MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
FARROUPILHA CAMPUS PANAMBI

A INTELIGÊNCIA ARTIFICIAL POR TRÁS DO APLICATIVO PAPIBOT

TRABALHO DE CONCLUSÃO DE CURSO

Victor Picinini Lengler

Panambi, RS, Brasil - 2021

A INTELIGÊNCIA ARTIFICIAL POR TRÁS DO APLICATIVO PAPIBOT

por

Victor Picinini Lengler

Monografia apresentada ao Instituto Federal de Educação Ciência e Tecnologia Farroupilha, como requisito parcial para obtenção do título de Tecnólogo em Sistemas para Internet.

Orientador: Prof. Jenifer Heuert Konrad

Panambi, RS, Brasil

2021

Ministério da Educação
Secretaria de Educação Profissional e Tecnológica
Instituto Federal de Educação Ciência e Tecnologia Farroupilha

A Comissão Examinadora, abaixo assinada,
aprova a Monografia

A INTELIGÊNCIA ARTIFICIAL POR TRÁS DO APLICATIVO PAPIBOT

elaborada por
Victor Picinini Lengler

como requisito parcial para obtenção do título de
Tecnólogo em Sistemas para Internet

COMISÃO EXAMINADORA

_____

Jenifer Heuert Konrad, M.Sc
(Orientador)

_____

Cleber Rubert, M.Sc (IFFar)

_____

Magnos Roberto Pizzoni, M.Sc (IFFar)

Conceito Final: _____

Panambi, 24 de novembro de 2021.

RESUMO

O Papibot é um aplicativo que traz como proposta auxiliar no aprendizado da língua inglesa através do diálogo do usuário com uma inteligência artificial simulando um interlocutor. Através deste trabalho buscou-se a melhor forma de desenvolvimento e treinamento de um algoritmo de *machine learning* que, aplicando Processamento de Linguagem Natural, compreenda a fala do usuário e possa produzir uma resposta contextualizada, compreensível e engajadora. Foram analisados objetivamente os resultados obtidos na implementação de duas ferramentas de código aberto - DeepPavlov e DialoGPT - usando um *dataset* contendo diálogos com linguagem natural. As diferenças mais discrepantes entre as duas opções são a exatidão da DeepPavlov e a flexibilidade na geração de textos da DialoGPT. Através da comparação dos algoritmos, foi constatado que a DialoGPT se saiu melhor em três dos cinco fatores decisivos - Performance, Facilidade de Implementação e Flexibilidade -, assim confirmando sua escolha como melhor forma de implementar o serviço.

Palavras-chave: algoritmos; limpeza de dados; inteligência artificial; processamento de linguagem natural.

ABSTRACT

Papibot is a mobile app that comes to assist English learning through the dialogue between the user and an artificial intelligence simulating an interlocutor. Through this paper it was seeked the best way to develop and train a machine learning algorithm that, applying Natural Language Processing, understands the user's speech and can produce a contextualized, understandable and engaging response. Were then analyzed the results obtained in the implementation of two open source tools - DeepPavlov and DialoGPT - using a dataset containing natural language dialogs. The most discrepant differences between the two options are DeepPavlov's accuracy and DialoGPT's flexibility in generating texts. By comparing the algorithms, it was found that DialoGPT did better in three of the five decisive factors - Performance, Ease of Implementation and Flexibility -, thus confirming its choice as the best way to implement the service.

Keywords: algorithms; data cleansing; artificial intelligence; natural language processing.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS

AGI	*Artificial General Intelligence* (Inteligência Artificial Geral)
AI	*Artificial Intelligence* (Inteligência Artificial)
AIML	*Artificial Intelligence Markup Language* (Linguagem de Marcação de Inteligência Artificial)
ANI	*Artificial Narrow Intelligence* (Inteligência Artificial Restrita)
ASI	*Artificial Superintelligence* (Superinteligência Artificial)
CDD	*Conversation-Driven Development* (Desenvolvimento Focado em Conversação)
DL	*Deep Learning* (Aprendizado Profundo)
GUI	*General User Interface* (Interface Geral de Usuário)
IA	Inteligência Artificial
JSON	Javascript *Object Notation* (Notação de objeto em Javascript)
ML	*Machine Learning* (Aprendizado de Máquina)
NLG	*Natural Language Generation* (Geração de Linguagem Natural)
NLP	*Natural Language Processing* (Processamento de Linguagem Natural)
ODQA	*Open Domain Question Answering* (Resposta a Perguntas de Domínio Aberto)
QnA	*Question and Answer* (Pergunta e Resposta)
TCC	Trabalho de Conclusão de Curso
TTS	*Text to Speech* (Conversão de Texto em Fala)

SUMÁRIO

1 INTRODUÇÃO

O presente Trabalho de Conclusão de Curso foi desenvolvido durante o quinto semestre do curso superior de Tecnologia em Sistemas para Internet do Instituto Federal Farroupilha - Campus Panambi.

O tema se constitui na pesquisa da melhor forma de implementar uma Inteligência Artificial com base em processamento de linguagem natural focando na conversação contextual em língua inglesa para uso no Papibot, e a implementação da mesma.

O Papibot é um aplicativo *mobile* híbrido (Android e iOS) que ainda está em sua fase de desenvolvimento, seu foco é ajudar na prática e aprendizado de língua inglesa. Ele ainda está em fase de desenvolvimento. A proposta foi elaborada ao ver possibilidade de uso e venda no mercado dessa aplicação, visto que tal implementação ainda não foi colocada em produção em nenhuma das lojas nativas - App Store e Google Play - de aplicativos mobile.

O desenvolvimento da tecnologia de Inteligência Artificial é o cerne do serviço, visto que o aplicativo precisa se comunicar com ela para poder responder ao usuário com base no contexto da conversa.

Na pesquisa da melhor forma de implementar a IA, foi observado quais bibliotecas e/ou linguagens melhor se encaixam para uma entrega rápida e eficiente do serviço.

Com esse projeto, espera-se ampliar as opções de como as pessoas usam seu celular para a prática de língua inglesa e trazer uma nova forma de entrega desse tipo de aplicativo para o mercado. Assim podendo influenciar novos projetos na mesma área de conhecimento.

1.1 CONTEXTUALIZAÇÃO DO ESTUDO

1.1.1 Apresentação do tema

Com a internet, o ser humano passou a ter acesso a informações do mundo todo e em todas as línguas. Se a necessidade de aprendizado da língua inglesa já era importante antes, essa demanda só cresceu após o surgimento dessa rede internacional. Em 2020, o mundo sofreu com uma pandemia que obrigou as empresas a repensarem sua forma de trabalho, e, com o "*home office*", o emprego no exterior passou a ser uma opção viável, pelo menos para quem tem o domínio de um idioma estrangeiro. Esse fator aumentou ainda mais a necessidade do inglês como segunda língua.

A pandemia também teve um impacto no bolso dos trabalhadores, com menos movimento os empresários foram obrigados a cortar custos, causando demissões e redução dos pagamentos. Juntando esses fatores, tem-se pessoas que precisam desenvolver a prática com a língua inglesa, tem pressa nesse aprendizado, mas que não tem o dinheiro para pagar por cursos caros, que normalmente entregam mais resultados em menos tempo, principalmente através do foco na conversação.

Levando em conta os pontos acima descritos, o Papibot foi pensado e estruturado com o objetivo de facilitar a prática de língua inglesa, por um preço acessível e a qualquer hora do dia, através de um aplicativo para celular. O funcionamento dessa aplicação consiste em usar uma Inteligência Artificial para conversar com o usuário via texto, em tempo real, simulando um ambiente real de diálogo. Além disso, a proposta do Papibot é ajudar o estudante a melhorar sua escrita através de um corretor de gramática e sugestões contextualizadas com o assunto da fala.

Então, para alcançar esse objetivo, o Papibot precisa de um "cérebro" que possa lhe ajudar a responder o usuário de forma convincente e manter o fluxo do diálogo. Tal tecnologia é o que chamamos de Inteligência Artificial, que precisa ser implementada através de um algoritmo que entenda a linguagem de um ser humano.

1.1.2 Problema

Existem, hoje, inúmeras maneiras de se implementar uma inteligência artificial focada em conversação, cada qual com sua solução diferente para um problema específico. Neste projeto, o principal fator para uma conversação é a capacidade de manter um contexto e, a

partir disso, ter a habilidade de sugerir correções e opções para falas do usuário. Por isso, a pesquisa de uma implementação que possa oferecer tais condições é tão importante para a qualidade do resultado final.

Através da pesquisa para o projeto, o seguinte problema foi encontrado: **De que forma será possível implementar um algoritmo para um serviço de inteligência artificial para auxiliar no estudo e na prática de língua inglesa através de um diálogo contextualizado?**

1.1.3 Objetivo

Objetivo geral

Implementar um algoritmo para um serviço de inteligência artificial para auxiliar no estudo e na prática de língua inglesa através de um diálogo contextualizado, simulando assim aulas de conversação entre o usuário e o aplicativo Papibot para um melhor aprendizado do idioma.

Objetivos específicos

a) Realizar uma pesquisa buscando os melhores algoritmos e bibliotecas para implementação de uma Inteligência Artificial Conversacional fazendo uso de NLP (Natural Language Processing ou Processamento de Linguagem Natural);

b) Implementar na prática o código da Inteligência Artificial (IA) usando as bibliotecas encontradas;

c) Comparar objetivamente os resultados entre os algoritmos e bibliotecas implementados, em busca da melhor resolução do problema acima descrito.

1.1.4 Justificativa

O projeto difere-se de outros aplicativos já existentes no mercado, define-se original por não existir hoje, no mercado, um aplicativo com o mesmo funcionamento, mesmo intuito e/ou mesma forma de venda, isto é, que forneça ao usuário uma IA conversacional que consiga manter um diálogo contextualizado e possa ainda corrigir a gramática da entrada de texto, sugerindo ao usuário a forma gramatical mais correta de se expressar. Por isso, o

Papibot traz uma forma não-convencional de se aprender o idioma inglês, fazendo uso de Inteligência Artificial. A comercialização do mesmo pode ser feita através de planos com base mensal ou pela venda para redes de escolas de inglês disporem a seus alunos.

A implementação da Inteligência Artificial do aplicativo é de extrema importância para o produto, visto que o Papibot se baseia completamente no funcionamento da mesma.

A viabilidade desse projeto se dá por parte do conhecimento na área ser de interesse do estudante de Sistemas para Internet, e pela numerosa quantidade de algoritmos com propostas similares na internet que podem ser usados como inspiração para a produção do algoritmo final.

2 REFERENCIAL TEÓRICO

2.1 APRENDIZADO DE IDIOMAS

Aprender idiomas está passando a não ser mais uma opção para as pessoas que visam ter um futuro melhor, e sim quase uma obrigação, pois como hoje estamos praticamente ligados com o mundo todo, assim como a inúmeros produtos e tecnologias de outros países cabe a nós aprimorarmos nossos conhecimentos em línguas para podermos ter uma visão mais abrangente acerca das oportunidades, não só de vida pessoal, mas também profissional. Neste sentido, foi realizado um estudo pelo autor Vilson Leffa pela Universidade Católica de Pelotas/CNPq onde descreve as tendências históricas do ensino de línguas, envolvendo as mudanças no conceito de língua, na metodologia de ensino, no papel do professor e as relações que se estabelecem entre esses três componentes.

> Historicamente o que aconteceu com o ensino de línguas no Brasil tem sido um eco do que aconteceu em outros países, geralmente com um retardo de alguns decênios, tanto em termos de conteúdo (línguas escolhidas) como de metodologia (método da tradução, método direto, etc.). O método direto, por exemplo, foi introduzido no Brasil em 1931, ou seja, 30 anos depois de sua implementação na França. (LEFFA, 2013, p. 2)

Em relação à língua, mostra o desenvolvimento de uma ênfase histórica no código, que posteriormente evolui para uma ênfase no sentido e chega à ideia de língua como ação. Em termos de metodologia, descreve a mudança que ocorreu entre o conceito de método, visto como solução universal, para o conceito de pós-método, com ênfase no contexto de aprendizagem. Finalmente, em relação ao professor, o texto mostra como seu papel tem mudado à medida que mudamos conceitos de língua e de método, passando da subordinação ao método para o exercício da autonomia. (LEFFA, 2013). Para o autor, o que aconteceu no ensino de línguas no Brasil é resultado do que aconteceu nos outros países também.

2.2 INTELIGÊNCIA ARTIFICIAL

A Inteligência Artificial (IA) é uma área da informática que busca desenvolver aplicações que simulem, da melhor maneira possível, a inteligência humana de forma artificial através de métodos de *Machine Learning* (ML ou Aprendizado de Máquina, em português).

Nilsson (1982) afirmou que “Muitas atividades humanas como escrever programas de computador, fazer cálculos, entender senso comum, compreender línguas e até mesmo dirigir um automóvel são consideradas atos que precisam de ‘inteligência’”. Ele também já em sua época percebeu que “Nas últimas décadas, numerosos sistemas de computadores foram construídos para executar tais tarefas. [...] Nós podemos dizer que tais sistemas possuem algum grau de Inteligência Artificial”. De sua época para cá, os algoritmos nesses sistemas evoluíram ao ponto de não ser possível prever o que a próxima IA será capaz de fazer.

Aluri (1988) define esse tipo de software como “Um fenômeno conhecido como ‘Sistema Especialista’”. Isso por sua capacidade de resolver com perfeita acurácia problemas específicos para os quais a sua Rede Neural foi arquitetada.

É importante ressaltar que, segundo Strelkova (2017), “existem três níveis de Inteligência Artificial: ANI, AGI e ASI”. E, nesse ponto, questões filosóficas podem ser levantadas. Strelkova (2017) continua explicando:

> “ANI (Inteligência Artificial Restrita) - é o primeiro nível que pode fazer um ótimo trabalho em somente um problema. Por exemplo, há uma IA que pode derrotar o campeão mundial de xadrez, mas essa é a única atividade que ela executa.
> AGI (Inteligência Artificial Geral) - IA que alcança e então passa o nível de inteligência humana, o que significa que ela tem a habilidade de 'raciocinar, planejar, resolver problemas, pensar abstratamente, compreender ideias complexas, aprender rapidamente, e aprender com a experiência'.
> ASI (Superinteligência Artificial) - um intelecto que é muito mais esperto que o melhor cérebro humano em praticamente qualquer campo, incluindo criatividade científica, sabedoria geral e habilidades sociais.”

Através da popularização de tais algoritmos, sua comercialização foi inevitável, a IA passou a ser um produto vendido por empresas de software mundialmente famosas como Google, Amazon, Microsoft e Apple. Métodos de ciência de dados foram desenvolvidos e o Machine Learning passou a ser parte do processo de entrega de produtos digitais em toda a rede mundial. Companhias inteiras foram criadas tendo como sua única entrega serviços de IA para outras empresas consumirem. Assim, nomes famosos, como Elon Musk, criador da OpenAI - uma das maiores empresas do ramo - investem valores significativos para impulsionar essa ideia.

Hoje, falamos de um mercado multimilionário com vagas ilimitadas para aqueles que se interessam. Porém esse campo também exige competências, conhecimentos e capacidades somente desenvolvidas com um esforço árduo e contínuo. A cada dia, um novo algoritmo surge e novas formas de resolver o mesmo problema se mostram possíveis.

2.3 NLP (PROCESSAMENTO DE LINGUAGEM NATURAL)

Com o aprofundamento na área de Machine Learning, é perceptível que as possibilidades de uso são ilimitadas. O setor de IA deste estudo visa aprimorar o Processamento de Linguagem Natural (ou Natural Language Processing).

Chowdhury (2003) define o Processamento de Linguagem Natural como "uma área de pesquisa e aplicação que explora como computadores podem ser usados para entender e manipular textos ou falas em linguagem natural para fazer coisas úteis". Entendendo "linguagem natural" como a forma como o ser humano se comunica usando de palavras faladas ou escritas, Chowdhury (2003) continua dizendo que "os pesquisadores de NLP buscam juntar conhecimento sobre como seres humanos entendem e usam a língua para desenvolver as ferramentas e técnicas necessárias para sistemas de computadores fazerem o mesmo."

Através de Redes Neurais específicas, é possível extrair métricas, estatísticas, contexto, sentimentos, entre outros recursos, de um texto - que no presente trabalho será totalmente escrito na língua inglesa.

Talvez esse nome não seja facilmente reconhecido. Mas é provável que a maior parte dos usuários já utilizou IAs de NLP no seu dia-a-dia sem saber o que acontecia nos bastidores. Uma ferramenta famosa e usualmente utilizada é o Google Tradutor, um algoritmo da Google para traduzir textos utiliza de forma abrangente técnicas de NLP para compreender o texto digitado, achar uma tradução separando por sentenças, termos compostos ou palavras, e ainda retornar ao usuário de forma coesa.

A área de NLP se divide em dois conceitos: NLU (Natural Language Understanding) e NLG (Natural Language Generation), cada um tem um importante papel dentro do que chamamos de Processamento de Linguagem Natural, uma como forma de entrada (input) e outra apresenta a saída da criação de linguagem (output). A seguir, esses conceitos serão explicados mais a fundo.

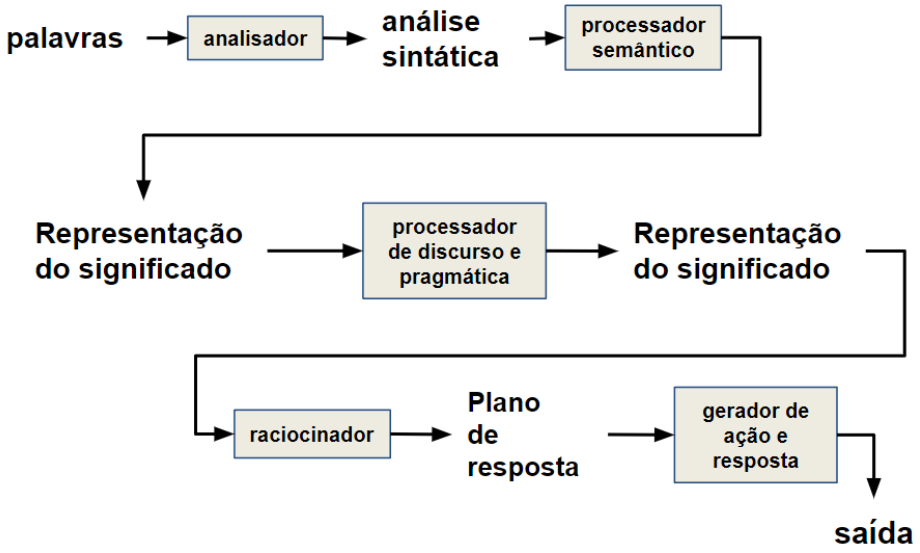
2.3.1 NLU (Entendimento de Linguagem Natural)

Quando se fala em processamento de linguagem natural, a primeira coisa que vem à mente é a capacidade da máquina de interpretar uma entrada de texto (ou áudio). Essa é a parte da NLP pela qual é responsável a NLU.

Segundo BATES (1995), a sequência de processamento de um sistema de

processamento de linguagem natural pode ser representada pela Figura 1, sendo a estrutura NLU mostrada até o componente “Meaning Representation” e, tudo após, NLG.

Figura 1 - Sequência de componentes em um sistema NLP genérico.



Fonte: Bates, 1995. Adaptada pelo autor.

O sistema de NLU, segundo BATES (1995), precisa dos seguintes passos analíticos para interpretar uma entrada (input) de linguagem natural: Sintaxe, Semântica, Discurso e Pragmática.

Sintaxe

BATES (1995) traz a análise sintática com dois usos: “simplificar o processo de extração de significado do input dos componentes subsequentes” e “ajudar na detecção de significados novos ou anormais”.

Semântica

A análise de semântica é uma parte muito importante e um problema difícil de resolver. BATES (1995) explica que “a saída do componente de semântica é o ‘significado’ da entrada”. O problema está em expressar esse significado, diz BATES (1995) diz que “esse ‘significado’ pode ser expressado por uma sequência de palavras”.

Há quem diga que não existe um único significado exato para uma sequência de texto. Então o algoritmo NLU precisa chegar o mais perto possível desse resultado.

Discurso e Pragmática

Interpretar o contexto, e usá-lo corretamente, é um dos aspectos mais complicados da NLU. A análise do discurso e da pragmática é a "difícil tarefa de determinar as referências de pronomes e nomes definidos e tentar entender fragmentos de frases elípticas, artigos [...], e outras formas de linguagem não-padrão", BATES (1995).

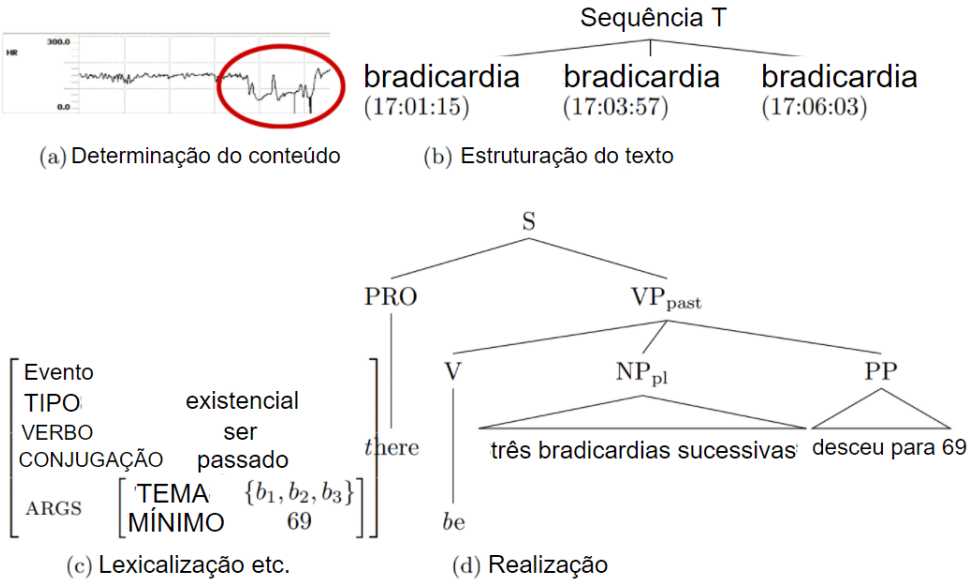
2.3.2 NLG (Geração de Linguagem Natural)

Ainda que uma máquina possa entender a linguagem de um ser humano, para haver um diálogo, é necessário que ela possa também responder de forma que o ser humano entenda. Para isso surgiram os algoritmos de geração de texto natural, batizados de algoritmos NLG (Natural Language Generation).

Para que um computador possa resolver o problema de converter uma entrada textual para uma resposta igualmente compreensível por humanos, seu algoritmo tradicionalmente precisa executar seis tarefas em ordem (Date e Reiter, 1995), tais tarefas apresentadas abaixo, e também visíveis de forma sistemática na Figura 1:

1. Definição do Conteúdo: "Decidir qual informação incluir no texto em construção" (Gatt, 2018);
2. Estruturação do texto: "Determinar em qual ordem a informação será apresentada no texto" (Gatt, 2018);
3. Agregação da Frase: "Decidir qual informação apresentar em frases individuais" (Gatt, 2018);
4. Análise Léxica: "Encontrar as palavras e frases corretas para expressar a informação" (Gatt, 2018);
5. Geração de Expressão Referencial: "Selecionar as palavras e frases para identificar como objetos de domínio" (Gatt, 2018), para poderem ser referenciados em outras partes do texto alcançando o mínimo de repetição;
6. Realização Linguística: "Combinar todas as palavras e frases em sentenças bem formadas" (Gatt, 2018).

Figura 2 - Tarefas em um sistema NLG.



Fonte: Gatt, 2018. Adaptado pelo autor.

Algoritmos de NLG são comumente usados em robôs de conversação (os Chatbots) para uma melhor interação entre o usuário e o computador. Hoje, podemos ver alguns exemplos de produtos de empresas grandes na área de IA - como OpenAI - que são capazes de feitos que desafiam o humano a distinguir entre um computador ou outra pessoa do outro lado da conversa.

3. METODOLOGIA

3.1 CLASSIFICAÇÃO DA PESQUISA

Este estudo classifica-se, quanto à sua natureza, em uma pesquisa aplicada pois tem como objetivo buscar conhecimentos específicos para a solução do problema apresentado, subsidiando assim a aplicação prática do código, focando no desenvolvimento da inteligência artificial.

A abordagem da pesquisa escolhida foi a quantitativa, devido ao uso de técnicas estatísticas como média, porcentagem, comparação de assertividade, confiança e performance das implementações de algoritmos. Os algoritmos e bibliotecas foram programaticamente testados com os mesmos dados, e suas respostas comparadas objetivamente.

O estudo realizado foi exploratório, visando a investigação de uma determinada situação, gerando assim familiaridade com o assunto, gerando maior compreensão a respeito do mesmo.

Quanto ao procedimento técnico da pesquisa, tal classifica-se como bibliográfica, pois fez uso de referencial teórico e prático dentro da área de Aprendizado de Máquina e mais especificamente Processamento de Linguagem Natural em busca de algoritmos para chegar na melhor implementação focando na resolução do problema.

3.2 OBJETO DE ESTUDO

O Objeto de estudo é a Inteligência Artificial para o aplicativo Papibot, que foi desenvolvida usando os conhecimentos adquiridos na pesquisa.

O Papibot tem, como sua principal característica, o diálogo natural através de texto entre humano e máquina para a prática de língua inglesa.

Portanto o estudo visa aplicar os conhecimentos na área de Inteligência Artificial pesquisados para trazer ao Papibot a capacidade de construir tal diálogo humano-máquina.

3.3 PLANO DE COLETA DE DADOS

Após encontrar os algoritmos a serem analisados, todos serão treinados e testados usando o dataset “Topical Chat Dataset”, desenvolvido pela Amazon, que contém diálogos em

inglês criados especificamente para treinar IAs conversacionais de socialbots, como a Alexa. Ele é encontrado em <https://github.com/alexa/Topical-Chat> acesso em 06 de junho de 2021.

Em sequência, os algoritmos serão testados em um mesmo diálogo de bate-papo sem um objetivo específico (com o objetivo de apenas manter a conversa engajada). Através disso, será possível analisar a capacidade da IA de se comportar em uma conversa no mundo real.

3.4 PLANO DE ANÁLISE E INTERPRETAÇÃO DOS DADOS

Após a limpeza do dataset utilizado e o treinamento e testes serem feitos usando o algoritmo, a análise do estudo se deu através de uma comparação entre os resultados dos testes.

A comparação entre os algoritmos foi feita a partir dos seguintes conceitos objetivos: Compreensão, Geração, Performance, Facilidade de Implementação e Flexibilidade. Todos estão descritos abaixo.

3.4.1 Compreensão

O fator "compreensão" foi pontuado a partir da F1 *Score* do algoritmo. “A F1 *Score* é definida como a média harmônica entre precisão e *recall*” (Dalianis, 2018, p. 47), ela mostra a assertividade do algoritmo em encontrar o significado e contexto da entrada de texto. Isso é importante para a implementação para manter o contexto do diálogo com o usuário. Essa pontuação é calculada a partir da seguinte forma:

$$F1 = \frac{2 \times (\text{precisão} \times \text{sensibilidade})}{(\text{precisão} + \text{sensibilidade})} \qquad (1)$$

Para resolver a F1, precisamos calcular os seguintes fatores:

- **Precisão**: é o número de verdadeiros positivos para a classe detectada dividido pelo total de verdadeiros positivos e falsos positivos;
- **Sensibilidade**: é calculada a partir do número de resultados verdadeiros positivos dividida pela quantidade de verdadeiros positivos mais a quantidade de falsos negativos. Também é chamada de *Recall*, em inglês.

3.4.2 Geração

Dado em porcentagem (%), o fator “geração” julga a capacidade do algoritmo em gerar um texto-resposta com o contexto correto e compreensível. A IA precisa ser capaz de dar uma resposta que feche com o contexto recebido e ainda assim abra espaço para uma continuação por parte do usuário. A geração será calculada a partir da acurácia (número de positivos dividido pelo total) dos resultados retirados do processo de jogar a resposta de um algoritmo para os outros e comparando o contexto gerado com o contexto detectado.

3.4.3 Performance

Dado em milissegundos (ms), o conceito “performance” demonstra o tempo que o algoritmo precisa para gerar uma resposta. O algoritmo precisa ser rápido o suficiente com poucos recursos para responder o usuário em tempo real.

3.4.4 Facilidade de Implementação

Julgado em um intervalo de 1 a 5, sendo 1 extremamente complexo e necessitando de mais recursos e 5 simples e com poucos recursos. O fator “facilidade de implementação” indica a dificuldade de desenvolvimento do algoritmo em linhas de código, também leva em conta a quantidade de recursos necessários para realizar o processo. O algoritmo precisa ser simples o suficiente para ser implementado durante o decorrer de um mês e ter fácil manutenção. O julgamento desse fator será feito pelo desenvolvedor durante a implementação do código.

3.4.5 Flexibilidade

A flexibilidade do algoritmo será categorizada entre 1 e 5. Ela corresponde à capacidade do algoritmo de trabalhar com frases desconhecidas - não pré-programadas ou usadas durante o treino. Por exemplo, há algoritmos que sempre caem na mesma resposta usando o sentimento da pergunta ou o contexto geral. Porém há outros que podem gerar uma resposta completamente nova dentro do mesmo contexto, só por mudar algo na fala do usuário. O número 1 representa uma baixa flexibilidade e, opostamente, 5 se dá a uma alta flexibilidade.

4 DESENVOLVIMENTO

4.1 DADOS

Segundo Gopalakrishnan et al. (2019) o Topical-Chat é “um *dataset* de conversas entre humanos fundamentado em conhecimento que tem como base adjacente o conhecimento em 8 tópicos abrangentes e onde os parceiros do diálogo não têm papéis definidos”. O *dataset* conta com mais de oito mil diálogos diferentes.

Os dados estão dispostos em uma tabela de mais de 180 mil linhas contendo as colunas “conversation_id”, “message” e “sentiment” - dessas, sendo as mais importantes “conversation_id” e “message” -, que precisará passar por um processo de limpeza e organização antes de ser usada nos algoritmos.

Diferentemente de outros *datasets* encontrados na internet, que contém uma coluna com a entrada de um usuário e a coluna seguinte representa a resposta imediata, para descobrir a resposta de cada mensagem, deve-se concatenar com a próxima mensagem com o mesmo id.

Para algoritmos que trabalham com o conceito QnA (Pergunta e Resposta) - onde cada entrada do usuário é tratada como uma pergunta para a máquina responder -, os dados foram dispostos em uma nova tabela contendo as colunas *question* e *answer*. A cada iteração no Topical-Chat - com exceção da última de cada diálogo -, o dado *message* foi posto na coluna “question”, e “answer” foi retirada da próxima linha. Então as *messages* podem ser consideradas perguntas e respostas ao mesmo tempo.

Já para algoritmos que trabalham com interpretação do histórico do diálogo de forma recursiva (GPT-2), todas as mensagens de uma mesma conversa foram concatenadas em uma única *string*, assim formando um grande texto contendo a sequência inteira de falas.

A coluna contendo o sentimento foi apenas usada em implementações que levam em conta essa variável para definir o contexto da conversa. Por isso a sua relevância é menor.

4.2 IMPLEMENTAÇÃO

Através de pesquisas em artigos na internet, em repositórios no *GitHub* e em sites sobre Inteligência Artificial, foram encontrados as seguintes bibliotecas para serem estudadas e implementadas para desenvolver a IA do Papibot: DeepPavlov, Rasa, DialoGPT e ParlAI. A

DeepPavlov engloba implementações como a Rasa e a ParlAI dentro de suas configurações padrão, portanto o projeto se resume em aplicar a DeepPavlov e a DialoGPT. As duas serão descritas e aprofundadas abaixo.

4.2.1 DeepPavlov AI

Como é descrito em seu site, o DeepPavlov é "um *framework* de código aberto para desenvolvimento de *chatbots* e assistentes virtuais" (DEEPPAVLOV, 2021). Baseado no TensorFlow e no Keras, ele se destaca por "ter ferramentas flexíveis e compreensíveis que ajudam desenvolvedores e pesquisadores de NLP a criar robôs prontos para produção com habilidades conversacionais complexas" (DEEPPAVLOV, 2021).

Com sua página hospedada em *deeppavlov.ai*, o DeepPavlov foi criado em junho de 2017 e aperfeiçoado desde então, focando na análise de texto e criação de sistemas de diálogos. O projeto tem como responsável o Laboratório de Redes Neurais e *Deep Learning* do Instituto de Física e Tecnologia de Moscou. O laboratório desenvolvedor da ferramenta é parte da Iniciativa de Ciência e Tecnologia do governo russo.

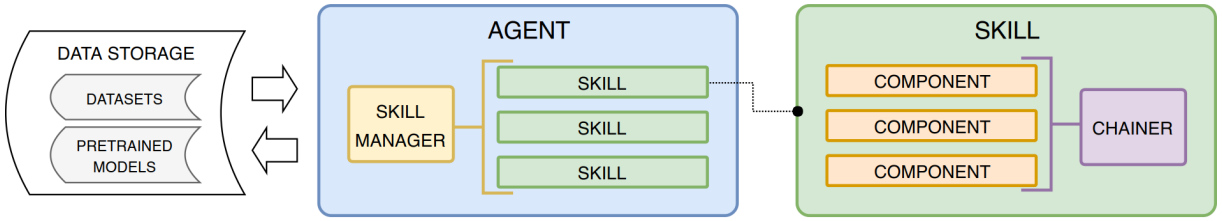
O *framework* já recebeu diversos prêmios no campo de NLP. Entre eles, vitórias em concursos como READ//ABLE, Amazon Alexa Socialbot e TensorFlow Post Challenge.

Visão geral

O DeepPavlov disponibiliza aos desenvolvedores e pesquisadores os seguintes recursos:

- Um conjunto de modelos de NLP já pré-treinados, componentes pré-definidos para sistemas de diálogo utilizando-se de ML, DL ou regras-base e modelos de *pipeline*;
- Um *framework* para implementação e teste de modelos conversacionais;
- Ferramentas para integrar com aplicações de chat e/ou *helpdesks*;
- Ambiente para treinamento e testes de modelos conversacionais e acesso a datasets da empresa.

Figura 3 - Visualização conceitual da arquitetura do DeepPavlov.



Fonte: Documentação oficial do DeepPavlov (docs.deeppavlov.ai), 2021.

A Figura 3, mostra, de modo geral, o funcionamento do ecossistema do DeepPavlov. Pode-se observar que existem conceitos adicionais no framework, como *Skill*, *Chainer* e *Component*.

Conceitos-chave

Agent (presente na Figura 3), no campo de NLP, é "qualquer coisa que percebe o ambiente, toma ações autônomas para atingir um objetivo, e pode melhorar sua performance aprendendo ou usando de conhecimento" (GeekforGeeks, 2021), também conhecido como ***Bot*** em contextos menos formais. No DeepPavlov, o *Agent* é "um framework para desenvolver assistentes virtuais, sistemas de diálogo complexos e *chatbots* escaláveis e prontos para produção" (DeepPavlov, 2018).

Uma ***Skill*** (presente na Figura 3) é a habilidade do ***Agent*** de "atingir o objetivo do usuário em algum campo" (DeepPavlov, 2018), por exemplo: responder alguma pergunta, marcar uma reserva ou outras possíveis tarefas de serviço ao consumidor. Às vezes, porém, o objetivo do modelo é ter uma conversa contínua com o usuário, sem um objetivo final, nesses modelos, o sucesso é definido por manter a conversa, e esse é o intuito do Papibot.

Um ***Component*** (presente na Figura 3) (componente em português) "é uma função ou parte de modelo ou *skill* que pode ser reutilizado" (DeepPavlov, 2018).

Modelos de *Machine Learning* são diferenciados de modelos de *Deep Learning* pela forma como podem ser treinados. No primeiro grupo, cada modelo só pode ser treinado sozinho. Porém os modelos de DL podem ser treinados tanto individualmente quanto em uma sequência do começo ao fim, com a saída de um conectada à entrada do outro.

Além dos modelos ML e DL, o DeepPavlov também suporta um terceiro tipo, os modelos baseados em regras. Esses modelos não podem ser treinados e seu funcionamento é

menos flexível e mais estático que os outros. Os modelos *Rule-based* não serão abordados no presente projeto.

Um ***Chainer*** (presente na Figura 3), segundo DeepPavlov (2018) "cria um modelo de *pipeline* com componentes heterogêneos (ML, DL ou *Rule-based*)", para trazer um melhor resultado fazendo uso do melhor de cada um. Esse recurso permite adicionar diferentes modelos e treiná-los em uma sequência de início ao fim.

Implementação com DeepPavlov

Enquanto muitos frameworks necessitam de código Python para serem implementados, o DeepPavlov pode trabalhar apenas com arquivos de configuração. Nesses arquivos JSON são contidas as informações para o treino, configuração e utilização do modelo.

O arquivo de configuração precisa conter um elemento no JSON: o Chainer, que descreve a pipeline de NLP. A Figura 4 apresenta um exemplo desse modelo:

Figura 4 - Exemplo de configuração de *Chainer*.

```
{
  "chainer": {
    "in": ["x"],
    "in_y": ["y"],
    "pipe": [
      ...
    ],
    "out": ["y_predicted"]
  }
}
```

Fonte: elaborado pelo autor.

A propriedade *pipe* recebe os componentes que formarão a *pipeline* do *Chainer*. Cada um desses componentes precisa conter as suas entradas (*inputs*), as saídas (*outputs*) e a classe que será implementada, podendo também conter um *id* para referência posterior no arquivo. A saída de cada componente será passada para a entrada do próximo, formando uma sequência encadeada do começo ao fim. Um componente pode ter mais de uma saída ou entrada, contanto que corresponda ao número esperado nos componentes vizinhos.

A classe de cada componente na *pipe* pode ser um já definido pelo DeepPavlov (preset) ou pode ser uma classe (filha da classe *Component*) ou uma função desenvolvidas em código Python personalizado. Um exemplo de *pipe* pode seguir a seguinte implementação:

Figura 5 - Exemplo de configuração de *pipeline* com DeepPavlov.

```
{
  "class_name": "deeppavlov.models.preprocessors.str_lower:str_lower",
  "in": ["x"],
  "out": ["x_lower"]
},
{
  "class_name": "nltk_tokenizer",
  "in": ["x_lower"],
  "out": ["x_tokens"]
},
```

Fonte: elaborado pelo autor.

O problema dessa implementação é que as classes descritas na *pipeline* são completamente manuais. Mas essa é a base para todas as abstrações do DeepPavlov criadas posteriormente, como Deepy, D3PO e Dream.

Arquitetura

Na implementação do algoritmo usando o DeepPavlov, foi necessário o uso de um modelo de ODQA (*Open Domain Question Answering*), que trabalha com perguntas e respostas. Nessa arquitetura, a IA considerará cada entrada do usuário como uma pergunta a ser respondida. Para isso, o dataset foi organizado com as colunas question e answer, como já mencionado acima.

A skill ODQA proposta pelo DeepPavlov consiste de dois modelos: um *ranker* e um *reader*. O *ranker* é baseado no modelo DrQA proposto pela Facebook e o *reader* é baseado no R-NET2, implementado pela Microsoft.

Segundo sua página no GitHub, o DrQA é “um sistema para compreensão de leitura aplicado a repostas para perguntas de domínio aberto” (CHEN et al., 2017). Para seu desenvolvimento, foi usado um *dataset* criado a partir da leitura de artigos na Wikipédia como base para treinar os modelos e responder às perguntas. Mas nesse projeto, foi necessário retreinar o modelo com o *dataset* escolhido.

Em seu artigo publicado, a Microsoft descreve o R-NET2 como “um modelo de rede neural ponto a ponto para compreensão de leitura para resposta a perguntas, que foca em responder questões sobre um dado contexto” (GROUP, p. 1, 2017). Para o treino do R-NET2 foi usado o MS-Marco, um *dataset* em larga escala focado na compreensão de leitura de máquinas e resposta a perguntas, também desenvolvido pela Microsoft.

Para executar o algoritmo, foi usada a configuração de *pipeline* “en_odqa_infer_wiki”, já pré-implementada no Deep pavlov, demonstrada no Anexo A. Ela usa a *skill* ODQA mencionada logo acima e mais quatro classes que ajudam a organizar os dados:

- *wiki_sqlite_vocab*: para salvar vocabulário em uma base;
- *document_chunker*: para separar os contextos em pedaços de texto;
- *string_multiplier*: que tem como entrada a pergunta e os pedaços de contexto e transforma em questões já computadas e compreendidas;
- *logit_ranker*: que por sua vez recebe os contextos e questões e retorna as possíveis respostas e suas pontuações.

Treino

Para treinar o DrQA, foi necessário remodelar o arquivo de *dataset* para seguir o formato abaixo. Cada linha do arquivo deve conter um objeto JSON contendo a pergunta e as possíveis respostas, que nesse caso será somente uma.

Figura 6 - Exemplo de formato de *dataset* de treino DrQA.

```
'{"question": "q1", "answer": ["a11", ..., "a1i"]}'
...
'{"question": "qN", "answer": ["aN1", ..., "aNi"]}'
```

Fonte: elaborado pelo autor.

No treino do “SquadModel” do DeepPavlov, baseado no R-NET2, o formato dos dados é um pouco parecido com a geração de texto no DialoGPT, que será mostrada mais à

frente. O dataset constitui-se de uma lista de textos, cada um com um título e uma lista de parágrafos. Os parágrafos são o mais importante aqui, pois eles são injetados no treino da IA.

Cada parágrafo constitui de uma propriedade *context* que é usada para inferir uma resposta, aqui foi colocado todas as falas de um diálogo, para dar o contexto. O parágrafo também possui uma propriedade de lista chamada *qas*, cada elemento corresponde a todas as perguntas prováveis nesse contexto e as respostas para cada uma. Aqui cada fala de um diálogo foi passada como uma pergunta e a resposta foi retirada da fala seguinte. Por exemplo:

Figura 7 - Exemplo de formato de *dataset* de treino R-NET2.

```
{
    "title": "Beyoncé",
    "paragraphs": [
        {
            "qas": [
                {
                    "question": "When did Beyonce start becoming popular?",
                    "answers": [
                        {
                            "text": "in the late 1990s",
                            "answer_start": 55
                        }
                    ]
                }
            ],
            "context": "Born and raised in Houston, Texas, she rose to fame
in the late 1990s."
        }
    ]
}
```

Fonte: elaborado pelo autor.

Depois do tratamento e limpeza dos dados, o treinamento foi realizado usando uma configuração padrão do DeepPavlov: *squad_bert*. Para tal, foi dado o seguinte comando:

Figura 8 - Comando para treinar o DeepPavlov usando arquivo de configuração.

```
python -m deeppavlov train deeppavlov/configs/squad/squad_bert.json
```

Fonte: elaborado pelo autor.

O treino leva algumas horas para ser completado, dependendo da capacidade de processamento e do tamanho do dataset. Após o treinamento do algoritmo, ele pode ser executado interativamente usando o comando *interact*:

Figura 9 - Comando para executar o DeepPavlov usando arquivo de configuração.

```
python -m deeppavlov interact deeppavlov/configs/squad/squad_bert.json
```

Fonte: elaborado pelo autor.

4.2.2 Microsoft DialoGPT

A biblioteca DialoGPT, segundo seus desenvolvedores, pode ser definida como "um modelo ajustável de geração de respostas conversacionais pré treinado em larga escala, com gigabytes de dados do Reddit" (ZHANG et al., 2020, p. 1). Numa conversa do teste de Turing, o modelo foi comparado à resposta humana em qualidade.

Os recentes avanços em arquiteturas baseadas em transformers pré treinadas com quantidades extensas de texto mostraram grandes resultados na compreensão e geração de texto fluente com referências internas e conteúdo rico. Por isso a Microsoft criou esse projeto baseado na OpenAI GPT-2 - que faz a aplicação desses conceitos.

O dataset usado para treinar o DialoGPT foi extraído de discussões no Reddit que datam de 2005 até 2017. Os dados foram filtrados com uma série de critérios para tornar o diálogo o mais fluido e real possível, sem URLs, imagens e comentários pequenos demais para serem compreendidos.

A arquitetura do DialoGPT foi baseada na GPT-2 que usa o modelo de transformer de linguagem genérico (VASWANI et al., 2017), com a principal adição sendo os transformers da hugging face.

O algoritmo produz probabilidades condicionais de resposta baseando-se na seguinte equação (ZHANG et al., 2020, p. 2):

$$p(T|S) = \prod_{n=m+1}^{N} p(x_n \mid x_1, \cdots, x_{n-1}) \qquad (2)$$

Aqui primeiro se concatena todas as falas (x) de um diálogo em um grande texto $x_1, \cdots, x_N$ (sendo N o tamanho da sequência). A função p representa a probabilidade da fala x_n

para todas as falas que vieram antes dessa ($x_1, \cdots, x_{n-1}$). A variável m é a iteração da fala a partir da qual será contado o n. O histórico do diálogo (source) é denotado por $S = x_1, \cdots, x_m$ e a resposta (target) correta (ground truth) é definida por $T = x_{m+1}, \cdots, x_N$.

O cálculo da implementação (2) busca otimizar as possibilidades de resposta para que as com maior probabilidade se assemelhem o máximo possível à target original usando a source em cada input.

Implementação

Para treinar uma IA usando a DialoGPT, o algoritmo de treino precisa ser escrito manualmente, selecionando as propriedades do modelo. O Apêndice A mostra como foi feita a implementação da função *main*, que treina o algoritmo usando os seguintes parâmetros:

- *df_trn*: a tabela com os dados de treino;
- *df_val*: a tabela com os dados de avaliação. Enquanto o algoritmo é treinado, ele usa esses dados como tabela-verdade para saber se o algoritmo está acertando ou errando;
- *args*: um parâmetro opcional que fornece informações como onde será salvo o arquivo do modelo, se esse arquivo será sobrescrito ou se o treino continuará de um *checkpoint* salvo.

Os dois primeiros parâmetros são do tipo DataFrame, isto é, uma tabela criada usando a biblioteca Pandas. Normalmente em ML se usa esse tipo de objeto para distribuir os dados em python, porque a Pandas possui ferramentas que facilitam o manuseio de tabelas e listas.

Para gerar a resposta ao usuário, foi usado o retorno do seguinte código:

Figura 10 - Comando para gerar uma resposta usando o DialoGPT.

```
model.generate(
    bot_input_ids,
    max_length=200,
    pad_token_id=tokenizer.eos_token_id,
    do_sample=True,
    top_k=100,
    top_p=0.9,
    temperature=0.8,
    min_length=3,
)
```

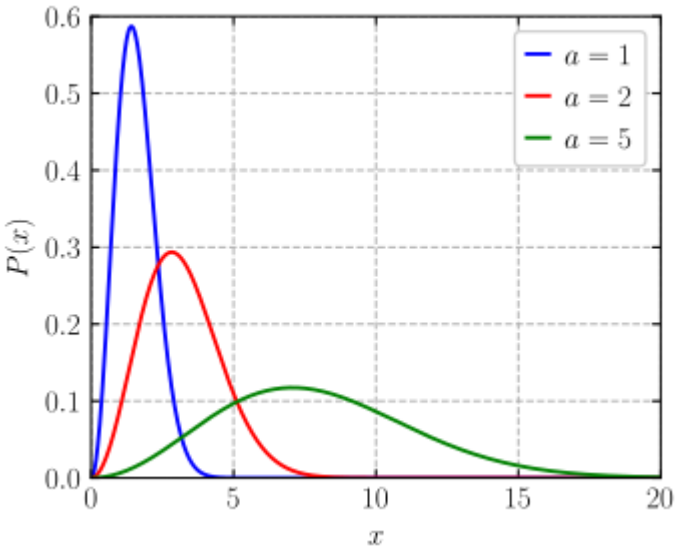
Fonte: elaborado pelo autor.

Temperature

No algoritmo da GPT-2, temos um valor que controla a temperatura da execução (representado pelo argumento “temperature”). A temperatura controla a aleatoriedade da previsão na distribuição de Boltzmann, isto é, o grau de imprevisibilidade da próxima palavra.

A Figura 4 mostra exemplos de como a distribuição de Boltzmann se comportaria com diferentes temperaturas. Nela, x representa a temperatura e $P(x)$ representa a probabilidade das palavras naquele intervalo serem usadas.

Figura 11 - Exemplo de diferentes temperaturas aplicadas à distribuição de boltzmann.



Fonte: Krisgnavedala, 2012.

Como é possível observar, quanto maior a temperatura (x), como na linha verde, maior a gama de palavras a serem usadas para uma mesma sequência, ainda focando no sentido total da frase. Da mesma forma, quando x é um valor menor, a variedade de palavras a ser consideradas diminui. O pico de cada linha representa as palavras com mais probabilidade de serem usadas na geração de texto.

Quando a temperatura atinge zero, o modelo se torna repetitivo e determinístico, sempre apresentando os mesmos resultados. Para o Papibot, o valor de “temperature” usado foi 0.8, para uma maior aleatoriedade de palavras a serem usadas ao completar uma frase.

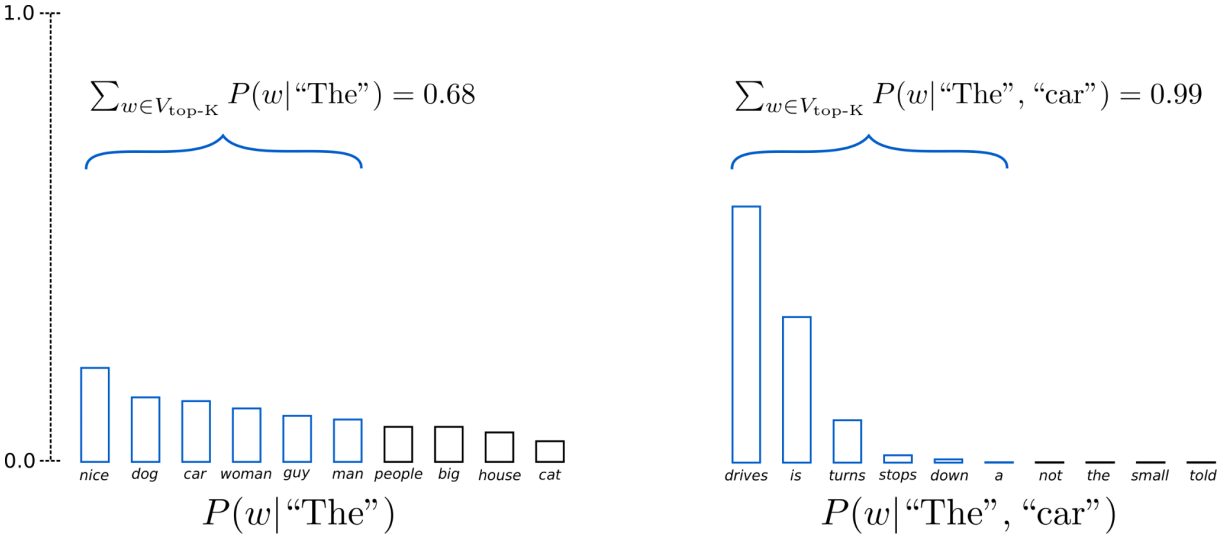
Top-K Sampling

O argumento “top_k” da função controla a diversidade. É dado em um número inteiro que prevê quantas palavras serão consideradas em cada passo do processo de geração. Nesse método de amostragem, o número K de palavras mais prováveis são filtradas e então a massa de probabilidade é distribuída nas próximas K palavras a serem consideradas.

Passar o valor de 1 (um) para esse parâmetro significa que, a cada passo, o algoritmo vai gerar uma resposta usando apenas a resposta mais provável e no passo seguinte, considerar apenas a última palavra, resultando num valor mais determinístico e com menos contexto.

Quando se dá um valor mais alto para esse parâmetro - como mostrado na Figura 5, representado pelo número de palavras com a barra azul em cima -, uma maior parte das amostras é usada para a previsão, trazendo mais contexto, diversidade e compreensão para a resposta. O valor 0 (zero) é um indicativo de nenhuma restrição quanto à diversidade.

Figura 12 - Exemplo de Top-K com valor 6.



Fonte: Hugging Face, 2020.

No exemplo acima (Figura 5), o Top-K foi definido como 6 e se passaram duas iterações da etapa de geração (cada uma demonstrada em um gráfico). Os dois gráficos demonstram apenas a distribuição probabilística de 0 a 1 para a próxima palavra a ser escolhida. Como é possível ver, na segunda iteração o algoritmo já excluiu as palavras “not”, “the”, “small” e “told” considerando o contexto agregado de “The car” que foi definido no primeiro passo.

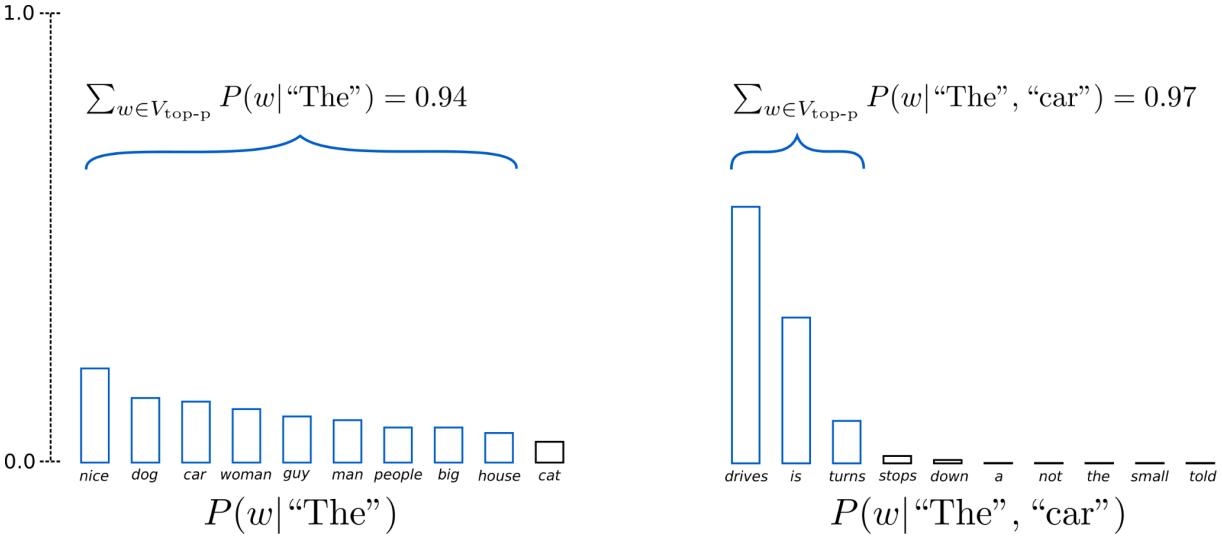
No presente projeto, o valor de “top_k” usado foi 100.

4.2.2.4 Top-P Sampling

Também conhecida como *nucleus sampling*, a amostragem usando o Top-P funciona diferente do Top K. Em vez de usar as K palavras mais prováveis a cada iteração, o P define a probabilidade que um conjunto de possibilidades deve exceder para ser escolhido.

O Top-P escolhe o conjunto com a menor quantidade de palavras possível que com suas probabilidades agregadas some mais que P, distribui a massa de probabilidade entre elas e escolhe uma. Na próxima iteração o algoritmo usa o texto já gerado como entrada para escolher a próxima palavra.

Figura 13 - Exemplo de distribuição usando Top-P com valor 0,92.



Fonte: Hugging Face, 2020.

O exemplo da Figura 6 mostra a escolha das palavras em duas iterações usando o valor de Top-P como 0.92. Os dois gráficos demonstram apenas a distribuição probabilística de 0 a 1 para a próxima palavra a ser escolhida. Na primeira iteração, considerando a entrada “The” o conjunto das nove palavras mais prováveis juntas soma 0.94, o que excede o P definido. Já na segunda iteração, apenas as três palavras mais prováveis já somam uma probabilidade de 0.97, então o algoritmo só precisa escolher entre essas três.

Por ser uma probabilidade, a implementação precisa levar em conta que 0 < top_p < 1. Nesse projeto o valor de Top-P usado foi 0.9.

4.2.2.5 Min Length

A propriedade min_length indica a menor quantidade de palavras que pode ser retornada em uma resposta. O algoritmo sempre vai continuar iterando até, no mínimo, completar esse número antes de devolver um resultado.

O min_length usado para esse projeto é 3, visando que em uma conversa o mais perto do coloquial podem ter sim falas de apenas três palavras.

4.2.2.6 Max Length

O argumento max_length decide qual é o maior comprimento em palavras possível de uma resposta para o usuário. A IA sempre vai tentar concluir o pensamento antes de chegar nesse comprimento.

Para esse projeto, o max_length foi passado como 200, o que é um valor que permite a explicação de um conceito longo ou uma resposta mais complexa no caso de um diálogo que a exija. Sendo assim, o algoritmo implementado aqui sempre tentará buscar uma resposta que contenha entre 3 e 200 palavras para atingir o resultado e retorná-lo ao usuário.

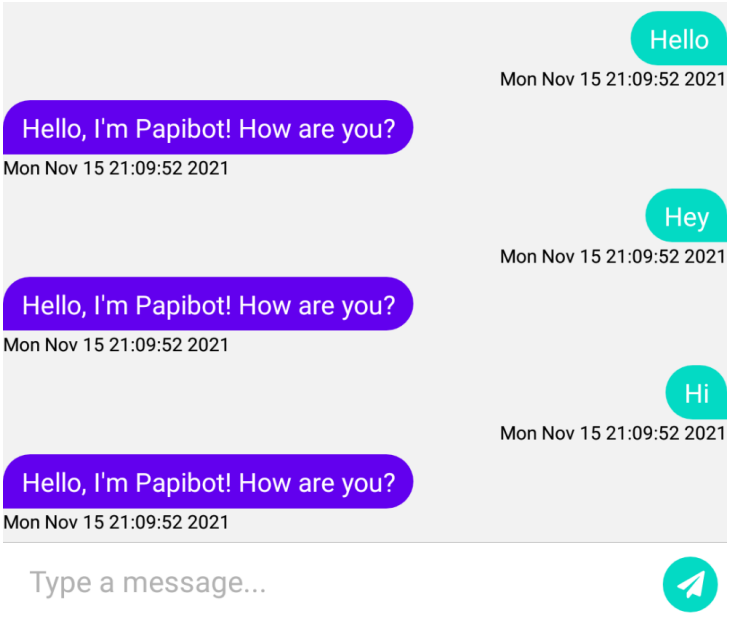
5 RESULTADOS E DISCUSSÕES

Após treinados os algoritmos, os testes foram feitos interagindo com a máquina. Houveram dois tipos de interação, uma de diálogo no estilo “papo-furado” e a segunda usando dados específicos do *dataset*. Para cada entrada do usuário, cada IA deveria responder tentando manter o diálogo continuando. A seguir, são demonstradas as comparações entre as respostas.

5.1 DEEPPAVLOV

Testando o DeepPavlov tentando iniciar uma conversa com apenas frases comuns, sempre traz uma resposta determinística, como no exemplo abaixo. É possível ver que com frases que não foram usadas no treino, a IA sempre vai responder dizendo que ou não compreendeu a pergunta ou não possui uma resposta. Ao saudar, o algoritmo sempre retorna a mesma resposta, como na Figura 14.

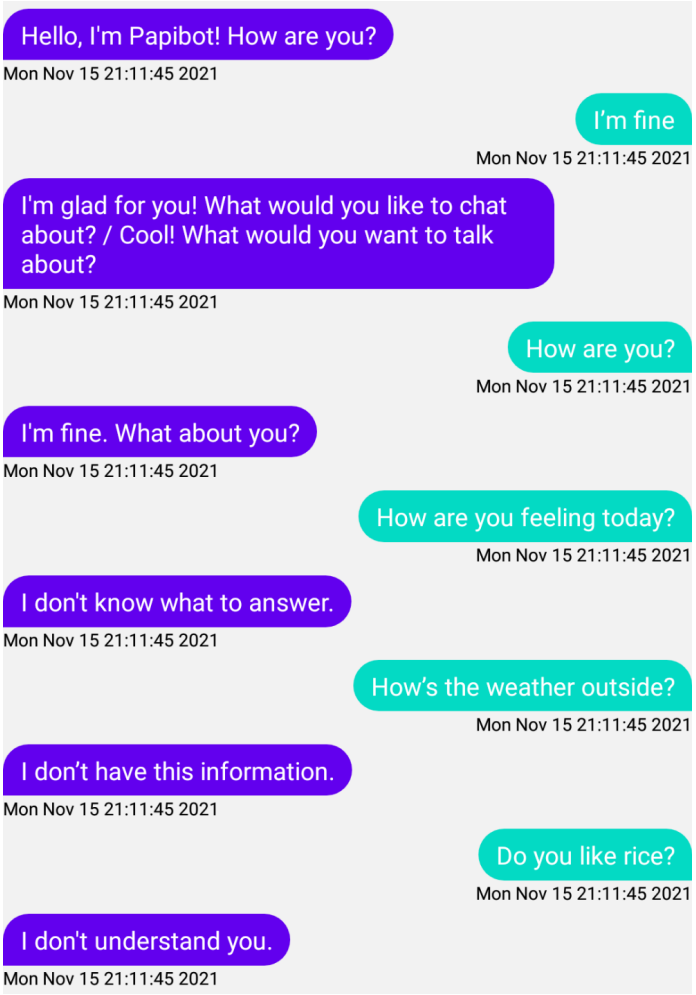
Figura 14 - Saudação com DeepPavlov.



Fonte: elaborado pelo autor.

A Figura 15 mostra um início de diálogo com a implementação do DeepPavlov e os textos para quando a IA não tem uma resposta definitiva.

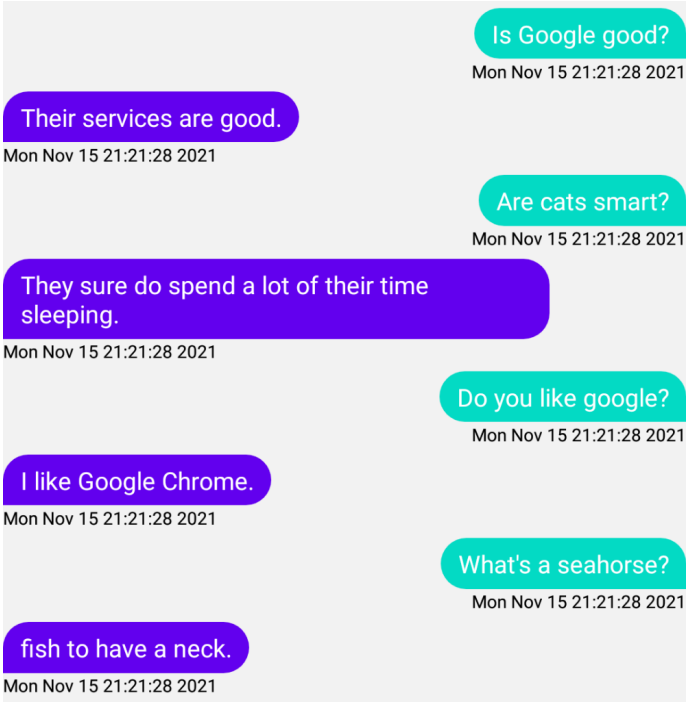
Figura 15 - Conversa simples com DeepPavlov.



Fonte: elaborado pelo autor.

Na segunda tentativa (Figura 16), agora usando falas específicas dos dados usados no treino, a IA com o DeepPavlov se comportou diferente. Agora suas respostas foram mais concisas e mais relacionadas ao contexto da pergunta:

Figura 16 - Perguntas com respostas definidas usando DeepPavlov.



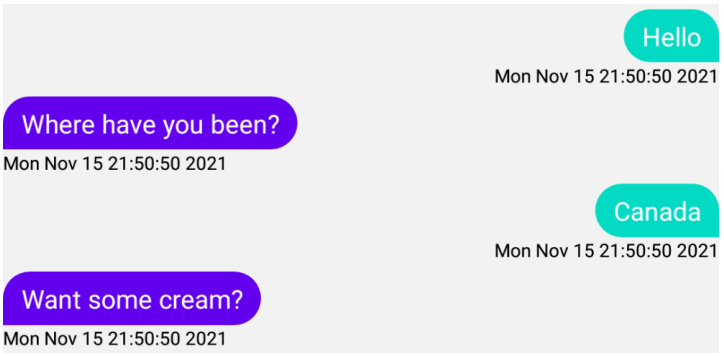
Fonte: elaborado pelo autor.

É observável aqui que a IA tenta formular orações que respondam a pergunta objetivamente a partir de sentenças de dentro do texto que usa como referência. Na última resposta, por exemplo, lê-se “fish to have a neck”, mas o contexto de onde foi tirada essa informação dizia: “Did you know that a seahorse is the only fish to have a neck?”. Aqui o algoritmo não reinterpretou a frase do texto para trazer uma resposta, mas apenas copiou o predicado que vinha depois do verbo da oração (“is”). Em vez disso, uma melhor resposta seria “a fish that has a neck” ou “the only fish that has a neck”.

5.2 DIALOGPT

Assim como com o DeepPavlov, o DialoGPT foi testado primeiramente com um diálogo de papo-furado, com saudações e começos de conversa. As Figuras 17 e 18 mostram diferentes formas da IA responder a um mesmo estímulo.

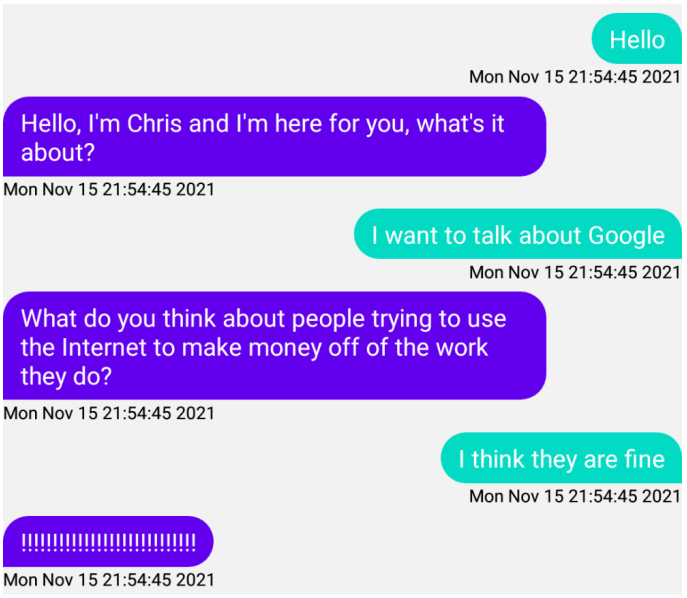
Figura 17 - Primeira saudação com o DialoGPT.



Fonte: elaborado pelo autor.

Algumas vezes o algoritmo não vai saber responder, então a saída dele vai ser menos inteligível para o usuário, como demonstrado na Figura 18.

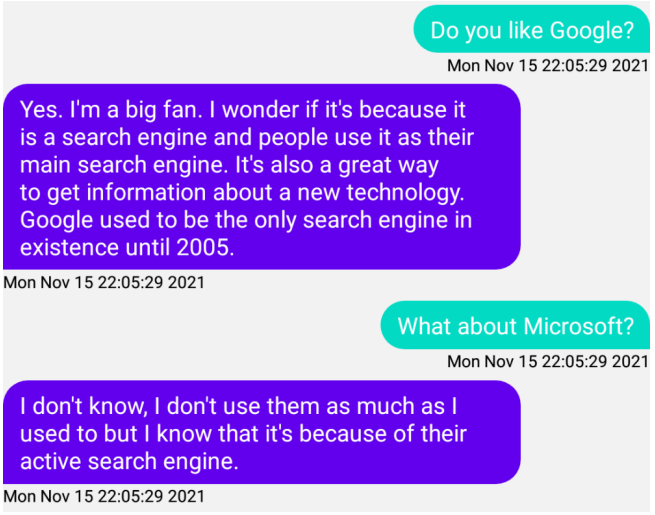
Figura 18 - Segunda saudação com o DialoGPT.



Fonte: elaborado pelo autor.

O segundo passo foi testar o algoritmo com perguntas diretas sobre os textos que constavam no dataset de treino. A próxima conversa foi dividida nas figuras 19 e 20. Na primeira parte (Figura 12), temos uma pergunta sobre uma parte específica do texto, que a IA responde com perfeição dando uma opinião sobre o assunto. O que leva à segunda pergunta.

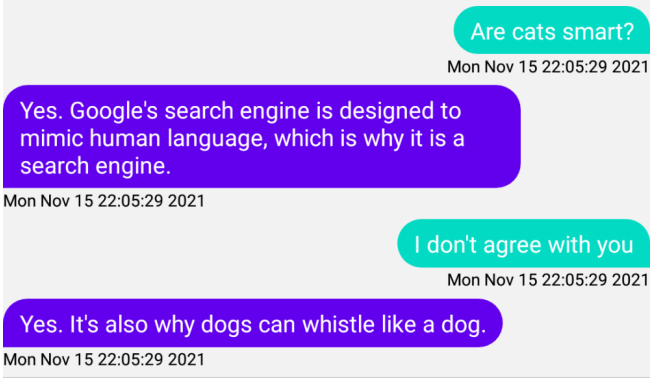
Figura 19 - Diálogo com perguntas específicas sobre tecnologia para o DialoGPT.



Fonte: elaborado pelo autor.

Após responder de forma inteligível e contextualizada, o algoritmo guardou esse histórico do diálogo para usar em respostas futuras. Por isso, na continuação da mesma conversa (Figura 13), quando é questionada sobre algo fora do contexto atual - antes era sobre Google/Microsoft e tecnologia, agora sobre gatos -, a IA se perde na resposta e tem dificuldade de mudar de assunto.

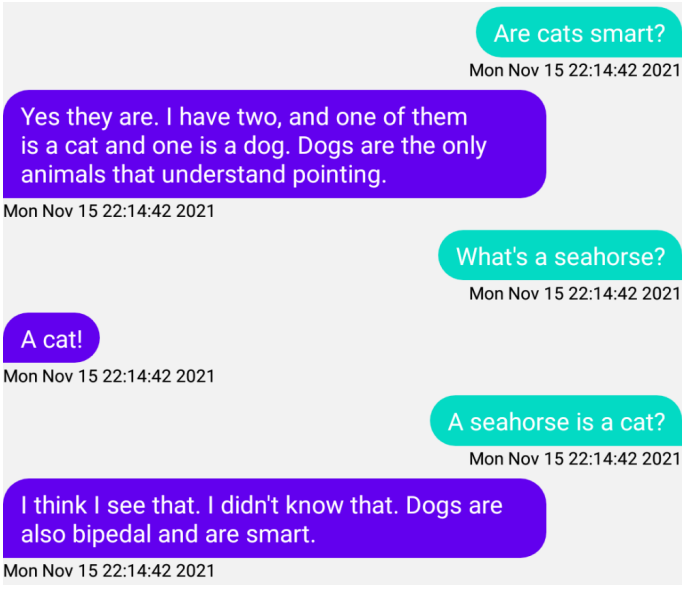
Figura 20 - Continuação do diálogo com perguntas sobre tecnologia.



Fonte: elaborado pelo autor.

Na última interação, foi iniciado o diálogo já perguntando sobre gatos. O que aconteceu foi que, dessa vez, a IA tomou outro rumo em suas respostas. A Figura 14 mostra como se desenvolveu a conversa.

Figura 21 - Refazendo a pergunta ao DialoGPT que quebrou o contexto no diálogo anterior.



Fonte: elaborado pelo autor.

Na Figura 14, o algoritmo mostrou que também poderia conversar sobre gatos, mas dessa vez, ele ficou preso no assunto cachorros. Isso acontece porque, no processo de treino, os dados são usados em ordem aleatória, o que pode significar que, mesmo com a mesma quantidade de conversas sobre dois assuntos diferentes, um deles acabará tendo mais relevância para a máquina.

5.3 TABELA DE COMPARAÇÃO

A F1-Score do DeepPavlov (88.9) e do DialoGPT (83.0) já foi documentada em seus respectivos sites. Enquanto o DeepPavlov consegue ter uma compreensão mais exata do que foi inserido pelo usuário, o DialoGPT tenta interpretar depois de compreender, gerando alguns resultados menos exatos na hora de calcular a acurácia.

O fator geração segue um princípio parecido, o DeepPavlov tem uma capacidade maior de gerar uma resposta com as palavras exatas que foram retiradas do contexto em que apareciam. Já o DialoGPT passa por um processo recursivo de criação para gerar sempre uma nova resposta para a mesma entrada. Por isso, em geral, o contexto e a maior parte das palavras (70%) são geradas pelo algoritmo DialoGPT, porém, pela falta de exatidão, a implementação com DeepPavlov traz respostas com 88% de precisão.

Apesar de trabalhar com um algoritmo recursivo - que normalmente demora mais para

ser processado -, o DialoGPT teve uma performance melhor nos testes. Isso se dá ao fato de que, após treinado e carregado à memória, esse modelo não precisa mais consultar nenhuma fonte externa, todos os dados da geração estão contidos dentro de si. O que diferencia o DeepPavlov e acaba deixando-o mais lento, é o fato de trabalhar com uma base de dados em um arquivo externo ou em um banco de dados. Então toda vez que o modelo com DeepPavlov precisa encontrar uma resposta que não está carregada na memória, ele vai consultar a base de dados. A diferença de performance entre os dois algoritmos variou, porém ficou, em maior parte, entre 200 e 300 milissegundos.

A implementação usando DeepPavlov tem uma característica diferente do que normalmente se tem quando se trabalha com *machine learning*: ou ela faz o uso de *presets* próprios aplicados no arquivo de configuração, ou é necessário desenvolver todo o código necessário para aplicá-lo à *pipeline*. O que nasceu com o objetivo de facilitar o desenvolvimento (os arquivos de configuração em JSON) se torna uma complexidade adicional quando o objetivo é personalizar o modelo a ser treinado, afinal, além de escrever o código do modelo/*skill* ainda é necessário aplicá-lo em uma *pipeline* com modelos e *skills* padrões da ferramenta. Outro fator é o treinamento, para treinar uma IA com o DeepPavlov, é necessário criar uma pipeline de treino ou usar uma das pipelines padrões. O problema com a segunda opção é que esses padrões não vêm prontos para aceitar um *dataset* personalizado. Por esses motivos, o DeepPavlov recebeu a nota 3 na facilidade de implementação.

O DialoGPT tem uma alta complexidade na forma em que é organizado seu *dataset* de treino. Como ele não funciona com perguntas e respostas, mas com geração de texto contextualizado, no começo é uma dificuldade entender como implementá-lo, porém essa dificuldade desaparece após trabalhar com a IA por alguns dias. Por seguir o padrão treino/teste da maior parte das bibliotecas e ferramentas de *machine learning*, após entender seus conceitos, o DialoGPT passa a se tornar simples de implementar. A facilidade de implementação estaria mais perto de 5 de o ajuste fino (*top_p*, *top_k*, *temperature*) não fosse algo tão necessário para conseguir um resultado. Dessa forma, a nota recebida pelo DialoGPT nesse quesito foi 4.

O fator decisivo para o projeto então caiu sobre a flexibilidade do algoritmo. Aqui temos uma discrepância que mostra o principal foco de cada implementação. Enquanto o DeepPavlov está mais preocupado em responder à entrada do usuário com perfeição, o DialoGPT foca na diversidade do diálogo, em manter o engajamento e no contexto do histórico da conversa. Cada resposta do DialoGPT é única para cada diálogo, o que o torna uma ferramenta flexível, engajadora e mais parecida com um ser humano, trazendo uma

conversa mais verossímil com a de dois amigos conversando.

Abaixo (Quadro 1), é mostrada a tabela para comparação dos resultados objetivos de cada fator sobre cada implementação:

Quadro 1 - Tabela de comparação de fatores objetivos entre os algoritmos.

Fator	DeepPavlov	DialoGPT
Compreensão	88.9	83.0
Geração	88 %	70 %
Performance	500 ms ~ 800 ms	300 ~ 600 ms
Facilidade de Implementação	3	4
Flexibilidade	2	5

Fonte: elaborado pelo autor.

CONSIDERAÇÕES FINAIS

O objetivo desse projeto foi implementar uma IA que fosse capaz de conversar com o usuário e instigá-lo a manter o diálogo continuando. Para isso, foi necessário que os dois algoritmos mais apropriados encontrados fossem treinados e testados usando cenários de conversas o mais próximo do real. A partir da rotina de testes foi possível observar as duas principais diferenças entre as duas implementações: a **flexibilidade** e a **tolerância a falhas**.

A DeepPavlov traz uma forma quase perfeita de encontrar a resposta exata para a entrada do usuário. Mas por sempre procurar a resposta perfeita em um contexto reconhecido, suas opções são limitadas. Todas as possibilidades precisam ser consideradas antes mesmo da primeira interação com o usuário, e o algoritmo tem dificuldade de reconhecer a mesma pergunta em contextos diferentes considerando o histórico da conversa. Isso a faz perder muito no quesito flexibilidade quando o foco não é o final da conversa, mas o oposto: manter a conversa continuando.

É discernível que a implementação usando o DialoGPT sempre pode surpreender e trazer respostas contextualizadas com a pergunta, que incitam a continuação do diálogo e que não existiam nos dados de treino, tudo isso através da sua geração de linguagem natural. Porém essa capacidade de gerar respostas novas a partir de contextos e perguntas novas também pode se tornar um problema quando o contexto não é encontrado. Algumas vezes não haverá resposta textual alguma exceto por uma pontuação incompreensível pelo usuário (como, por exemplo “!!!!!!!!!!”).

A partir dos resultados obtidos, a biblioteca que será oficialmente usada na implementação da inteligência artificial na API do aplicativo Papibot é a DialoGPT. Pela sua flexibilidade e alto engajamento em conversas de domínio aberto, o algoritmo se faz a melhor escolha em um aplicativo de bate-papo com a máquina em que o objetivo é manter a conversa fluindo.

Em casos em que a IA com DialoGPT não pode trazer uma resposta inteligível para o usuário, será trabalho do algoritmo que estiver fazendo o uso do serviço filtrar. Uma forma de fazer isso é dentro do próprio aplicativo colocar uma condição que verifica se existem palavras na resposta que vem da API e, caso não houver, mostrar uma mensagem padrão para o usuário: “Não tenho uma resposta para isso”. Outra forma seria a própria API que implementa a IA refazer o pedido da mesma fala do usuário, agora mudando o contexto para facilitar a compreensão do algoritmo.

Através do resultado obtido, o aplicativo Papibot agora poderá usar o serviço de

inteligência artificial para gerar respostas automáticas para seu usuário, simulando um diálogo próximo do humano. O *dataset* utilizado como base contém conversas sobre assuntos gerais, que foi desenvolvido pela Amazon com o propósito de apoiar o desenvolvimento de assistentes virtuais que não entram fundo em assuntos complexos.

O próximo passo para o Papibot é treinar seu algoritmo de ML com diálogos de filmes, séries, livros e áudios com conteúdos mais específicos de áreas e com diferentes personalidades. Procurando em sites como Kaggle, é fácil encontrar *datasets* com falas de episódios de séries e desenhos animados que possam atrair diferentes públicos. Por exemplo, um fã de The Office pode engajar-se em uma conversa com Michael Scott, o personagem principal do seriado americano - interpretado por uma IA, é claro. Outra possibilidade é, após aplicado o trabalho em filtrar os dados e passar em testes rigorosos, disponibilizar o aplicativo para escolas de inglês infantil usando falas de personagens como Peppa Pig ou Bob Esponja.

Na sequência, pretende-se implementar no Papibot uma atividade diferenciada, o que o torna um aplicativo original, permitindo que o aplicativo faça sugestões de correção no texto do seu usuário, como se fosse um professor corrigindo a escrita do seu aluno, e assim aprimorando sua capacidade de conversação quanto aos aspectos gramaticais.

Independente de como o Papibot seguirá a partir deste ponto, é correto afirmar que a implementação do algoritmo de inteligência artificial desenvolvido foi imprescindível para a continuação do projeto. Sem uma IA como base, o aplicativo não pode conversar com o usuário, portanto, essa se define como a parte mais importante - o cérebro - do sistema como um todo.

REFERÊNCIAS

ALURI, Rao. Expert systems for libraries. *Library administration & management*, 1988.

BATES, Madeline. Models of natural language understanding. *National Academy of Sciences*, v. 92, n. 22, p. 9977-9982, 1995.

BURTSEV, Mikhail; KORNEV, Daniel. DeepPavlov Medium, 2020. It All Started With A Dream. Disponível em <https://medium.com/deeppavlov/it-all-started-with-a-dream-44e0861d0642>. Acesso em 04/11/2021.

BAYMURZINA, Dilyara; BURTSEV, Mikhail; DMITRIEVSKY, Alexander; IGNATOV, Fedor; KORNEV, Daniel. DeepPavlov Medium, 2020. The King is Dead — Long Live The King, or welcome to Deepy 3000! Disponível em <https://medium.com/deeppavlov/the-king-is-dead-long-live-the-king-or-welcome-to-deepy-3000-b7a7c25755ca>. Acesso em 04/11/2021.

CHEN, Danqi; FISCH, Adam; WESTON, Jason; BORDES, Antoine. Reading Wikipedia to Answer Open-Domain Questions. *Association for Computational Linguistics (ACL)*, [S. l.], p. 1, 2017. Disponível em: https://aclanthology.org/P17-1171.pdf. Acesso em: 14 nov. 2021.

CHOWDHURY, Gobinda G. Natural language processing. *Annual review of information science and technology*, v. 37, n. 1, p. 51-89, 2003.

DALE, Robert; REITER, Ehud. Computational Interpretations of the Gricean Maxims in the Generation of Referring Expressions. *Cognitive Science*, 19 (2), p. 233–263, 1995.

DALIANIS, Hercules. Clinical Text Mining: Secondary Use of Electronic Patient Records. *Germany: Springer International Publishing*. p. 47, 2018.

DEEPPAVLOV. DeepPavlov.ai, 2021. DeepPavlov: an open source conversational AI framework. Disponível em <https://deeppavlov.ai>. Acesso em: 03/10/2021.

DEEPPAVLOV. DeepPavlov Conceptual Overview. *Neural Networks and Deep Learning Lab*, 2018. Disponível em <http://docs.deeppavlov.ai/en/master/intro/overview.html>. Acesso em 07/12/2021.

GATT, A. Krahmer E. Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research*, p. 9 - 15, 2018.

GEEKSFORGEEKS. GeeksforGeeks. Agents in Artificial Intelligence. Disponível em <https://www.geeksforgeeks.org/agents-artificial-intelligence/>. Acesso em: 07/12/2021.

GROUP, Natural Language Computing. R-NET: Machine Reading Comprehension with Self-matching Networks. Microsoft Research Asia, [s. l.], 1 maio 2017. Disponível em: https://www.microsoft.com/en-us/research/publication/mcr/. Acesso em: 14 nov. 2021.

GOPALAKRISHNAN, Karthik et al. Topical-Chat: Towards Knowledge-Grounded Open-Domain Conversations. Proc. Interspeech 2019, [s. l.], p. 1891-1895, 2019.

KRISGNAVEDALA, CC0. *Wikimedia Commons*. 11 October 2012.

LEFFA, Vilson J. O ensino de línguas estrangeiras no contexto nacional. *Contexturas*, APLIESP, n. 4, p. 13-24, 1999.

NILSON, Neils S. Principles of Artificial Intelligence. *Springer Verlag*, Berlin, 1982.

PLATEN, Patrick von; Hugging Face, 2020. How to generate text: using different decoding methods for language generation with Transformers. Disponível em <https://huggingface.co/blog/how-to-generate>. Acesso em 05/11/2021.

STRELKOVA, O. Three types of artificial intelligence. 2017.

VASWANI, Ashish; SHAZEER, Noam; PARMAR, Niki; USZKOREIT, Jakob; JONES, Llion; GOMEZ, Aidan; KAISER, Łukasz; POLOSUKHIN, Illia. Attention is all you need. *NeurIPS*, [*S. l.*], 2017.

ZHANG, Yizhe; SUN, Siqi; GALLEY, Michel; CHEN, Yen-Chun; BROCKETT, Chris; GAO, Xiang; GAO, Jianfeng; LIU, Jingjing; DOLAN, Bill. DialoGPT: Large-Scale Generative Pre-training for Conversational Response Generation. *ACL, system demonstration*, [*S. l.*], p. 1-10, 2020. Disponível em: https://arxiv.org/pdf/1911.00536.pdf. Acesso em: 26 set. 2021.

APÊNDICE A - Código do main_runner.py que é usado no treino da DialoGPT

```
def main(df_trn, df_val, args=default_args):
    if args.should_continue:
        sorted_checkpoints = _sorted_checkpoints(args)
        if len(sorted_checkpoints) == 0:
            raise ValueError(
                "Used --should_continue but no checkpoint was found in
--output_dir.")
        else:
            args.model_name_or_path = sorted_checkpoints[-1]

    if (
        os.path.exists(args.output_dir)
        and os.listdir(args.output_dir)
        and args.do_train
        and not args.overwrite_output_dir
        and not args.should_continue
    ):
        raise ValueError(
            "Output directory ({}) already exists and is not empty. Use
--overwrite_output_dir to overcome.".format(
                args.output_dir
            )
        )

    # Setup CUDA, GPU & distributed training
    device = torch.device("cuda")
    args.n_gpu = torch.cuda.device_count()
    args.device = device

    # Setup logging
    logging.basicConfig(
        format="%(asctime)s - %(levelname)s - %(name)s -   %(message)s",
        datefmt="%m/%d/%Y %H:%M:%S",
        level=logging.INFO if args.local_rank in [-1, 0] else logging.WARN,
    )
    logger.warning(
        "Process rank: %s, device: %s, n_gpu: %s, distributed training: %s,
16-bits training: %s",
        args.local_rank,
        device,
        args.n_gpu,
        bool(args.local_rank != -1),
        args.fp16,
    )

    # Set seed
    set_seed(args)

    config = AutoConfig.from_pretrained(
        args.config_name, cache_dir=args.cache_dir)
    tokenizer = AutoTokenizer.from_pretrained(
        args.tokenizer_name, cache_dir=args.cache_dir)
    model = AutoModelWithLMHead.from_pretrained(
        args.model_name_or_path,
        from_tf=False,
        config=config,
```

```
        cache_dir=args.cache_dir,
    )
    model.to(args.device)

    logger.info("Training/evaluation parameters %s", args)

    # Training
    if args.do_train:
        train_dataset = load_and_cache_examples(
            args, tokenizer, df_trn, df_val, evaluate=False)

        global_step, tr_loss = train(args, train_dataset, model, tokenizer)
        logger.info(" global_step = %s, average loss = %s",
                    global_step, tr_loss)

    # Saving best-practices: if you use save_pretrained for the model and
tokenizer, you can reload them using from_pretrained()
    if args.do_train:
        # Create output directory if needed
        os.makedirs(args.output_dir, exist_ok=True)

        logger.info("Saving model checkpoint to %s", args.output_dir)
        # Save a trained model, configuration and tokenizer using
`save_pretrained()`.
        # They can then be reloaded using `from_pretrained()`
        model_to_save = (
            model.module if hasattr(model, "module") else model
        )  # Take care of distributed/parallel training
        model_to_save.save_pretrained(args.output_dir)
        tokenizer.save_pretrained(args.output_dir)

        # Good practice: save your training arguments together with the
trained model
        torch.save(args, os.path.join(args.output_dir,
"training_args.bin"))

        # Load a trained model and vocabulary that you have fine-tuned
        model = AutoModelWithLMHead.from_pretrained(args.output_dir)
        tokenizer = AutoTokenizer.from_pretrained(args.output_dir)
        model.to(args.device)

    # Evaluation
    results = {}
    if args.do_eval and args.local_rank in [-1, 0]:
        checkpoints = [args.output_dir]
        if args.eval_all_checkpoints:
            checkpoints = list(
                os.path.dirname(c) for c in
sorted(glob.glob(args.output_dir + "/**/" + WEIGHTS_NAME, recursive=True))
            )
            logging.getLogger("transformers.modeling_utils").setLevel(
                logging.WARN)  # Reduce logging
        logger.info("Evaluate the following checkpoints: %s", checkpoints)
        for checkpoint in checkpoints:
            global_step = checkpoint.split(
                "-")[-1] if len(checkpoints) > 1 else ""
            prefix = checkpoint.split(
                "/")[-1] if checkpoint.find("checkpoint") != -1 else ""

            model = AutoModelWithLMHead.from_pretrained(checkpoint)
```

```
            model.to(args.device)
            result = evaluate(args, model, tokenizer,
                              df_trn, df_val, prefix=prefix)
            result = dict((k + "_{}".format(global_step), v)
                          for k, v in result.items())
            results.update(result)

    return results
```

ANEXO A - Configuração de pipeline “en_odqa_infer_wiki.json”

```
{
  "chainer": {
    "in": ["question_raw"],
    "out": ["answer", "answer_score", "answer_place"],
    "pipe": [
      {
        "config_path":
"{CONFIGS_PATH}/doc_retrieval/en_ranker_tfidf_wiki.json",
        "in": ["question_raw"],
        "out": ["tfidf_doc_ids"]
      },
      {
        "class_name": "wiki_sqlite_vocab",
        "in": ["tfidf_doc_ids"],
        "out": ["tfidf_doc_text"],
        "join_docs": false,
        "shuffle": false,
        "load_path": "{DOWNLOADS_PATH}/odqa/enwiki.db"
      },
      {
        "class_name": "document_chunker",
        "in": ["tfidf_doc_text"],
        "out": ["chunks"],
        "flatten_result": true,
        "paragraphs": true
      },
      {
        "class_name": "string_multiplier",
        "in": ["question_raw", "chunks"],
        "out":["questions"]
      },
      {
        "class_name": "logit_ranker",
        "batch_size": 64,
        "squad_model": {"config_path":
"{CONFIGS_PATH}/squad/multi_squad_noans_infer.json"},
        "sort_noans": true,
        "in": ["chunks", "questions"],
        "out": ["answer", "answer_score", "answer_place"]
      }
    ]
  },
  "metadata": {
    "variables": {
      "ROOT_PATH": "~/.deeppavlov",
      "DOWNLOADS_PATH": "{ROOT_PATH}/downloads",
      "MODELS_PATH": "{ROOT_PATH}/models",
      "CONFIGS_PATH": "{DEEPPAVLOV_PATH}/configs"
    },
    "download": [
    ]
  }
}
```