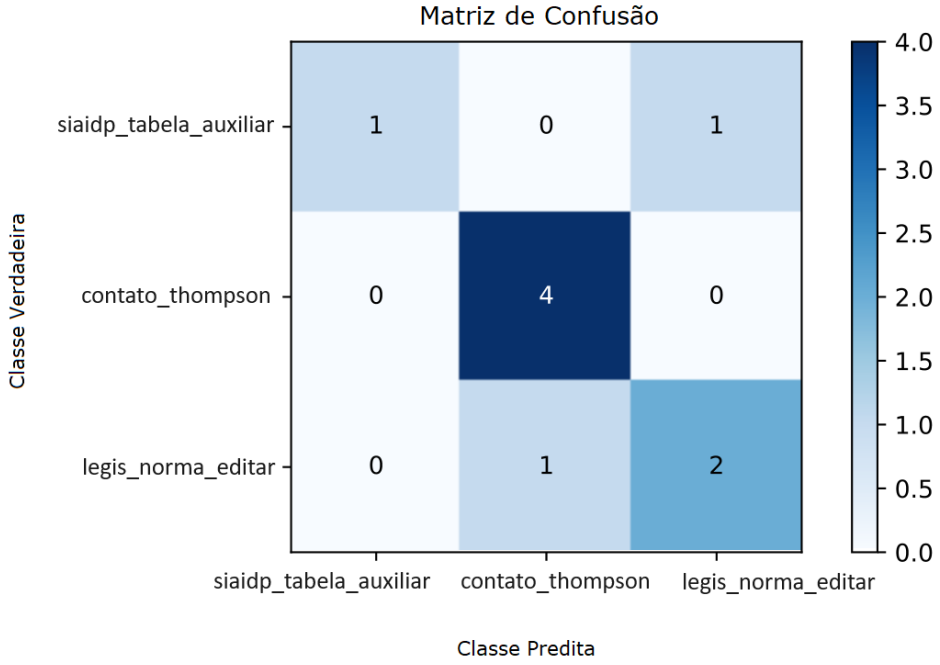

Figura 24 – Predição de uma sentença da classe *siaidp_tabela_auxiliar*

Todas as classes possuem 6 exemplos, ou seja, a base de dados está balanceada. Isso é importante no cálculo da acurácia, de forma que todas as classes possuem igual influência sobre o valor calculado. O TP contabiliza as classes classificadas corretamente, o FP são as classificadas incorretamente, e FN são as classificações que resultam em *fallback*. O cálculo da precisão leva em consideração o FP, enquanto a revocação usa o FN, e a métrica F1 considera as duas medidas.

Em ambas as Tabelas 6 e 7, os modelos estão ordenados do melhor ao pior resultado. Comparando-as, é possível observar que houve um melhor resultado em todos os modelos no teste com 3-*folds*. A explicação para isso é de que no teste 2-*folds*, o conjunto de dados será separado em duas partes iguais, uma para treinamento e outra para validação; já

com 3-*folds*, a quebra é desigual, ficando 66,7% dos dados como conjunto de treinamento e 33,3% para validação. Com a rede treinada com mais dados, é plausível concluir que ela obtenha melhores resultados, o que de fato se concretizou.

Comparativo 2-*folds*		
Modelo	Métrica	Pontuação
LaBSE	Acurácia	0,985 $\pm$ 0,001
	Precisão	0,987 $\pm$ 0,000
DistilBERT	Acurácia	0,981 $\pm$ 0,000
	Precisão	0,981 $\pm$ 0,001
BERTimbau	Acurácia	0,977 $\pm$ 0,003
	Precisão	0,980 $\pm$ 0,003
DIET	Acurácia	0,974 $\pm$ 0,000
	Precisão	0,976 $\pm$ 0,001
RoBERTa	Acurácia	0,974 $\pm$ 0,001
	Precisão	0,975 $\pm$ 0,002
GPT	Acurácia	0,973 $\pm$ 0,001
	Precisão	0,973 $\pm$ 0,001
XLNET	Acurácia	0,915 $\pm$ 0,001
	Precisão	0,920 $\pm$ 0,001
GPT-2	Acurácia	0,782 $\pm$ 0,050
	Precisão	0,767 $\pm$ 0,058

Tabela 6 – Comparativo entre modelos em um teste CV com 2-*folds*.

Ao compararmos as tabelas observamos que a ordem se manteve, exceto por um par, os modelos DIET e RoBERTa inverteram sua ordem de um teste para o outro. Enquanto no teste de 2-*folds* esses modelos apresentam resultados praticamente similares, com 3-*folds* observa-se uma ligeira vantagem do RoBERTa. Somente com esse teste entretanto, não é possível concluir se isso é uma tendência, nem se o modelo RoBERTa continuaria abrindo vantagens maiores em testes com mais *folds*. Ainda assim, são resultados bastante próximos, e como o teste só foi executado uma vez com cada modelo, é possível que em outras execuções, devido a proximidade de resultados, essa ordem possa se inverter novamente.

O indicado aqui seria, para cada validação cruzada de *n-folds*, repetir o mesmo teste determinado número de vezes e calcular a média dos resultados. Isso evitaria possíveis erros que determinada execução do teste possa trazer devido a fatores externos a ele, como o sistema operacional processando uma tarefa em segundo plano ou fazendo o *download* de uma atualização paralelo ao teste. Entretanto, os testes aqui são bastante custosos, o que pode ser notado na Tabela 9, com testes que ultrapassaram as 24 horas, tornando essa proposta inviável.

Essa constatação de que modelos com pontuação muito próxima poderiam se inverter na ordenação de qualidade, caso os testes fossem executados mais de uma vez,

Comparativo 3-*folds*		
Modelo	Métrica	Pontuação
LaBSE	Acurácia	0,992 $\pm$ 0,002
	Precisão	0,992 $\pm$ 0,002
DistilBERT	Acurácia	0,982 $\pm$ 0,001
	Precisão	0,981 $\pm$ 0,001
BERTimbau	Acurácia	0,982 $\pm$ 0,001
	Precisão	0,980 $\pm$ 0,001
RoBERTa	Acurácia	0,982 $\pm$ 0,001
	Precisão	0,980 $\pm$ 0,002
DIET	Acurácia	0,981 $\pm$ 0,001
	Precisão	0,979 $\pm$ 0,001
GPT	Acurácia	0,978 $\pm$ 0,003
	Precisão	0,976 $\pm$ 0,004
XLNET	Acurácia	0,936 $\pm$ 0,008
	Precisão	0,939 $\pm$ 0,010
GPT-2	Acurácia	0,832 $\pm$ 0,048
	Precisão	0,806 $\pm$ 0,057

Tabela 7 – Comparativo entre modelos em um teste CV com 3-*folds*.

pode ser levada em consideração para os outros modelos além do DIET e do RoBERTa. No teste de 2-*folds* conseguiu-se obter resultados mais espaçados entre os modelos, enquanto com 3-*folds*, do DistilBERT ao GPT, a diferença de pontuação é mínima. Apesar disso, o LaBSE obteve o melhor resultado em ambos testes, com um resultado mais espaçado a frente do segundo colocado, enquanto os modelos XLNET e GPT-2 tiveram os piores resultados, também descolados dos modelos a frente deles.

Uma indagação que pode ser feita é de como o GPT-2 em ambos os testes obteve resultados piores que o GPT. No artigo de RADFORD et al. (2019) são apresentados quatro modelos GPT-2 de diferentes tamanhos, sendo o modelo estado da arte o de maior tamanho, possuindo 1,5 bilhões de parâmetros. O Rasa, por padrão, usa o GPT-2 de menor tamanho, que possui 117 milhões de parâmetros. A título de comparação, o GPT possui 120 milhões de parâmetros. O GPT-2 de menor tamanho e o GPT tem tamanhos bem similares e mesma arquitetura, e mesmo assim houve uma diferença expressiva nos resultados. A literatura não cobre muito bem o comparativo entre eles, e normalmente usa o modelo de maior tamanho como sinônimo de GPT-2, de forma que o resultado obtido nesse comparativo fica carente de uma melhor explicação.

Foi aplicado aos dados o teste de Friedmann com o objetivo de verificar se existem diferenças estatísticas nas medições. Esse é um teste estatístico não-paramétrico, ideal para quando existem poucas amostras no conjunto de dados. Como pode ser constatado na Tabela 8, para todas as métricas, o p-valor obtido foi maior que 0,05 e portanto não

existe diferença estatística entre as medições.

Medida	p-valor
Acurácia	0,057
Precisão	0,060

Tabela 8 – Teste de Friedman.

Levando em consideração o fator tempo, a Tabela 9 possui o tempo gasto na fase de treinamento e no teste completo da validação cruzada para 2 e 3 *folds*. Já na Tabela 10 é calculado o tempo que o modelo leva para processar a resposta ao usuário. Foi testado um conjunto de 6 intenções. Para o teste de cada intenção, foi submetida ao modelo uma frase de entrada idêntica a frase que havia na sua base de dados, de forma que ele processaria que aquela frase pertencia a determinada intenção e responderia corretamente ao usuário. Em cada intenção, a entrada era submetida ao *chatbot* 5 vezes, e depois era calculada a sua média. Além dessas médias, a tabela também traz uma média geral, na última coluna.

Modelo	Treinamento	CV 2-*folds*	CV 3-*folds*
BERTimbau	04:29	13:46	24:27
DIET	09:29	10:20	18:46
DistilBERT	09:37	12:54	23:03
RoBERTa	09:44	14:24	26:31
LaBSE	10:32	13:52	25:23
GPT	10:52	13:36	26:09
GPT-2	11:15	15:07	26:37
XLNET	11:32	14:58	27:21

Tabela 9 – Tempos de execução em horas.

Modelo/Frase	int6	int22	int9	int13	int15	int25	Média
DIET	193	194	214	224	224	229	213
DistilBERT	358	376	387	415	429	392	392,8
GPT	566	521	483	644	632	564	568,3
LaBSE	558	573	524	602	611	573	573,5
BERTimbau	533	546	539	624	649	571	577
GPT-2	537	563	521	635	685	598	589,8
XLNET	569	608	543	690	707	608	620,8
RoBERTa	600	590	610	693	708	586	631,1

Tabela 10 – Tempos de resposta em mili-segundos.

Na fase de treinamento da Tabela 9, o destaque foi o BERTimbau, duas vezes mais rápido que os modelos DIET, DistilBERT e RoBERTa. Porém esse número não reflete uma vantagem nos tempos dos testes de validação cruzada, resultando aqui em números

piores que o DistilBERT. Isso pode ser explicado no comparativo de tempo de resposta da Tabela 10, com o DIET respondendo em média quase três vezes mais rápido que ele. Na análise dos autores desse trabalho, o tempo de resposta é uma medida mas crítica do que o tempo de treinamento. Enquanto é possível treinar uma rede apenas uma vez e tê-la funcionando por semanas até que mais dados sejam capturados, o tempo de resposta vai estar sempre sendo posto a prova na comunicação com o usuário final do sistema. Para esse usuário o treinamento da rede é transparente, porém o tempo de resposta é crucial para mantê-lo usando o *chatbot* ou não. Assim, como destaque no fator tempo, os modelos DIET e DistilBERT obtiveram os resultados mais animadores, e esse resultado pode ser melhor constatado graficamente na Figura 25.

Foi aplicado o teste de Friedmann aos tempos de resposta e obteve-se um p-valor menor que 0,001 indicando a existência de diferença estatística. Diante disso, foi executado o teste pareado *post-hoc* Durbin-Conover para comparar os modelos dois a dois, com o objetivo de mostrar os pontos em que existe essa diferença. Como pode ser visto na Tabela 11, houve diferença estatística significativa na ampla maioria das comparações.

A proposta do DistilBERT é a de ser um modelo mais leve e rápido que o BERT, entretanto, para chegar nessas características, o modelo performaria pior na métrica F1. Nesse trabalho, ele cumpre o esperado, estando entre os melhores tempos de resposta, e surpreende nas métricas de qualidade, apresentado o segundo melhor resultado dentre os modelos da família BERT. Já o DIET consegue entregar a melhor performance, e essa característica deve estar associado ao fato de que ele é o modelo que acopla os outros nesse comparativo. No Rasa, não é possível testar nenhum modelo *transformer* sem o DIET como intermediário, de forma que nesse comparativo, o DIET seria um modelo mais leve até que o DistilBERT, o que deve explicar pertencer a ele o melhor tempo de resposta.

No critério Precisão, nota-se que apesar do LaBSE ter se descolado em relação aos outros modelos, em comparação ao segundo colocado, a melhoria de pontuação é bastante pequena, como pode ser visto graficamente nas Figuras 26 e 27, com o DistilBERT sendo 1,2% pior que o LaBSE e 0,2% melhor que o DIET na CV de 3-*folds*. Trazendo para a análise o fator tempo de resposta, o DistilBERT teve tempos 54% piores em média que o DIET e 68,4% melhores que o LaBSE. A qualidade entre esses modelos é bastante semelhante, de forma que se um dos modelos responder errado ou não entender a entrada do usuário, é bastante provável que o outro apresente o mesmo resultado; porém com o DIET, a resposta ao usuário é 1,5 vezes mais rápido que com o DistilBERT.

Assim, levando em consideração os critérios de pontuação de qualidade, tempo de treinamento e resposta do modelo *transformer*, consideramos o DIET a melhor escolha, principalmente por ser a escolha mais equilibrada desse comparativo, com ênfase nos tempos de resposta, porém bastante competitivo nas métricas de qualidade.

Modelo A	Modelo B	p-valor
DIET	DistilBERT	0,094
DIET	GPT	<0,001
DIET	LaBSE	<0,001
DIET	BERTimbau	<0,001
DIET	GPT-2	<0,001
DIET	XLNET	<0,001
DIET	RoBERTa	<0,001
DistilBERT	GPT	<0,001
DistilBERT	LaBSE	<0,001
DistilBERT	BERTimbau	<0,001
DistilBERT	GPT-2	<0,001
DistilBERT	XLNET	<0,001
DistilBERT	RoBERTa	<0,001
GPT	LaBSE	0,569
GPT	BERTimbau	0,776
GPT	GPT-2	0,094
GPT	XLNET	<0,001
GPT	RoBERTa	<0,001
LaBSE	BERTimbau	0,776
LaBSE	GPT-2	0,258
LaBSE	XLNET	<0,001
LaBSE	RoBERTa	<0,001
BERTimbau	GPT-2	0,160
BERTimbau	XLNET	<0,001
BERTimbau	RoBERTa	<0,001
GPT-2	XLNET	<0,001
GPT-2	RoBERTa	<0,001
XLNET	RoBERTa	0,776

Tabela 11 – Teste *post-hoc* Durbin-Conover.

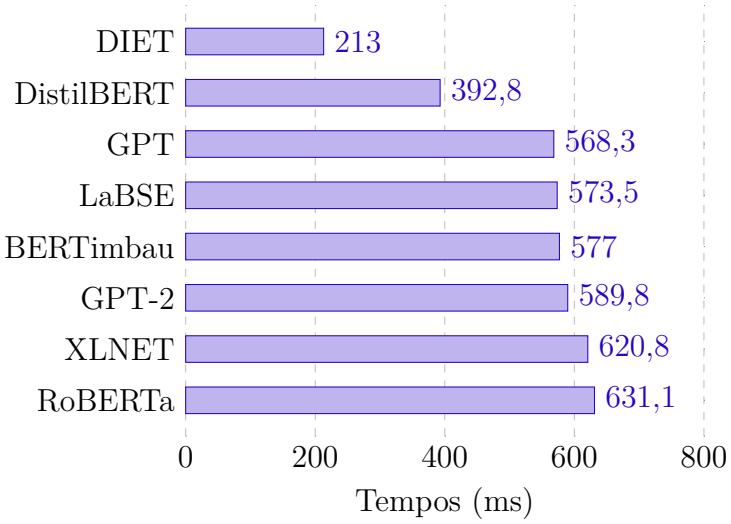

Figura 25 – Gráfico de tempos de resposta

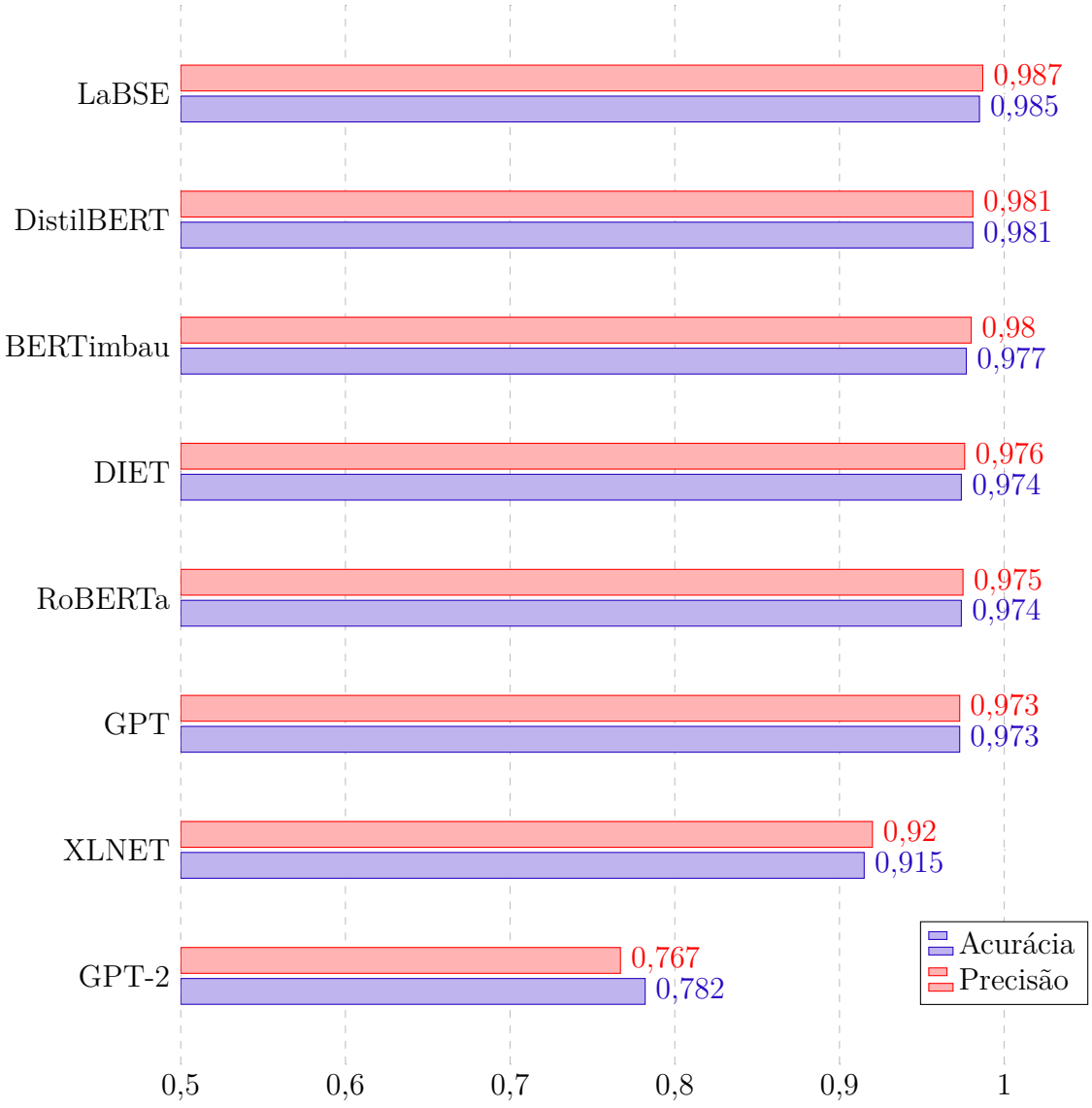

Figura 26 – Gráfico 2-folds

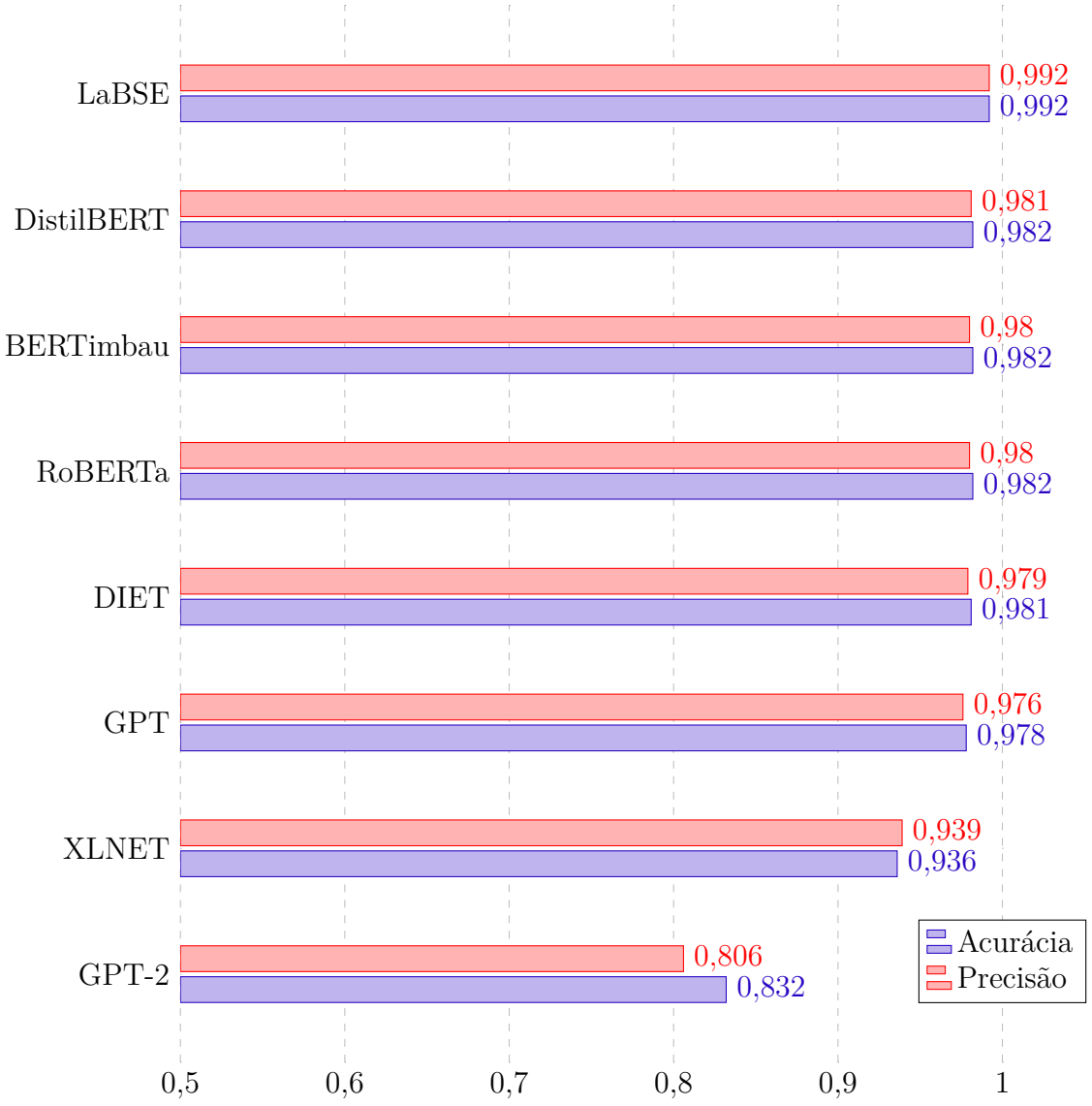

Figura 27 – Gráfico 3-folds

6 CONSIDERAÇÕES FINAIS

O *chatbot* pode ser uma excelente ferramenta para complementar as formas de atendimento ao cliente, e na SECEX espera-se que ele consiga melhorar a eficiência do setor. A classificação de intenções é uma das tarefas mais importantes nesse processo, e para ter sucesso nela precisamos de uma boa base de dados e um bom classificador.

No trabalho, pôde-se testar os modelos pré-treinados baseados em *transformers*, que incorporam as vantagens da auto-atenção e foram treinados em *corpus* gigantescos de texto; e o classificador DIET, que tem vantagens como arquitetura modular também baseada em *transformers*, classificação de intenções e entidades em um mesmo fluxo, e a possibilidade de usar diferentes modelos *transformers* pré-treinados para agregar na sua capacidade de classificação, também usando auto-atenção.

Toda a base do *chatbot* foi desenvolvida usando Rasa, incluindo a integração de modelos pré-treinados ao classificador DIET. Além disso, também foi criada a ferramenta que integra o Rasa ao WhatsApp, permitindo que a troca de mensagens entre usuários e *chatbot* aconteça. Para a fase de testes, foi feita uma limpeza da base de dados, incluindo uma fase automatizada e posteriormente uma revisão manual desses dados. Todos os modelos foram submetidos as mesmas configurações e condições similares de teste.

Apesar de testes iniciais apontarem indícios de que modelos *transformers* pré-treinados se tratavam da melhor opção, esses testes ficaram prejudicados devido a massa de dados limitada que foi utilizada neles. Nos novos testes, a base de dados aumentou de 19 para 903 intenções, e com isso, fatores como tempo de treinamento e resposta tiveram muito mais peso nessa última análise. O modelo DIET se mostrou a escolha mais adequada aos requisitos desse trabalho. Apesar de não apresentar o melhor resultado em todos os quesitos, ele foi o modelo mais equilibrado, e obteve resultados satisfatórios em qualidade e eficiência.

Com o crescente número de sistemas e o consequente aumento de atendimentos da CAJ, recomenda-se a SECEX que sejam estipuladas regras a serem seguidas nas transcrições dos atendimentos na base de formulários. Ao criar essas regras, todos os atendimentos seguirão a mesma estrutura de texto, permitindo que a DIN possa adaptar os algoritmos desse trabalho a essas regras, obtendo um resultado mais eficaz na filtragem de texto do que os obtidos nesse trabalho.

O *pipeline* desse trabalho foi pré-configurado pelo Rasa e usado sem nenhuma alteração em todos os comparativos. Como trabalhos futuros, seria interessante fazer o estudo do uso dos componentes vistos na Seção 4.3 e dos parâmetros usados na calibração

deles. Também com a ferramenta já implantada e funcionando, será possível armazenar os diálogos com usuários reais, para estudo dos assuntos mais recorrentes, dos que mais resultam em *fallback*, dos casos que fazem os usuários parar o uso da ferramenta no meio do diálogo, e outros pontos que poderão ser melhor definidos com o uso da ferramenta. O resultado disso pode ser um trabalho acadêmico focado em melhorias, e o seu produto poderia ser um painel de BI (*Business Inteligence*), que faria o monitoramento desses diálogos, separando-os em categorias de interesse. Esses dados trariam *insights* interessantes, quanto aos sistemas e módulos que geram mais dúvidas.

REFERÊNCIAS

ACADEMY, E. *Pesquisa Teórica vs. Pesquisa Empírica*. 2020. Disponível em: <https://www.enago.com.br/academy/pesquisa-teorica-vs-pesquisa-empirica/>.

AHMED, M.; KHAN, H. U.; MUNIR, E. U. Conversational ai: An explication of few-shot learning problem in transformers-based chatbot systems. *IEEE Transactions on Computational Social Systems*, p. 1–19, 2023.

ALUíSIO, S. M. et al. An account of the challenge of tagging a reference corpus for brazilian portuguese. In: _____. *International Workshop Computational Processing of the Portuguese Language - PROPOR 2003*. [S.l.]: Springer, 2003.

AU-YEUNG, J. et al. Ai chatbots not yet ready for clinical use. v. 5, p. 23, 04 2023.

BARRUS, T. pyspellchecker. In: . [s.n.], 2023. Disponível em: <https://pypi.org/project-/pyspellchecker/>.

BI, B. et al. PALM: pre-training an autoencoding&autoregressive language model for context-conditioned generation. *CoRR*, abs/2004.07159, 2020. Disponível em: <https://arxiv.org/abs/2004.07159>.

BOCKLISCH, T. et al. Rasa: Open source language understanding and dialogue management. 12 2017.

BOOK, D. L. *Redes Neurais Recorrentes*. 2022. Disponível em: <https://www-.deeplearningbook.com.br/redes-neurais-recorrentes/>.

BROWN, T. B. et al. *Language Models are Few-Shot Learners*. 2020.

BUNK, T. et al. DIET: lightweight language understanding for dialogue systems. *CoRR*, abs/2004.09936, 2020. Disponível em: <https://arxiv.org/abs/2004.09936>.

CAMPOS, J. M. R. Avaliação de desempenho e de satisfação do usuário do assistente virtual ifes.talk. 2022.

CONNEAU, A.; LAMPLE, G. Cross-lingual language model pretraining. In: WALLACH, H. et al. (Ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019. v. 32. Disponível em: <https://proceedings.neurips.cc/paper_files/paper/2019-/file/c04c19c2c2474dbf5f7ac4372c5b9af1-Paper.pdf>.

CORTES, E. G. Quando, onde, quem, o que ou por que? um modelo híbrido de classificação de perguntas para sistemas de question answering. 2019.

CURRY, C.; O'SHEA, J.; CROCKETT, K. The design, production and implementation of a storytelling chatbot - poster. In: . [S.l.: s.n.], 2012.

DAI, Z. et al. Transformer-xl: Attentive language models beyond a fixed-length context. *CoRR*, abs/1901.02860, 2019. Disponível em: <http://arxiv.org/abs/1901.02860>.

DEVLIN, J. et al. BERT: Pre-training of deep bidirectional transformers for language understanding. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, 2019. p. 4171–4186. Disponível em: <https://aclanthology.org/N19-1423>.

FENG, F. et al. Language-agnostic BERT sentence embedding. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland: Association for Computational Linguistics, 2022. p. 878–891. Disponível em: <https://aclanthology.org/2022.acl-long.62>.

FILHO, J. A. W. et al. The brWaC corpus: A new open resource for Brazilian Portuguese. In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Miyazaki, Japan: European Language Resources Association (ELRA), 2018. Disponível em: <https://aclanthology.org/L18-1686>.

GILSON, A. et al. *How Well Does ChatGPT Do When Taking the Medical Licensing Exams? The Implications of Large Language Models for Medical Education and Knowledge Assessment*. 2022.

GUO, M. et al. Effective parallel corpus mining using bilingual sentence embeddings. In: *Proceedings of the Third Conference on Machine Translation: Research Papers*. Brussels, Belgium: Association for Computational Linguistics, 2018. p. 165–176. Disponível em: <https://aclanthology.org/W18-6317>.

HINTON, G.; VINYALS, O.; DEAN, J. *Distilling the Knowledge in a Neural Network*. 2015.

JOHNSTON, J. H. *The allure of machinic life : cybernetics, artificial life, and the new AI*. [S.l.]: Massachusetts Institute of Technology, 2008.

JUNIOR, A. F. L. J. Buti: um companheiro virtual baseado em computação afetiva para auxiliar na manutenção da saúde cardiovascular. 2008.

KOCOŃ, J. et al. ChatGPT: Jack of all trades, master of none. *Information Fusion*, Elsevier BV, v. 99, p. 101861, nov 2023. Disponível em: <https://doi.org/10.1016%2Fj.inffus.2023.101861>.

LACHENBRUCH, P. A.; MICKEY, M. R. Estimation of error rates in discriminant analysis. *Technometrics*, 1968.

LEONHARDT, M. et al. Elektra: Um chatterbot para uso em ambiente educacional. *Revista Novas Tecnologias na Educação (RENOTE)*, v. 1, p. 1–11, 09 2003.

LIMA, A. *INTRODUÇÃO AOS TRANSFORMERS*. 2022. Disponível em: <https://acervolima.com/introducao-aos-transformers/>.

LIU, Y. et al. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692, 2019.

LUNTZ, A.; BRAILOVSKY, V. On estimation of characters obtained in statistical procedure of recognition (in russian). *Techicheskaya Kibernetica*, 1969.

MALHOTRA, K. *Language Agnostic Sentence Embeddings*. 2021. Disponível em: <https://medium.com/@kmkaran212/language-agnostic-sentence-embeddings-57445eba02a>.

MARIETTO, M. das G. B. et al. Artificial intelligence markup language: A brief tutorial. *CoRR*, abs/1307.3091, 2013. Disponível em: <http://arxiv.org/abs/1307.3091>.

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*. [S.l.]: AAAI Press, 1994. (AAAI'94), p. 16–21.

MERRITT, R. *What Is a Transformer Model?* 2022. Disponível em: <https://blogs.nvidia.com/blog/2022/03/25/what-is-a-transformer-model/>.

NEVES, A.; BARROS, F.; HODGES, C. Iaiml: A mechanism to treat intentionality in aiml chatterbots. In: . [S.l.: s.n.], 2006. p. 225–231.

NORTE, A. L. do Rio Grande do. *CONSTITUIÇÃO DO ESTADO DO RIO GRANDE DO NORTE*. 2019. Disponível em: <http://www.al.rn.leg.br/portal/_ups/legislacao-/constituicaoestadual.pdf>.

ORKESTRAL. *Venom Bot*. 2022. Disponível em: <https://github.com/orkestral/venom>.

PALASUNDRAM, K. et al. Seq2seq++: A multitasking-based seq2seq model to generate meaningful and relevant answers. *IEEE Access*, v. 9, p. 164949–164975, 2021.

PAULA, R. T. Geração de dados sintéticos para treinamento de chatbots baseados na carta de serviços do governo do ceará. 2021.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, p. 2825—-2830, 2011.

PEFFERS, K. et al. A design science research methodology for information systems research. *Journal of Management Information Systems*, Routledge, v. 24, n. 3, p. 45–77, 2007. Disponível em: </brokenurl# https://doi.org/10.2753/MIS0742-1222240302>.

Qiu, Q. et al. A question answering system based on mineral exploration ontology generation: A deep learning methodology. *Ore Geology Reviews*, v. 153, p. 105294, fev. 2023.

RADFORD, A. et al. Improving language understanding by generative pre-training. 2018.

RADFORD, A. et al. Language models are unsupervised multitask learners. In: . [S.l.: s.n.], 2019.

RASA. *Components*. 2022. Disponível em: <https://rasa.com/docs/rasa/components>.

RASA. *DIETClassifier*. 2023. Disponível em: <https://rasa.com/docs/rasa/components-/#dietclassifier>.

SAMEERA; JOHN, D. Survey on chatbot design techniques in speech conversation systems. *International Journal of Advanced Computer Science and Applications*, v. 6, 07 2015.

SANH, V. et al. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019. Disponível em: <http://arxiv.org/abs/1910.01108>.

SELLANES, D. *NLP Transformer DIET explained*. 2022. Disponível em: <https://blog-.marvik.ai/2022/06/23/nlp-transformer-diet-explained/>.

SILVA, M. da. Desenvolvimento de *chatbot* usando aprendizagem de máquina como auxílio para responder perguntas sobre vacinas para covid-19. 2021.

SOUSA, A. W.; FABRO, M. Iudicium textum dataset uma base de textos jurídicos para nlp. In: . [S.l.: s.n.], 2019.

SOUZA, F.; NOGUEIRA, R.; LOTUFO, R. Bertimbau: Pretrained bert models for brazilian portuguese. In: CERRI, R.; PRATI, R. C. (Ed.). *Intelligent Systems*. Cham: Springer International Publishing, 2020. p. 403–417. ISBN 978-3-030-61377-8.

SUN, X. et al. *Interpreting Deep Learning Models in Natural Language Processing: A Review*. 2021.

VASWANI, A. et al. Attention is all you need. In: GUYON, I. et al. (Ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017. v. 30. Disponível em: <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf>.

WALLACE, R. *The elements of AIML style. ALICE AI Foundation.* 2004.

WALLACE, R. S. The anatomy of a.l.i.c.e. In: _____. *Parsing the Turing Test: Philosophical and Methodological Issues in the Quest for the Thinking Computer*. Dordrecht: Springer Netherlands, 2009. p. 181–210. ISBN 978-1-4020-6710-5. Disponível em: <https://doi.org/10.1007/978-1-4020-6710-5_13>.

WARMERDAM, V. *How to Use BERT in Rasa NLU*. 2022. Disponível em: <https://rasa.com/blog/how-to-benchmark-bert/>.

WEIZENBAUM, J. Eliza—a computer program for the study of natural language communication between man and machine. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 9, n. 1, p. 36–45, jan 1966. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/365153.365168>.

YANG, Y. et al. Improving multilingual sentence embedding using bi-directional dual encoder with additive margin softmax. *CoRR*, abs/1902.08564, 2019. Disponível em: <http://arxiv.org/abs/1902.08564>.

YANG, Z. et al. Xlnet: Generalized autoregressive pretraining for language understanding. In: WALLACH, H. et al. (Ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019. v. 32. Disponível em: <https://proceedings.neurips.cc-/paper_files/paper/2019/file/dc6a7e655d7e5840e66733e9ee67cc69-Paper.pdf>.

A CONFIGURAÇÃO DA FERRAMENTA

Neste apêndice será explicado como executar o *software* do lado servidor. Antes de iniciar, certifique-se que o RASA e o NodeJS estão instalados. Depois, abra dois terminais. Em um deles será executado o RASA e em outro o *software* que intermediará a troca de mensagens entre o RASA e o WhatsApp.

Num dos terminais, partindo da raiz do projeto, execute o seguinte comando para treinar o modelo no RASA. Esse comando só é necessário se for a primeira execução, ou se dados novos de treinamento forem adicionados.

```
rasa train
```

O próximo comando executará o projeto no endereço http://0.0.0.0:5005. Acompanhe o log do terminal para garantir que ele está realmente executando no endereço esperado.

```
rasa run -p 5005
```

Agora, no outro terminal, partindo da pasta raiz do projeto, execute os seguintes comandos para instalar as dependências do *software* que intermediará a troca de mensagens.

```
cd client
npm install
```

Como dica, certifique-se de que está utilizando a última versão da biblioteca Venom Bot. Algum mau funcionamento da ferramenta pode estar diretamente relacionado as constantes atualizações do WhatsApp, e o Venom Bot deve estar sempre atualizado para manter a troca de mensagens funcionando. No arquivo package.js é possível alterar sua versão, e no site https://www.npmjs.com/package/venom-bot pode-se verificar qual é a versão mais recente. Lembrando de executar sempre *npm install* após qualquer alteração. Agora o comando para executar o software.

```
npm start
```

Em determinado momento será mostrado um *qrcode* em tela. Ele deverá ser reconhecido por um *smartphone* usando o aplicativo WhatsApp. Toda a troca de mensagens

Figura 28 – Conexão do *software* com o WhatsApp

acontecerá através desse aparelho, de modo que o *bot* assumirá o controle das mensagens que serão respondidas no WhatsApp a partir desse momento.

Finalmente, o *software* está executando e pronto para receber e responder as mensagens do usuário. Lembrando que o número de telefone utilizado para troca é o número registrado no WhatsApp do *smartphone* conectado ao *software*.

B CONFIGURAÇÃO DE *PIPELINE*

Neste apêndice estão as configurações a que se referem a Seção 4.4.

```
# DIET
pipeline:
  - name: WhitespaceTokenizer
  - name: RegexFeaturizer
  - name: CountVectorsFeaturizer
  - name: CountVectorsFeaturizer
    analyzer: char_wb
    min_ngram: 1
    max_ngram: 4
  - name: DIETClassifier
    epochs: 100
    constrain_similarities: true
  - name: FallbackClassifier
    threshold: 0.7
    ambiguity_threshold: 0.1

# DIET + BERTimbau
pipeline:
  - name: WhitespaceTokenizer
  - name: LanguageModelFeaturizer
    model_name: "bert"
    model_weights: "neuralmind/bert-base-portuguese-cased"
  - name: RegexFeaturizer
  - name: CountVectorsFeaturizer
  - name: CountVectorsFeaturizer
    analyzer: char_wb
    min_ngram: 1
    max_ngram: 4
  - name: DIETClassifier
    epochs: 100
    constrain_similarities: true
  - name: FallbackClassifier
    threshold: 0.7
    ambiguity_threshold: 0.1
```