

**FUNDAÇÃO GETÚLIO VARGAS
ESCOLA DE MATEMÁTICA APLICADA
CURSO DE PÓS-GRADUAÇÃO EM INTELIGÊNCIA ARTIFICIAL**

**BRUNO DE ALMEIDA GOMES
LAIO EVERTON LIMA MAGALHÃES**

**DESENVOLVIMENTO E TREINAMENTO DE CHATBOT PARA OTIMIZAÇÃO DE
DOCUMENTOS OPERATIVOS NAS SEÇÕES DE ESTADO-MAIOR DOS
BATALHÕES**

TRABALHO DE CONCLUSÃO DE CURSO

RIO DE JANEIRO

2024

**BRUNO DE ALMEIDA GOMES
LAIO EVERTON LIMA MAGALHÃES**

**DESENVOLVIMENTO E TREINAMENTO DE CHATBOT PARA OTIMIZAÇÃO DE
DOCUMENTOS OPERATIVOS NAS SEÇÕES DE ESTADO-MAIOR DOS
BATALHÕES**

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título Pós-Graduação em Inteligência Artificial, da Escola de Matemática Aplicada, da Fundação Getúlio Vargas.

Orientador: Prof. Dr. Rafael de Pinho Andrade

RIO DE JANEIRO

2024

RESUMO

Este trabalho tem como objetivo o desenvolvimento de uma ferramenta Chatbot capaz de otimizar a produção de documentos gerados pelas Seções de Estado-Maior de um Batalhão de Infantaria, sendo implementado em linguagem de programação Python integrada à plataforma ChatGPT da empresa OpenAI. Para o treinamento do Chatbot, visando a precisão nas respostas às questões dos futuros usuários, foram utilizados os manuais doutrinários ostensivos do Comando-Geral do Corpo de Fuzileiros Navais (CGCFN). Essa abordagem permite ao Chatbot elaborar possíveis respostas às Ordens de Operação advindas do Escalão Superior, oferecendo ao planejador uma análise mais rápida do problema a ser enfrentado e acelerando o Ciclo de Boyd das Seções do Batalhão.

Palavras-chave: *Natural Language Processing*; Inteligência Artificial Aplicada ao Ambiente Militar; *Large Language Models*; Documentos Operativos; Implicações Éticas Atinentes ao uso de IA no Militarismo. Desenvolvimento de Chatbots.

ABSTRACT

This work aims to develop a Chatbot tool capable of optimizing the production of operational documents generated by the General Staff Sections of an Infantry Battalion, using the Python programming language integrated into the OpenAI company's ChatGPT platform. In the process of training the Chatbot, so that it can respond accurately to future users' questions, the ostensible doctrinal manuals of the General Command of the Brazilian Marine Corps (CGCFN) were implemented, enabling the Chatbot to develop possible solutions to Operation Orders coming from Higher Echelon and giving the planner the opportunity for a faster analysis of the problem to be faced, accelerating the Boyd Cycle of the Battalion Sections.

Keywords: *Natural Language Processing. Artificial Intelligence Applied to Military Environment. Large Language Models. Operational Documents. Ethical Implications Concerning AI uses in the Militarism. Chatbot Development.*

LISTA DE QUADROS

Quadro 1 – Importação de Bibliotecas para o Projeto	35
Quadro 2 – Inicialização do Módulo <i>Flask</i>	35
Quadro 3 – Criação de Ambiente Seguro para a <i>API-Key</i>	35
Quadro 4 – Carregamento do Arquivo JSON	35
Quadro 5 – Busca o Arquivo HTML no Diretório	35
Quadro 6 – Busca o Arquivo <i>Static</i> no Diretório.	36
Quadro 7 – Função para Divisão de Textos em Chunks.	36
Quadro 8 – Função para a Criação de Embeddings (Vetorização).	36
Quadro 9 – Função de Busca Direta por Abreviações Utilizando RegEx e Similarity.	37
Quadro 10 – Integração de Respostas do Chat.	37
Quadro 11 – Comando de Execução Principal e Módulo Live Reload Aplicado.	40
Quadro 12 – Código Produzido com a IA do Google.	41
Quadro 13 – Código Utilizado para a Criação de Ambiente HTML.	46
Quadro 14 – Código Utilizado na Confeção da Caixa de Mensagens do Chatbot. .	48
Quadro 15 – Código Utilizado para Formatação da Caixa de Mensagens.....	51
Quadro 16 – Criação de Arquivo Dockerfile.	57
Quadro 17 – Exemplo de Arquivo Requirements.txt.	58
Quadro 18 – Criação de Arquivo .env.	58
Quadro 19 – Alterações no Código Flask-App.	58
Quadro 20 – Alterações no Arquivo Dockerfile.....	59
Quadro 21 – Alteração no Arquivo .env.....	59

LISTA DE TABELAS

Tabela 1 - Perguntas Elaboradas de Acordo com os Manuais.....	61
Tabela 2 - Respostas Geradas pelo Chatbot	67

LISTA DE ILUSTRAÇÕES

Figura 1 – Acesso à Plataforma da OpenAI	68
Figura 2 - Acesso à Plataforma Google Cloud	68
Figura 3 – Criação da API-Key da OpenAI	69
Figura 4 – Acesso ao Google Cloud (APIs & Services).....	69
Figura 5 – Docker Build Criada em Docker Desktop	70
Figura 6 – Demonstração do Chatbot	70
Figura 7 – Frontend Criado na Plataforma Wix	71
Figura 8 – Integração do URL Criado no Google Cloud com Wix Website	71

SUMÁRIO

RESUMO	3
ABSTRACT	4
LISTA DE ILUSTRAÇÕES	7
1 INTRODUÇÃO	10
1.1 OBJETIVOS.....	11
1.1.1 Objetivo Geral.....	11
1.1.2 Objetivos Específicos.....	11
2 REVISÃO DA LITERATURA	12
2.1 CHATBOTS E PROCESSAMENTO DE LINGUAGEM NATURAL (PLN)	12
2.2 MILITARIZAÇÃO E ARMAMENTIZAÇÃO DA IA	16
2.3 CONSIDERAÇÕES ÉTICAS E DESAFIOS	18
2.4 LACUNAS E OPORTUNIDADES.....	20
3 MODELAGEM COMPUTACIONAL	20
3.1 CRIANDO UMA <i>API-KEY</i>	21
3.2 DESENVOLVIMENTO DO CÓDIGO	22
3.2.1 <i>Layer Flask</i>	22
3.2.2 Integração OpenAI.....	22
3.2.3 Camada de Dados	23
3.2.4 <i>APIs e Routes</i>	23
3.2.5 Processamento de Texto e Lógica de Busca.....	23
3.2.6 Configuração do Servidor e do Ambiente	24
3.3 TESTES DO CÓDIGO	24
3.4 <i>DEPLOYMENT</i> DO APLICATIVO.....	26
3.5 INTEGRAÇÃO DO APP EM WEBPAGE	27
3.6 AVALIAÇÃO DO APLICATIVO	28
3.6.1 Análise Comparativa entre Respostas.....	28
4 CONCLUSÃO	28
REFERÊNCIAS	30
APÊNDICES	34
APÊNDICE A – Módulo OpenAI.....	35
APÊNDICE B – Código com o Módulo Gemini.....	41
APÊNDICE C – Código em Linguagem HTML.....	46
APÊNDICE D – Código em Linguagem JavaScript.....	48
APÊNDICE E – Código em Linguagem CSS	51
APÊNDICE F – Arquivos para Criação de Imagem.....	57
APÊNDICE H – Perguntas e Respostas	60
ANEXO A – Figuras dos Ambientes Computacionais	68

1 INTRODUÇÃO

Com a progressão exponencial da criação de novas tecnologias, tanto no mundo civil quanto no militar, vê-se a necessidade de utilização de ferramentas que maximizam por exemplo a produção de carros, medicamentos, a plantação de alimentos, entre outras variedades de itens que são gerados com o intuito de melhorar a qualidade de vida da população de uma cidade, estado, ou do país e do mundo. Essa maximização deve ser feita não só para criar quantidades exacerbadas de *outputs*, mas sim, produzir de maneira eficiente, eficaz e efetiva aquilo que é necessário para atender o bem-estar social das pessoas e obter um desenvolvimento sustentável do ambiente a ser analisado como um todo. Isso pode ser denominado como processo de otimização, em que as variáveis como custo e tempo são constantemente analisadas para se obter o menor gasto no menor tempo possível, independente se a avaliação está sendo feita no setor privado ou público.

Ao longo do tempo, pôde-se observar que tanto o setor público quanto o privado puderam desenvolver excelentes técnicas de gerenciamento e administração, e que ambos tiraram proveito delas, adaptando as técnicas elaboradas em um setor para a realidade do outro. No caso das Forças Armadas (FFAA), apesar de serem órgãos bem específicos do setor público, não é diferente, pois sua administração trabalha, na maioria dos casos, de forma idêntica a qualquer outro órgão ou entidade do governo ou Pessoa Jurídica de Direito Privado, cujo objetivo é de obter respostas rápidas às demandas impostas pelos Superiores Hierárquicos advindas da sociedade. Então, fez-se necessária a busca da integração, adaptação e renovação constante de modelos de planejamento e gerência militares aos modelos civis como meios elétricos, eletrônicos, e de rede dados que são capazes de prover velocidade às tomadas de decisões em qualquer nível hierárquico da corporação, e uma ferramenta que está sendo amplamente utilizada para isso são os sistemas implementados com software de Inteligência Artificial.

A Inteligência Artificial ou *Artificial Intelligence (AI)* já está bastante difundida em áreas multivariadas como a medicina, engenharia, biologia e inclusive no militarismo, aplicada em veículos e armas autônomas e nos processos de planejamento militar das Seções de Inteligência. Como a gama de emprego e métodos de *Machine Learning* para facilitar a visualização e resolução de um problema, o

planejamento e a tomada de decisão são absurdamente grandes, para reduzir o foco, este trabalho foi limitado à confecção de um *Chatbot*.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

É de natureza fundamental no militarismo o aprendizado de ferramentas computacionais que ajudem o planejador a assessorar o Superior Hierárquico em sua tomada de decisão. O desenvolvimento de modelos computacionais de *A.I* podem auxiliar na logística, inteligência, operações e recursos humanos identificando com êxito “*bottlenecks*” que impedem ou dificultam os bons resultados de um trabalho, aumentando gastos com pessoal, equipamentos, combustível e o com o ativo mais valioso que existe na terra que é o tempo. Logo, o objetivo deste trabalho é o de um software capaz de receber *uploads* de documentos ou perguntas relativas aos manuais CGCFN Ostensivos e transmitir respostas que possam auxiliar a produção de com documentos e tirar dúvidas sobre a doutrina do Corpo de Fuzileiros.

1.1.2 Objetivos Específicos

1. Desenvolver um modelo de *Chatbot*.
2. Utilizar de linguagem código Python e da ferramenta OpenAI para a confecção do software.
3. Implementar os documentos e manuais da Marinha do Brasil, especificamente as CGCFN Ostensivas, como treino.
4. Avaliar se o *bot* é capaz de responder às diversas perguntas que podem surgir durante no ambiente operacional.

2 REVISÃO DA LITERATURA

Atualmente, um número gigantesco de aplicações da Inteligência Artificial (I.A) em diversos campos da ciência, dos negócios, do direito entre outras, vem imergindo o mundo de forma acelerada em conhecimentos que antes eram de difícil acesso e reprodução. Isso foi devido ao avanço e desenvolvimento de novos algoritmos (*Machine Learning*) capazes de identificar, classificar, e quantificar *inputs* postos a uma máquina, melhorando a sua capacidade de entender a informação dada, transformando-as em linguagem de máquina, processando e transmitindo – *output* –, tal informação de forma clara, concisa, objetiva e com a rapidez necessária ao usuário que não saiba absolutamente nada de linguagem de programação. No campo militar não é diferente, há um universo de pesquisas em I.A sendo realizadas inundando as instituições de tecnologia do setor bélico, porém, com disponibilidade bem mais restritas à população.

Segundo Adib Bin Rashid (2023, p. 2), o estudo da I.A pode ser dividido em sete áreas, que podem ser chamadas de *Seven Patterns of Artificial Intelligence* citando-as: *Hyper-Personalization, Recognition, Predictive and Analytics Decision, Goal-Driven Systems, Autonomous Systems, Patterns and Anomalies, e Conversation and Human Interaction*. Destes sete conjuntos que formam praticamente toda a base da Inteligência Artificial, este trabalho aprofundará apenas em *Conversation and Human Interaction* o suficiente para elaborar um Chatbot capaz de subsidiar respostas para a confeccionar uma Ordem de Operação Militar nível Batalhão de Infantaria, levando em consideração o treinamento do *bot* com os manuais da Marinha do Brasil.

2.1 CHATBOTS E PROCESSAMENTO DE LINGUAGEM NATURAL (PLN)

Um dos primeiros Chatbots a ser criado chamava-se ELIZA, elaborado por volta de 1964–1966, no MIT *Artificial Intelligence Laboratory*, por Joseph Weizenbaum, cientista alemão que fugiu do seu país no período da segunda-guerra mundial e se estabeleceu nos Estados Unidos com sua família. ELIZA foi desenvolvida para simular um psicoterapeuta Rogeriano, capaz de elaborar respostas convincentes aos usuários, que pensavam estar interagindo com outra pessoa. Seguindo a linha do tempo, A.L.I.C.E bot (*Artificial Linguistic Internet Entity*), construído pelo Dr. Richard Wallace em 1995, foi um Chatbot de *Natural Language Processing* (NLP) que

empregava *Artificial Intelligence Markup Language* (AIML) para simular diálogos. Em 1998, A.L.I.C.E sofreu uma pequena modificação, sendo escrita em linguagem Java para obter maior acurácia nas respostas (Adib Bin Rashid, 2023, p. 6). Com o passar dos anos, os Chatbots adquiriram capacidades inimagináveis de respostas, sendo os mais populares: o Siri criado pela Apple Inc. em 2011; Alexa, criado pela Amazon em 2014; Cortana, criado pela Microsoft; o Google Assistant criado em 2016; o ChatGPT, da OpenAI, em 2020 (possuindo atualizações recentes como o ChatGPT - 4.0 e outros mais avançados); LaMDA, da Google, em 2021; o Anthropic's Claude, da empresa Anthropic, em 2024; e o Mitsuku – tendo o Steve Worswick como seu criador –, que continua em constante *updating*. Os avanços em Inteligência Artificial e NLP continuam desbravando fronteiras antes inalcançáveis pelos Chatbots, fazendo-os mais sofisticados e com interações perceptivelmente humanoides. Esses avanços refletem os esforços contínuos em melhorar a categoria *Conversational AI*, fazendo do *bot* uma ferramenta mais útil, confiável, e de fácil usabilidade para uma variedade extensa de aplicações.

Mas o que viria a ser o Processamento de Linguagem Natural (NLP) exatamente? O NLP é uma vertente da AI que foca na interação linguística homem-máquina, ajuda os computadores a entender a linguagem humana, e permite que tais máquinas sejam capazes de comunicar com o usuário. São, basicamente, algoritmos e modelos matemáticos capazes de fazer com que o computador interprete, entenda e gere linguagem humana com significado. A vertente NLP se ramifica em dois processos: 1) Compreensão da Linguagem Natural (*Natural Language Understanding – NLU*); e 2) Geração de Linguagem Natural (*Natural Language Generation – NLG*). NLU foca em interpretar e extrair significado da linguagem humana, visa à compreensão do texto ou da fala, retirar informações relevantes destes, e entender a intenção e as perguntas dos usuários utilizando técnicas como *Text Parsing*, *Entity Recognition*, *Sentiment Analysis*, e *Intent Detection*. Já o NLG, lida em repassar ao usuário – textos ou falas – uma resposta com características semelhantes a de uma pessoa baseado em dados estruturados ou *inputs* dos sistemas de NLU. Os sistemas de NLG geram textos ou discursos coerentes e relevantes combinando regras linguísticas, *templates*, e às vezes modelos de *Machine Learning*, possibilitando as aplicações em resumos de texto, tradução entre línguas, Chatbot *output*, e geração de conteúdo para artigos jornalísticos ou reportagens (M. MOHAN, 2024, p. 6).

Os modelos NLP seguem uma estrutura hierárquica de comandos – *Pipeline* – que aborda as seguintes características: 1) *Sentence Segmentation* - aqui é onde se inicia o tratamento dos dados (orações/frases), dividindo-os em sentenças menores; 2) *Word Tokenization* - quebra as sentenças criadas anteriormente em palavras ou *tokens* como são chamados; 3) *Stemming* – normaliza as palavras em suas formas bases (*roots*), às vezes sem significado; 4) *Lemmatization* – similar ao *Stemming*, porém a forma básica das palavras contém significado, sendo usado para agrupar formas derivadas das palavras chamadas de *Lemma*; 4) Identificação de *Stop Words* – é a filtragem de conectivos, preposições ou qualquer palavra que não tem muito sentido na frase, e são filtradas antes de qualquer análise estatística; 5) *Dependency Parsing* – identifica como as palavras estão correlacionadas em uma dada sentença; 6) *Parts-Of-a-Speech (POS)* – indica quão bem uma palavra funciona semântica quanto gramaticalmente; 7) *Name Entity Recognition (NER)* – processo de identificação do nome da entidade citada na sentença tais como o nome de uma pessoa, filme, cidades; 8) *Chunking* – usado para coletar as partes individuais da informação e agrupando-as em partes maiores da sentença.

Antes de produzir o Chatbot que atenda bem à sua necessidade com o custo monetário e computacional adequados, é preciso identificar quais são os tipos de Chatbots que existem no mercado. Atualmente, existem três tipos de Chatbots de acordo com a classificação de funcionamento, são eles: 1) *Ruled-Based Chatbots* – seguem regras e padrões pré-definidos para responder aos *inputs* dos usuários; 2) *Machine-Learning-Based Chatbots* – que usam algoritmos de *Machine Learning* para aprender com os próprios dados e melhorar suas habilidades durante o tempo, entendendo o contexto e adaptando às preferências dos usuários; e 3) Modelos híbridos – que usam ambos os algoritmos citados. Arelados ao tipo de Chatbot vêm os seus componentes, sendo: 1) *NLP Engine* que processa os *inputs* dos usuários para entender a intenção, extrair as entidades, gerar as respostas e inclui tarefas como *tokenization*, *POS tagging*, *NER*, e *Sentiment Analysis*; 2) há, também, o componente *Dialog Management*, que gerencia o fluxo da conversa, lida com o contexto, mantém os estados, e determina respostas apropriadas baseadas nos *inputs* dos usuários e do conhecimento do sistema. O componente *Backend Integration* conecta o Chatbot com sistemas externos como base de dados, *Application Programming Interfaces (API)*, ou serviços para buscar informações, exercer ações, completar o que foi pedido pelo usuário. O último componente é a Interface Usuário-Máquina, como o próprio

nome diz, provê uma interface de fácil uso para o usuário, podendo ser do tipo texto-texto, texto-voz e vice-versa, e voz-voz.

O desenvolvimento de Chatbot necessita de diversas técnicas e modelos matemáticos, entre eles temos o *Attention Mechanism*, usado em *Neural Networks* para focar em partes relevantes de sequências submetidas à máquina. Essa técnica foi criada para aumentar a performance do *encoder-decoder* (seq2seq) do modelo RNN (*Recurrent Neural Network*), permitindo que o modelo “preste atenção” a certas partes dos dados e dá a eles mais peso ao fazer previsões. Existem outros tipos de *Attention Mechanism*, porém, vale a pena ressaltar aqui o mecanismo de atenção local e de atenção global. O mecanismo de atenção global considera toda a sequência dada ao calcular os pesos da atenção, isso requer que o computo dos pesos de todos os elementos da sequência, levando à alta complexidade computacional. Apesar disso, a técnica de atenção global permite que o modelo alinhe a sequência dada com a resposta, tornando-a mais flexível. Já a técnica de atenção local restringe o mecanismo de atenção a regiões ou janelas específicas da sequência dada, isso reduz a carga computacional por focar em subconjuntos dos elementos da sequência. A desvantagem em comparação com o outro mecanismo é que força um alinhamento mais rígido entre os *inputs* e *outputs* (M MOHANA, 2024, p. 72-75).

A depender da utilização do Chatbot, um modelo mais simples pode atender às demandas específicas muito bem sem maiores problemas. Entretanto, para aplicações mais complexas, como é o caso deste projeto, há a necessidade de implementação de AI para resolver questões de níveis mais avançados. É aí que entra os modelos de linguagem ampla, ou *Large Language Models (LLM)*. Os LLM são um tipo de AI, treinados em vastas quantidades de dados textuais, usando modelos de transformação (*Transformers Models*) que os permitem reconhecer, traduzir, reconhecer, ou gerar textos ou outros conteúdos. Exemplos desses modelos atualmente são o ChatGPT 3.5 e 4.0 da OpenAI, o Copilot e Gemini da Google. Estes funcionam processando os textos, analisando padrões, e gerando respostas baseadas em informações aprendidas dos dados de treinamento. Os LLM são treinados retirando dados da própria Internet, o que pode ser um grande problema, já que, a não observação dos dados provenientes de tais fontes leva o programa a adicionar conteúdo que porventura poderá ser falso. A técnica de tratamento de dados é de fundamental importância para o desenvolvimento dos modelos LLM para evitar os riscos de conteúdos falsos ou incorretos. Além disso, esses modelos têm outras

desvantagens tais como alucinações – quando produz respostas totalmente diferentes das esperadas –; riscos de integridade da segurança – pode vaziar informações privadas, ser utilizado em *phishing scams* e produzir *spams* –; e dificuldade em gerenciar o tempo e os recursos para manter o modelo *up-to-date*.

2.2 MILITARIZAÇÃO E ARMAMENTIZAÇÃO DA IA

A inteligência artificial tem se tornado uma realidade no mundo contemporâneo, especialmente no âmbito das Forças Armadas, sendo designada de “quarta revolução industrial” por Klaus Schwab (presidente e fundador do Fórum econômico mundial). Os sistemas militares de IA são capazes de processar grandes volumes de dados de maneira mais eficiente do que os sistemas tradicionais. Devido às suas capacidades intrínsecas de computação e tomada de decisão, a IA também aumenta a autonomia dos sistemas de combate, promovendo o autocontrole, autorregulação e auto atuação desses sistemas (RASHID et al., 2023).

O uso da IA em aplicações militares é amplo e está em constante expansão, impulsionado pelo apoio contínuo de agências de pesquisas militares para o desenvolvimento de novas e avançadas tecnologias baseadas em IA. Esse apoio é fundamental para atender à crescente demanda por sistemas impulsionados por IA nas Forças Armadas.

A IA é utilizada, praticamente, em todas as aplicações militares, destacando-se áreas como reconhecimento de alvos, vigilância, segurança cibernética, logística militar e veículos autônomos. As capacidades da IA permitem uma maior eficiência, precisão e capacidade de tomada de decisões. Algumas das principais aplicações no campo militar são: 1) Reconhecimento de Alvos e Armas Autônomas – a IA permite o desenvolvimento de sistemas de armas autônomas (AWS), que podem operar sem intervenção humana. Exemplos incluem o drone *MQ-9 Reaper* e o sistema de mísseis *Patriot*, que utilizam sensores avançados para detectar e atacar alvos com precisão. No entanto, esses sistemas levantam questões éticas e legais significativas, especialmente em relação à responsabilidade e ao potencial de uso indevido. Os sistemas autônomos como o *Patriot* e o *Phalanx* são considerados altamente eficientes, mas ainda necessitam de melhorias contínuas para garantir a precisão e minimizar riscos de falhas (RASHID et al., 2023); 2) Vigilância e Monitoramento – os sistemas de IA são amplamente utilizados para vigilância e monitoramento, tanto em

operações de combate quanto em segurança interna. Tecnologias avançadas, como drones autônomos e sistemas de reconhecimento facial, permitem a coleta e análise de dados em tempo real, melhorando a tomada de decisões em campo. O estudo de Rashid et al. (2023) menciona o uso de veículos subaquáticos autônomos (AUVs) equipados com métodos de navegação baseados em redes neurais para detectar minas submarinas, demonstrando a aplicação crítica da IA na vigilância militar; 3) Segurança Cibernética – desempenha um papel crucial na proteção contra ataques cibernéticos, utilizando redes neurais para detectar padrões e anomalias que indicam atividades maliciosas. Esses sistemas são essenciais para defender infraestruturas críticas e garantir a segurança das operações militares. Segundo Rashid et al. (2023), as redes neurais são especialmente eficazes na rápida identificação de novos padrões e na defesa de contra-ataques cibernéticos sofisticados, tornando-as uma ferramenta valiosa para a cibersegurança militar; 4) Função Logística e Transporte – na logística militar, melhora a eficiência na gestão de suprimentos e transporte. Sistemas autônomos de veículos e drones facilitam a entrega de recursos em áreas de difícil acesso, otimizando a cadeia de suprimentos. Rashid et al. (2023) destaca que a IA pode ser utilizada para prever a necessidade de manutenção de equipamentos militares, garantindo que os veículos e outros recursos estejam sempre operacionais; 5) Treinamento e Simulação de Combate – usada para criar simulações de combate realistas, proporcionando um ambiente seguro para o treinamento de soldados. Essas simulações ajudam a preparar as tropas para situações complexas e dinâmicas, aumentando a eficácia operacional. Os sistemas de treinamento baseados em IA podem ser personalizados para atender às necessidades individuais dos soldados, melhorando a eficácia do treinamento (RASHID et al., 2023).

Apesar dos benefícios, o uso da IA no setor militar apresenta vários desafios e riscos potenciais. A possibilidade de falhas nos sistemas autônomos, a vulnerabilidade a ataques cibernéticos e as implicações éticas do uso de armas autônomas são preocupações significativas. A regulamentação e o desenvolvimento de diretrizes para o uso responsável da IA são essenciais para mitigar esses riscos. Rashid et al. (2023) enfatiza a importância de abordar os desafios éticos e legais associados ao uso da IA no campo militar, particularmente em relação às armas autônomas e o potencial para danos não intencionais.

2.3 CONSIDERAÇÕES ÉTICAS E DESAFIOS

A utilização de inteligência artificial (IA) em contextos militares tem sido objeto de intenso debate, especialmente no que tange às implicações éticas e aos desafios operacionais que essas tecnologias emergentes apresentam. No livro *Military Applications of Artificial Intelligence: Ethical Concerns in an Uncertain World*, Morgan et al. (2020) explora profundamente essas questões, oferecendo uma análise abrangente sobre as possíveis repercussões do emprego de IA em operações militares.

Um dos principais pontos de preocupação ética discutidos pelo autor é a autonomia dos sistemas de IA no campo de batalha. A possibilidade de armas autônomas tomarem decisões de vida ou morte sem a intervenção humana levanta sérias questões sobre responsabilidade moral e legal. Como destacado por Morgan et al. (2020), há um receio significativo de que essas "máquinas assassinas" possam agir de maneira imprevisíveis, resultando em danos colaterais ou em ataques a alvos errados devido a falhas no reconhecimento ou na interpretação de dados.

Além disso, o autor argumenta que a integração de sistemas de reconhecimento facial e outras tecnologias de vigilância aprimoradas por IA em contextos militares pode violar direitos humanos fundamentais. Existe o risco de cidadãos inocentes serem erroneamente identificados como ameaças, o que pode levar a detenções ou até mesmo a execuções injustificadas. Morgan et al. (2020) enfatiza que tais possibilidades não apenas comprometem a dignidade humana, mas também podem minar a confiança pública nas instituições militares e governamentais que utilizam essas tecnologias.

Outro desafio destacado no livro é a questão da regulamentação e do controle dessas tecnologias. Morgan et al. (2020) ressalta que, apesar dos Estados Unidos tentarem impor diretrizes rigorosas para o desenvolvimento e uso de IA militar, não há garantia de que adversários, como China e Rússia, sigam padrões éticos semelhantes. Essa disparidade pode colocar os EUA em uma posição desvantajosa, caso esses países adotem uma abordagem mais permissiva e agressiva no uso de IA em conflitos armados.

No livro há também discussão da necessidade de um equilíbrio entre o avanço tecnológico e a mitigação de riscos. Sugere-se que, para maximizar os benefícios potenciais das aplicações militares da IA, é crucial que os líderes militares e

governamentais dos EUA desenvolvam políticas robustas que garantam o uso ético e responsável dessas tecnologias. Isso inclui a implementação de controles rigorosos para garantir que operadores humanos mantenham a supervisão e o controle sobre decisões críticas no campo de batalha (MORGAN et al., 2020). No caso específico do uso de Chatbot, uma das principais preocupações éticas com a militarização desses sistemas é o potencial para a disseminação de desinformação e propaganda. Chatbots militares podem ser programados para espalhar informações falsas ou enganosas, manipulando a opinião pública e confundindo adversários. Esse uso da tecnologia pode violar princípios éticos básicos, como a honestidade e a integridade, e pode ter consequências graves para a confiança pública nas instituições militares e governamentais (MORGAN et al., 2020). Chatbots militares também podem ser utilizados para coletar informações pessoais de indivíduos sem o seu consentimento. A vigilância não autorizada e a coleta de dados pessoais levantam questões éticas significativas relacionadas à privacidade e aos direitos humanos. Morgan et al. (2020) destaca que o uso de tecnologias de IA para monitorar e analisar comunicações privadas pode resultar em abusos de poder e violação de direitos fundamentais.

A análise feita por Morgan et al. (2020) leva a um pensamento crítico sobre a militarização dos mais poderosos Chatbots que existem no mundo hoje, que são o ChatGPT e Gemini (entre outros não tão difundidos ao conhecimento da população). Estes modelos de IA têm uma capacidade inimaginável de respostas que porventura podem ser usadas no meio militar. Som Biwas, um médico pesquisador da *University of Tennessee Heath Science Center, Memphis, Tennessee*, vem utilizando o ChatGPT como fonte para a produção de artigos científicos. Um dos seus trabalhos, chamado de *Prospective Role of ChatGPT in the Military: According to ChatGPT*, descreve de forma bastante acertiva de como o ChatGPT poderia ser utilizado no combate, no apoio ao combate e no apoio de serviço ao combate. A geração de texto de forma extensa faz desses modelos LLM uma arma de desinformação e manipulação de opinião pública sem quaisquer limites, pois podem ser utilizados por qualquer ser humano que tenha acesso à internet. Além disso, esses modelos podem ajudar a criar outros modelos de IA capazes de violar a dignidade e privacidade de indivíduos ou de grupos. Logo, observa-se a necessidade de leis regulamentares para os LLM, impedindo que conteúdos maliciosos sejam produzidos e utilizados.

Em resumo, a militarização de Chatbots envolve uma série de preocupações éticas que precisam ser cuidadosamente consideradas. Desde a disseminação de

desinformação até a violação de privacidade e a desumanização da guerra, as implicações do uso dessa tecnologia em contextos militares são profundas e complexas. Como destacado por Morgan et al. (2020), é essencial que líderes militares e governamentais desenvolvam políticas robustas para garantir que o uso de Chatbots e outras tecnologias de IA sejam conduzidos de maneira ética e responsável.

2.4 LACUNAS E OPORTUNIDADES

Este projeto é desenvolvido com a intenção de dar suporte aos Oficiais e Praças responsáveis e contribuintes pelos planejamentos das Operações Militares e confecções de Documentos Operativos, conseqüentemente, facilitando a produção de Ordem de Operação de um Batalhão de Infantaria. Inicialmente, o Chatbot é capaz de extrair informações importantes dos Manuais CGFCFN Ostensivos de interesse específico, o que reduz o tempo de busca de informações específicas para a produção Documentos Operativos pelas seções de Estado-Maior. Por outro lado, o programa não é capaz de propor Linhas de Ação aos planejadores, e isto seria um dos pontos mais importantes para produção de um Documento Operativo. Para propor as Linhas de Ação, o Chatbot teria que conseguir identificar em um mapa, ou calco, onde que se situa o batalhão, quem estão nos flancos desse batalhão, quem está à sua retaguarda e identificar no terreno por onde uma tropa conseguiria passar para chegar ao objetivo. Ou seja, a parte mais importante do documento que é a execução, o como se dará o desenrolar da missão, não consegue ser feito pelo Chatbot, pois dependeria de vetorização das cartas e calcos oferecidos pelas seções de Inteligência. Sem contar que necessitaria de inúmeros exemplos desses Documentos Operativos para se treinar um Chatbot que seria capaz de elaborar de forma correta Linhas de Ação ou um Conceito da Operação. Porém, apesar das dificuldades impostas às melhorias deste projeto, não é impossível o desenvolvimento de tal ferramenta.

3 MODELAGEM COMPUTACIONAL

Nesta seção será apresentada a sequência seguida para se chegar ao resultado pretendido, que é a produção de um Chatbot capaz de responder a perguntas relativas à doutrina empregada no Corpo de Fuzileiros Navais, obtida nos

manuais do CGCFN, e identificar partes importantes de documentos desenvolvidos nas seções de Estado-Maior a fim de dar celeridade nos trabalhos dessas seções.

3.1 CRIANDO UMA *API-KEY*

Hoje em dia, a criação de um Chatbot fica muito facilitada devido à integração que pode ser feita com IA's de grandes empresas como o Gemini da Google, ChatGPT da OpenAI, o Copilot da Microsoft e outros de instituições menores. Essas IA's eliminam o trabalho de ter que criar um banco de dados para treinar o Chatbot em linguagem natural ou informações já existentes na internet, o que faz qualquer resposta dada ao *bot* parecer a de um ser humano.

Para utilização de uma IA específica, primeiro é preciso criar uma *API-Key*. Uma *API-Key* (*Application Programming Interface - Key*) é um código utilizado para identificar, autenticar e ou autorizar uma aplicação ou um usuário. Essa “chave”, que pode ser obtida por um indivíduo ou uma entidade/empresa, serve para dar acesso aos produtos desenvolvidos pelas empresas criadoras de IA. Ela não somente dá acesso à própria IA, mas também, como por exemplo, dá acesso a modelos de *text embeddings* e métodos de tokenização que podem ser implementados como módulos em linguagem de programação tipo o Python, e servem para matematizar e vetorizar as letras, palavras ou os textos.

O primeiro passo a se fazer é criar uma conta na plataforma da OpenAI ou na Google. Caso já tenha conta da Google, também pode-se acessar a plataforma da OpenAI utilizando aquela conta, a conta da Microsoft ou da Apple como mostrado na Figura 1 do Anexo A. O acesso à plataforma para a criação da *API-Key* da empresa Google precisa ser via website Google Cloud (Figura 2, Anexo A).

Após a criação da conta em uma das plataformas, a criação da chave *API* passa a ser um processo simples. No site da OpenAI, por exemplo, ao clicar no ícone inicial *your profile*, na coluna à esquerda da tela, e depois no ícone *User API Keys* ou *Project API Keys*. Seguindo-se a sequência citada anteriormente, na tela abaixo, aparecerá o ícone + *Create new secret key* e, ao clicar neste, sua *API-Key* será criada, como mostra a Figura 3.

A criação da chave na plataforma Google Cloud não tem um processo diferente da OpenAI. Após o primeiro acesso ao Google Cloud, é só clicar nos ícones na sequência: *APIs & Services* -> *Credentials* -> *+CREATE CREDENTIALS* -> *API Key*. Na Figura 4, do Anexo A, pode-se visualizar o caminho a ser seguido para a criação da *API-Key* da Google.

Após a criação da chave, tanto da Google quanto da OpenAI, o usuário terá acesso aos diversos software, módulos e suportes técnicos capazes de apoiar a construção de projetos em programação. É importante frisar que há diversas diferenças entre as chaves das duas empresas aqui citadas, e claro, haverá diferenças entre outras empresas no mercado. Porém, a principal divergência entre a *API* da OpenAI e da Google é a disponibilização da sua IA, já que esta empresa libera o uso dos modelos Gemini de forma gratuita, mesmo que em um nível mais limitado e aquela outra não libera o uso dos modelos GPT gratuitamente.

3.2 DESENVOLVIMENTO DO CÓDIGO

Nesta seção será mostrado como o Chatbot foi feito, descrevendo-se a funcionalidade de cada parte do código em linguagem de programação Python.

Com o objetivo de averiguar a aplicabilidade de ambas, um código utilizando a IA da OpenAI e o outro a AI da Google foi produzido. A seguir, tem-se o código implementado com o ChatGPT como IA.

3.2.1 *Layer Flask*

Flask Application (Quadro 2, Apêndice A): O principal ponto de entrada é a aplicação Flask, que provê uma interface API para interagir com o bot. O app define *endpoints* e lida com solicitações dos clientes. Essa camada de aplicação expõe rotas de acesso aos arquivos *static* e interage com a funcionalidade do chat.

3.2.2 Integração OpenAI

Criação de *Embedding* (Quadro 8, do Apêndice A): Usa a API da OpenAI para gerar *embeddings* para as perguntas dos usuários e o texto dos manuais. Isso permite

correspondência baseada em similaridade entre *inputs* do usuário e o conteúdo do documento.

Chat Completion (Quadro 10, do Apêndice A): Usa a *API ChatCompletion* da OpenAI para gerar respostas às perguntas dos usuários baseadas no conteúdo dos manuais.

3.2.3 Camada de Dados

Embeddings Preprocessados (Quadro 4, Apêndice A): essa parte do código carrega informações de um arquivo do tipo JSON (*document_embeddings.json*) – os *embeddings* formados com os textos dos manuais –. Cada entrada nesse arquivo representa um *chunk* (pedaço) do documento com o seu respectivo *embedding* pré-computado, possibilitando uma correspondência por similaridade eficiente, sem que haja a necessidade de recomputar os *embeddings* toda vez que uma requisição seja feita.

3.2.4 APIs e Routes

Static File Serving (Quadro 6, Apêndice A): Apresenta os arquivos estáticos, tais como o CSS e JS, do diretório *static*.

Chat Route (Quadro 10, Apêndice A): Processa solicitações do chat e tenta responder a perguntas sobre abreviação diretamente. Se nenhuma abreviação é detectada, é executada a busca semântica e gera uma resposta baseada no conteúdo mais relevante do documento.

3.2.5 Processamento de Texto e Lógica de Busca

Text Chunks (Quadro 7, Apêndice A): Divide o texto em *chunks* (pedaços), facilitando a manipulação de grandes documentos.

Busca por Abreviações *Regex-Based* e Cálculo do Cosseno (Quadro 9, Apêndice A): Procura por abreviações e suas definições no documento usando expressões regulares, e faz filtragem por similaridades.

Busca Semântica (Quadro 10, Apêndice A): Calcula por similaridade do cosseno entre os *embeddings* da pergunta e cada chunk embedding do documento para encontrar as partes mais relevantes do texto.

3.2.6 Configuração do Servidor e do Ambiente

Variável Ambiente (Quadro 3, Quadro 16, Quadro 19, e Quadro 20): Usa variáveis de ambiente para dados sensíveis, tais como a chave OpenAI API e o número da porta.

Live Reload (Quadro 19, Apêndice F): A seção comentada do módulo *livereload* verifica mudanças nos arquivos, possibilitando atualização do servidor durante o desenvolvimento. Essa arquitetura permite que o *bot* responda a questões sobre os manuais aproveitando-se de *embeddings* pré-computados e dos modelos de linguagem da OpenAI para dar respostas acuradas e apreciar as abreviações.

Com intuito de observar as diferenças entre duas I.A, dois códigos forem desenvolvidos em linguagem Python. O primeiro código, como visto anteriormente, utiliza do GPT como sua I.A. Já o segundo código, Quadro 12, do Apêndice B, faz uso do Gemini. A principal divergência observada entre ambas está no uso, pois a Google disponibiliza a Gemini de forma gratuita, apesar de ter limites em algumas funções. Por causa dessas limitações, as respostas do Chatbot perdem qualidade, não correspondendo às expectativas do projeto que é a capacidade de fazer *reply a query* dos usuários relativa aos documentos da Marinha do Brasil (manuais CGCFN). Tentando suplantar as limitações do Gemini gratuito, mais funções foram implementadas no código, obtendo-se uma melhora das respostas.

3.3 TESTES DO CÓDIGO

Antes de passar o código para uso em rede, faz-se de importância a avaliação do mesmo realizando testes. Pode-se testar o código utilizando a própria *IDE* (*Integrated Development Enviroment*) utilizada para a produção do mesmo ou, se assim desejar, pode ser criado um ambiente de teste em linguagem html em conjunto com JavaScript e CSS, o que dá uma noção futura de como ficará visualmente o aplicativo posto na internet. Nos quadros 13 e 14, respectivamente, constam os

códigos desenvolvidos em html e em JavaScript que formam um ambiente de teste na máquina local.

Para o bom desenvolvimento do aplicativo é de boa prática a criação de uma imagem e, posteriormente, de um container no software Docker Desktop. A criação do container para teste permitirá ao desenvolvedor verificar as dependências necessárias ao lançamento do programa pela internet. Primeiro, deve-se criar um arquivo “*Dockerfile*”, sem extensão específica, e um arquivo de texto nomeado de *requirements* (Quadro 17). Esses arquivos são *scripts* que descrevem como construir uma “*Docker Image*” contendo todas as configurações e dependências necessárias para que o código desenvolvido possa ser testado. No Quadro 16, é mostrado o arquivo *Dockerfile* utilizado para construir a *Docker Image* deste projeto, o arquivo *requirements.txt* mostrado no Quadro 17 – que são módulos do ambiente Python (Obs: foi delimitado a mostrar apenas os módulos mais necessários para desenvolver este projeto, devido à grande quantidade daqueles que foram postos no arquivo *requirements.txt*) – e o arquivo *.env*, que é mostrado 18.

Depois de ter criado os arquivos mencionados acima, insira o comando *docker build -t flask-app-dev .*, no painel *Command Prompt* de sua máquina (Desktop, laptop, entre outros). O comando irá criar no software Docker Desktop um Docker Image, como é mostrado na Figura 5. É importante mencionar que todos os arquivos necessários para o projeto devem estar separados em uma única pasta no computador do usuário, caso o contrário, o comando *build -t flask-app-dev .* irá copiar qualquer arquivo que esteja dentro da mesma pasta que o *Dockerfile* e o arquivo *Flask-app*.

Logo, em seguinte, é de bom costume testar localmente o *Docker Container* antes da expedição do aplicativo na internet com o seguinte comando *docker run -p 5000:5000 -v <directory>:/app flask-app-dev*. Este comando irá conectar a porta 5000 da máquina local com a porta 5000 do container, podendo o desenvolvedor acessar o app via <http://localhost:5000>. Essa parte do código citado acima, *-v <directory>:/app*, faz a ligação do diretório de trabalho da máquina ao */app* dentro do container, possibilitando o desenvolvedor atualizar o código em seu local de trabalho, e automaticamente o código dentro do container irá alterar sem que seja necessário refazer a Docker Image. Estando tudo certo com a produção da imagem e do container, agora é só inserir *docker run --env-file .env -p 5000:5000 -v <directory>:/app flask-app-dev* no *Command Prompt*. Salienta-se que esses códigos são para

máquinas com sistema operacional Windows, sendo um pouco diferentes para outros sistemas operacionais. Após certificar-se de que o aplicativo funciona corretamente na máquina local, o container poderá ser criado no *Google Cloud Run* para *deployment* em webpage. Existem diversas plataformas que servem como *Backend Hosts* para o arquivo Flask-app como AWS, PythonAnywhere, e todas oferecem serviço gratuito, porém com limitações.

3.4 DEPLOYMENT DO APLICATIVO

Após o desenvolvimento do código em plataforma pessoal (<https://localhost:5000>), tendo testado e verificado a correta implementação dos parâmetros utilizados no mesmo, o código poderá ser distribuído em plataformas pela internet.

Para iniciar o processo de *deployment* do código, é necessário que o desenvolvedor utilize de ferramentas de containerização que ajudam a armazenar grande volume de dados, compartilhá-los, e ou modificá-los sem criar a dependência de uma única máquina (computador) na execução de processos. Como visto anteriormente, este trabalho fez uso do software de containerização para chamado *Docker Desktop* o que possibilitou fazer testes na máquina local, adiante, será mostrada a containerização no *Google Cloud*, e na sequência o lançamento do app em rede. É válido comentar que os arquivos citados anteriormente precisam de pequenas alterações ou a implementação no *Google Cloud* não ocorrerá bem. A seguir, será mostrado essas alterações

Dando continuidade ao projeto, em uma conta Google Cloud, crie um novo projeto nomeando-o e identifique o seu ID. Após isso, acesse o link *APIs & Services* e clique em *Library*. Dentro desta biblioteca as seguintes *APIs* deverão ser habilitadas: *Cloud Run API*, *Cloud Build API*, *Cloud Storage API*. A habilitação dessas ferramentas se faz necessária para produção do container no *Google Cloud*. É preciso salientar também a necessidade de habilitar um método de pagamento (e.g. cartão de crédito), ou não será permitido a produção do app. (aplicativo) em rede. Para novos assinantes, por enquanto, a Google disponibiliza \$300 em créditos, por 90 dias, para poder utilizar quaisquer ferramentas do *Google Cloud*. Vencendo os 90 dias, ou acabando-se os créditos, o desenvolvedor poderá continuar usando os serviços sendo cobrado a depender da ferramenta que for utilizada.

Será necessário que o desenvolvedor instale em sua máquina o *Google Cloud SDK*. Este software possibilita que o usuário interaja com a plataforma *Google Cloud* do terminal *Command Prompt*. Com isso, uma autenticação usuário-máquina será feita via terminal com o seguinte comando: *gcloud auth login*. Logo, em seguida, digite o comando *gcloud config set project <ID>*, possibilitando o Google Cloud identificar qual projeto está sendo observado.

Estando tudo certo nos passos mencionados acima, um container será construído ao submeter o comando *gcloud builds submit --tag gcr.io/<ID>/flask-app* no terminal. A posteriori, digite no *Command Prompt* *gcloud run deploy --image gcr.io/<ID>/flask-app --platform managed*, assim, criar-se-á um endereço URL (*Uniform Resource Locator*) para o arquivo Flask-app do tipo <https://your-app-name-<region>.run.app>, possibilitando o acesso de qualquer usuário via internet como mostrado na Figura 6.

3.5 INTEGRAÇÃO DO APP EM WEBPAGE

O desenvolvimento de um aplicativo se dá em várias etapas, desde a implementação do código em sua linguagem de programação (*Backend*) até a interface máquina e aplicativo (*Frontend*). Nesta seção, mostrar-se-á como chegar ao produto (Chatbot) em uma webpage. Lembrando que o *deployment* do app., seguindo todos os passos citas acima, já está feito, tendo como endereço virtual algo parecido com <https://your-app-name-<region>.run.app>. Caso o desenvolvedor tenha vastos conhecimentos em JavaScript, CMS, e html, não terá muitos problemas em fazer com que a página de acesso ao Chatbot fique com uma aparência agradável aos olhos dos usuários. Porém, se o projetista preferir, pois é mais ágil fazer isso, e o Chatbot pode ser integrado a uma página na web já existente, tirando todo o trabalho de ter que estruturar o *Frontend* desde o início.

Por sorte, hoje em dia já existem plataformas que permitem aos usuários criar e gerenciar websites sem necessidade do conhecimento de uma linguagem de programação específica. Com isso, Wix Website foi utilizado como editor Frontend facilitando a integração em webpage.

A interface site-usuário construída com a ajuda do Wix é bem simples e direta. Basta, somente, o desenvolvedor seguir os tópicos deste site fazendo *upload* de imagens, criando *text-boxes* ou outras ferramentas que forem necessárias para a boa

utilização do site produzido. Poderá ser feito tanto uma interface para uso em desktop ou em celular sem maiores dificuldades. A Figura 7 mostra como exemplo a interface site criada com a plataforma Wix. Já na Figura 8, o URL lançado via Google Cloud para o Chatbot foi implementado no site utilizando as funções *Add Elements -> Embed Code -> Popular Embeds -> Embed a Site* em *Wix Text Editor*. Com isso, poderá ser copiado o URL do Chatbot para o site produzido com o Wix, e assim, obter-se-á resposta do chat através da página desenvolvida, por fim, finalizando o projeto.

3.6 AVALIAÇÃO DO APLICATIVO

Como forma de poder avaliar a precisão das respostas dadas pelo Chatbot, foi desenvolvido um questionário contendo 20 perguntas, em que as respostas podem ser obtidas pelos manuais CGCFN implementados neste projeto. A seguir, são demonstradas as referidas tabelas de Perguntas Elaboradas de Acordo com os Manuais, e Respostas Geradas pelo Chatbot, respectivamente.

3.6.1 Análise Comparativa entre Respostas

Para facilitar a análise, as respostas da Tabela 2 podem ser classificadas em três níveis: resposta nível 1) ótima-bom – o *bot* é capaz de gerar com precisão a resposta de acordo com os manuais, nos mínimos detalhes; resposta nível 2) boa-regular – a respostas faz sentido e tem a ver com o assunto, mas por estar bastante resumida ou por não dar mais apropriada ela perde o valor; resposta nível 3) regular-ruim – incapacidade de responder à pergunta, sem sentido ou fora do contexto. Analisando as respostas geradas, foi observado que o Chatbot produziu 10 (50%) respostas nível 1, 6 (30%) respostas nível 2 e 4 (20%) respostas nível 3. Vale salientar que, o nível da pergunta também deve ser observado. Sendo que perguntas mais diretas ou que são melhores formuladas têm uma maior probabilidade de serem encontradas e reproduzidas pelo Chatbot (Como o exemplo da questão 20). Em geral, pode-se inferir, de forma bastante conservativa, que o aplicativo tem uma eficiência acima de 50%.

4 CONCLUSÃO

O projeto do Chatbot foi testado com dois tipos de IA: uma que utiliza a IA da OpenAI, mais especificamente o GPT-4; e a outra, utiliza a IA Gemini 1.5, da empresa Google. O GPT-4 obteve resultados melhores, com um número menor de linhas de código escritas do que o modelo Gemini 1.5. Um dos motivos dessa diferença pode estar no método de vetorização do texto, em que a OpenAI disponibiliza o método *text-embedding-ada-002*, que cria vetores de 1536 dimensões e a Google disponibiliza o método *text-embedding-004*, que gera vetores de 768 dimensões. Essa diferença de duas vezes de tamanho entre tais vetores pode estar ocasionando perda de informação, fazendo com que o Gemini 1.5 não consiga identificar qual pergunta está sendo feita (usando a função *cosine*) a respeito dos documentos vetorizados. Apesar dos maus resultados, é importante enfatizar que toda ferramenta Google usada neste trabalho foi gratuitamente ofertada, logo, elas têm limitações o que também justifica a divergência obtida em relação as respostas dadas pelo modelo GPT-4.

REFERÊNCIAS

- AMINI, ALEXANDER. **MIT 6.S191**: Convolution Neural Networks, 2023. Disponível em: https://www.youtube.com/watch?v=NmLK_WQBxB4&list=PLtBw6njQRU-rwp5__7C0oIVt26ZgjG9NI&index=11 Acesso em: 9 nov. 2024.
- AMINI, ALEXANDER. **MIT 6.S191**: Deep Generative Modeling, 2023. Disponível em: https://www.youtube.com/watch?v=3G5hWM6jqPk&list=PLtBw6njQRU-rwp5__7C0oIVt26ZgjG9NI&index=12 Acesso em: 9 nov. 2024.
- AMINI, ALEXANDER. **MIT 6.S191**: Introduction to Deep Learning, 2023. Disponível em: https://www.youtube.com/watch?v=QDX-1M5Nj7s&list=PLtBw6njQRU-rwp5__7C0oIVt26ZgjG9NI&index=9 Acesso em: 9 nov. 2024.
- AMINI, ALEXANDER. **MIT 6.S191**: Learning new frontiers, 2023. Disponível em: https://www.youtube.com/watch?v=FHeCmnNe0P8&list=PLtBw6njQRU-rwp5__7C0oIVt26ZgjG9NI&index=15 Acesso em: 9 nov. 2024.
- AMINI, ALEXANDER. **MIT 6.S191**: Recurrent Neural Networks, Transformers, and Attention, 2023. Disponível em: https://www.youtube.com/watch?v=ySEx_Bqxvvo&list=PLtBw6njQRU-rwp5__7C0oIVt26ZgjG9NI&index=10 Acesso em: 9 nov. 2024.
- AMINI, ALEXANDER. **MIT 6.S191**: Reinforcement Learning, 2023. Disponível em: https://www.youtube.com/watch?v=AhyznRSDjw8&list=PLtBw6njQRU-rwp5__7C0oIVt26ZgjG9NI&index=14 Acesso em: 9 nov. 2024.
- AMINI, ALEXANDER. **MIT 6.S191**: Robust and trustworthy Deep Learning, 2023. Disponível em: https://www.youtube.com/watch?v=kliO4VSrivU&list=PLtBw6njQRU-rwp5__7C0oIVt26ZgjG9NI&index=13 Acesso em: 9 nov. 2024.
- AMINI, ALEXANDER. **MIT 6.S191**: Text-to-image generation, 2023. Disponível em: https://www.youtube.com/watch?v=SA-v6Op2kL4&list=PLtBw6njQRU-rwp5__7C0oIVt26ZgjG9NI&index=16 Acesso em: 9 nov. 2024.
- AMINI, ALEXANDER. **MIT 6.S191**: Text-to-image generation, 2023. Disponível em: https://www.youtube.com/watch?v=SA-v6Op2kL4&list=PLtBw6njQRU-rwp5__7C0oIVt26ZgjG9NI&index=16 Acesso em: 9 nov. 2024.
- AMINI, ALEXANDER. **MIT 6.S191**: The future of robot learning, 2023. Disponível em: https://www.youtube.com/watch?v=WHvWSYKGMdQ&list=PLtBw6njQRU-rwp5__7C0oIVt26ZgjG9NI&index=18 Acesso em: 9 nov. 2024.
- AMINI, ALEXANDER. **MIT 6.S191**: The modern era of statistics, 2023. Disponível em: https://www.youtube.com/watch?v=p1NpGC8K-vs&list=PLtBw6njQRU-rwp5__7C0oIVt26ZgjG9NI&index=17 Acesso em: 9 nov. 2024.
- BISWAS, SOM. **Prospective role of ChatGPT in the military**: according to ChatGPT. *Qeios*, 27 fev. 2023. Disponível em: <https://doi.org/10.32388/8WYYOD>. Acesso em: 4 nov. 2024.

GANEGEDARA, THUSHAN. **Natural Language Processing with Tensorflow**. 2nd. ed. Birmingham, United Kingdom: Packt Publishing, 2022.

GENERATIVE DEEP LEARNING WITH PYTHON: **unleashing the creative power of AI**. 1st. ed. Dallas, Texas: Quantum Technologies LLC, 2023.

MORGAN, FORREST E.; BOUDREAUX, BENJAMIN; LOHN, ANDREW J.; ASHBY, MARK; CURRIDEN, CHRISTIAN; KLIMA, KELLY; GROSSMAN, DEREK. **Military applications of artificial intelligence: ethical concerns in an uncertain world**. Santa Monica, Califórnia: RAND Corporation, 2020. Disponível em: https://www.rand.org/pubs/research_reports/RR3139-1.html. Acesso em: 4 nov. 2024.

MURPHY, KEVIN P. **Machine Learning: a probabilistic perspective**. Cambridge: MIT Press, 2012.

MURUGAN, M. **Natural language processing (NLP)**. Apresentação. Avinashilingam University, 03 abr. 2024. Disponível em: <https://www.researchgate.net/publication/379513103>. Acesso em: 4 nov. 2024.

OPENAI. Assistência no desenvolvimento de código em Python utilizando o ChatGPT. *ChatGPT (modelo GPT-4)*. Disponível em: <https://chat.openai.com/>. Acesso em: 15 set 2024.

RASHID, ADIB BIN; KAUSIK, ASHFAKUL KARIM; SUNNY, AHAMED AL HASSAN; BAPPY, MEHEDY HASSAN. Artificial intelligence in the military: an overview of the capabilities, applications, and challenges. *International Journal of Intelligent Systems*, v. 2023, p. 1-31, 2023. DOI: 10.1155/2023/8676366.

STANFORD ONLINE. **Stanford CSS224N: NLP with Deep Learning | Winter 2019 | Lecture 1 – Introduction and Word Vectors**, 2019. Disponível em: <https://www.youtube.com/watch?v=8rXD5-xhemo&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z>. Acesso em: 4 nov. 2024.

STANFORD ONLINE. **Stanford CSS224N: NLP with Deep Learning | Winter 2019 | Lecture 2 – Word Vectors and Word Senses**, 2019. Disponível em: <https://www.youtube.com/watch?v=kEMJRjEdNzM&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z&index=2>. Acesso em: 4 nov. 2024.

STANFORD ONLINE. **Stanford CSS224N: NLP with Deep Learning | Winter 2019 | Lecture 3 – Neural Networks**, 2019. Disponível em: <https://www.youtube.com/watch?v=8CWyBNX6eDo&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z&index=3>. Acesso em: 4 nov. 2024.

STANFORD ONLINE. **Stanford CSS224N: NLP with Deep Learning | Winter 2019 | Lecture 4 – Backpropagation**, 2019. Disponível em: <https://www.youtube.com/watch?v=yLYHDSv-288&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z&index=4>. Acesso em: 4 nov. 2024.

STANFORD ONLINE. **Stanford CSS224N: NLP with Deep Learning | Winter 2019 | Lecture 5 – Dependency Parsing**, 2019. Disponível em: https://www.youtube.com/watch?v=nC9_RfjYwqA&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z&index=5. Acesso em: 4 nov. 2024.

STANFORD ONLINE. **Stanford CSS224N: NLP with Deep Learning | Winter 2019 | Lecture 6 – Language Models and RNNs**, 2019. Disponível em: <https://www.youtube.com/watch?v=iWea12EAu6U&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z&index=6>. Acesso em: 4 nov. 2024.

STANFORD ONLINE. **Stanford CSS224N: NLP with Deep Learning | Winter 2019 | Lecture 7 – Vanishing Gradients, Fancy RNNs**, 2019. Disponível em: <https://www.youtube.com/watch?v=QEw0qEa0E50&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z&index=7>. Acesso em: 4 nov. 2024.

STANFORD ONLINE. **Stanford CSS224N: NLP with Deep Learning | Winter 2019 | Lecture 8 – Translation, Seq2Seq, Attention**, 2019. Disponível em: <https://www.youtube.com/watch?v=XXtpJxZBa2c&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z&index=8>. Acesso em: 4 nov. 2024.

STANFORD ONLINE. **Stanford CSS224N: NLP with Deep Learning | Winter 2019 | Lecture 9 – Practical Tips for Projects**, 2019. Disponível em: <https://www.youtube.com/watch?v=fyqm8fRDgl0&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z&index=9>. Acesso em: 4 nov. 2024.

STANFORD ONLINE. **Stanford CSS224N: NLP with Deep Learning | Winter 2019 | Lecture 10 – Question Answering**, 2019. Disponível em: <https://www.youtube.com/watch?v=yldF-17HwSk&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z&index=10>. Acesso em: 4 nov. 2024.

STANFORD ONLINE. **Stanford CSS224N: NLP with Deep Learning | Winter 2019 | Lecture 11 – Convolutional Networks for NLP**, 2019. Disponível em: <https://www.youtube.com/watch?v=EAJoRA0KX7I&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z&index=11>. Acesso em: 4 nov. 2024.

STANFORD ONLINE. **Stanford CSS224N: NLP with Deep Learning | Winter 2019 | Lecture 12 – Subword Models**, 2019. Disponível em: <https://www.youtube.com/watch?v=9oTHFx0Gg3Q&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z&index=12>. Acesso em: 4 nov. 2024.

STANFORD ONLINE. **Stanford CSS224N: NLP with Deep Learning | Winter 2019 | Lecture 13 – Contextual Word Embeddings**, 2019. Disponível em: <https://www.youtube.com/watch?v=S-CspeZ8FHc&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z&index=13>. Acesso em: 4 nov. 2024.

STANFORD ONLINE. **Stanford CSS224N: NLP with Deep Learning | Winter 2019 | Lecture 14 – Transformers and Self-Attention**, 2019. Disponível em:

<https://www.youtube.com/watch?v=5vcj8kSwBCY&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z&index=14>. Acesso em: 4 nov. 2024.

STANFORD ONLINE. **Stanford CSS224N: NLP with Deep Learning | Winter 2019 | Lecture 15 – Natural Language Generation**, 2019. Disponível em: <https://www.youtube.com/watch?v=4uG1NMKNWCU&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z&index=15>. Acesso em: 4 nov. 2024.

STANFORD ONLINE. **Stanford CSS224N: NLP with Deep Learning | Winter 2020 | BERT and Other Pre-trained Language Models**, 2019. Disponível em: <https://www.youtube.com/watch?v=knTc-NQsJKA&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z&index=22>. Acesso em: 4 nov. 2024.

SVENMARCK, PETER; LUOTSINEN, LINUS; NILSSON, MATTIAS; SCHUBERT, JOHAN. **Possibilities and challenges for artificial intelligence in military applications**. *STO-MP-IST-160*, Swedish Defence Research Agency, Stockholm, Suécia, 2020. Disponível em: <https://www.researchgate.net/publication/326774966>. Acesso em: 4 nov. 2024.

SZABADFÖLDI, ISTVÁN. **Artificial intelligence in military application – opportunities and challenges**. *Land Forces Academy Review*, v. XXVI, n. 2(102), p. 157-165, 2021. DOI: 10.2478/raft-2021-0022.

APÊNDICES

APÊNDICE A – Módulo OpenAI.

Quadro 1 – Importação de Bibliotecas para o Projeto.

```
import openai
import os
import json
from scipy.spatial.distance import cosine
from flask import Flask, request, jsonify, send_from_directory
import re
from livereload import Server
```

Fonte: Elaborado pelos autores.

Quadro 2 – Inicialização do Módulo *Flask*.

```
app = Flask(__name__)
```

Fonte: Elaborado pelos autores.

Quadro 3 – Criação de Ambiente Seguro para a *API-Key*.

```
# Set your OpenAI API key here
openai.api_key = os.getenv('OPENAI_API_KEY')
```

Fonte: Elaborado pelos autores.

Quadro 4 – Carregamento do Arquivo JSON.

```
# Load preprocessed document data
with open('document_embeddings.json', 'r') as f:
    document_data = json.load(f)
```

Fonte: Elaborado pelos autores.

Quadro 5 – Busca o Arquivo HTML no Diretório.

```
@app.route('/')
```

```
def index():
    """Serve the main chat HTML file."""
    return send_from_directory(r'C:\Users\user_\OneDrive\
Documentos', 'ChatJuruna.html')
```

Fonte: Elaborado pelos autores

Quadro 6 – Busca o Arquivo *Static* no Diretório.

```
@app.route('/static/<path:path>')
def send_static(path):
    return send_from_directory('static', path)
```

Fonte: Elaborado pelos autores.

Quadro 7 – Função para Divisão de Textos em Chunks.

```
def chunk_text(text, chunk_size=100):
    words = text.split()
    for I in range(0, len(words), chunk_size):
        yield " ".join(words[i:I + chunk_size])
```

Fonte: Elaborado pelos autores com o auxílio da ferramenta ChatGPT (2024).

Quadro 8 – Função para a Criação de Embeddings (Vetorização).

```
def create_embeddings(text):
    response = openai.Embedding.create(
        model="text-embedding-ada-002",
        input=text
    )
    return response['data'][0]['embedding']
```

Fonte: Elaborado pelos autores com o auxílio da ferramenta ChatGPT (2024).

Quadro 9 – Função de Busca Direta por Abreviações Utilizando RegEx e Similarity.

```
def search_abbreviation(query, document_data, max_words=10):
    abbreviation_pattern = re.compile(r'\b' + re.escape(query)
+ r'\b\s*-\s*(.*)', re.IGNORECASE)

    best_match = None
    best_similarity = float('inf') # Start with the maximum
possible distance
    for chunk_data in document_data:
        match = re.search(abbreviation_pattern, chunk_data['text'])
        if match:
            full_definition = match.group(1).strip()
            truncated_definition = full_definition[:max_words] + '...' if
len(full_definition.split()) > max_words else full_definition
            query_embedding = create_embeddings(query)
            definition_embedding = create_embeddings(truncated_definition)
            similarity = cosine(query_embedding,
definition_embedding)
            if similarity < best_similarity: # Lower
distance means more similar
                best_similarity = similarity
                best_match = truncated_definition
    return best_match
```

Fonte: Elaborado pelos autores com o auxílio da ferramenta ChatGPT (2024).

Quadro 10 – Integração de Respostas do Chat.

```
@app.route('/chat', methods=['POST'])
def chat():
```

```

user_input = request.json.get('message')

# Extract potential abbreviation from the user input
abbreviation_query = re.findall(r'\b[A-Z]{2,}\b',
user_input)

# Keywords that may indicate the user is asking for a
definition
definition_keywords = ["o que é", "o que significa",
"defina", "definição de"]

if abbreviation_query and any(keyword in
user_input.lower() for keyword in definition_keywords):
    for abbreviation in abbreviation_query:
        direct_result =
search_abbreviation(abbreviation, document_data)
        if direct_result:
            bot_reply = f"A definição de
'{abbreviation}' é: {direct_result}"
            return jsonify({"reply": bot_reply})
try:
    # Embed the user's query
    query_embedding = create_embeddings(user_input)

    # Search for the most relevant document chunk
    most_relevant_chunk = None
    highest_similarity = -1

    for chunk_data in document_data:
        chunk_embedding = chunk_data['embedding']
        similarity = 1 - cosine(query_embedding,
chunk_embedding)
        if similarity > highest_similarity:
            most_relevant_chunk = chunk_data['text']

```

```

        highest_similarity = similarity

    if most_relevant_chunk:
        print(f"Seção do documento selecionada:
{most_relevant_chunk[:500]}") # Log the first 500 characters
of the chunk

# Generate a response using the most relevant document chunk
    response = openai.ChatCompletion.create(
        model="gpt-4",
        messages=[
            {"role": "system", "content": f"Use o
seguinte conteúdo do documento para responder à pergunta:
{most_relevant_chunk}"},
            {"role": "user", "content": user_input}
        ]
    )

    bot_reply =
response['choices'][0]['message']['content']
    return jsonify({"reply": bot_reply})
    else:
        return jsonify({"reply": "Não foi possível
encontrar informações relevantes no documento."})

    except Exception as e:
        print(f"Error occurred: {e}")
        return jsonify({"error": "Failed to get a response from
the bot."}), 500

```

Fonte: Elaborado pelos autores com o auxílio da ferramenta ChatGPT (2024).

Quadro 11 – Comando de Execução Principal e Módulo Live Reload Aplicado.

```
if __name__ == '__main__':  
    port = int(os.environ.get("PORT", 5000))  
    app.run(host="0.0.0.0", port=port)  
    server = Server(app.wsgi_app)  
    # Watch static folder for changes in CSS/JS  
    server.watch('static/')  
    # Watch templates folder for changes in HTML files  
    server.watch('Openai_Bot/')
```

Fonte: Elaborado pelos autores com o auxílio da ferramenta ChatGPT (2024).

APÊNDICE B – Código com o Módulo Gemini.**Quadro 12 – Código Produzido com a IA do Google.**

```
from flask import Flask, render_template, request, jsonify
import google.generativeai as gemini
import json
import numpy as np
from sentence_transformers import SentenceTransformer
from sklearn.decomposition import PCA

app = Flask(__name__)

app.secret_key = 'yyyyyyyy'
# Step 1: Configure the API key
gemini.configure(api_key="xxxxxxxxxxxx")

# Step 2: Initialize the Generative Model Client
gemini_client = gemini.GenerativeModel()

# Step 1: Load preprocessed document data
with open('new_embeddings_2.0.json', 'r') as f:
    document_data = json.load(f)
print(f"Loaded {len(document_data)} chunks from the JSON
file.") # Debugging

# Load the sentence transformer model
sentence_model = SentenceTransformer('paraphrase-MiniLM-L6-
v2')

# Extract all document embeddings for PCA fitting
document_embeddings = np.array([doc['embedding'] for doc in
document_data])

# Initialize PCA and fit it on the document embeddings
```

```

pca = PCA(n_components=384)
pca.fit(document_embeddings)

# Function to create an embedding for a given text using the
sentence transformer model and reduce its dimensionality
def create_query_embedding(text):
    original_embedding =
sentence_model.encode(text).reshape(1, -1) # Ensure 2D array
for PCA
    reduced_embedding = pca.transform(original_embedding) #
Apply the fitted PCA
    return reduced_embedding.flatten() # Flatten back to 1-
D array

# Function to calculate cosine similarity using numpy
def calculate_semantic_similarity(query_embedding,
chunk_embedding):
    # Ensure both embeddings are 1-D numpy arrays
    query_embedding = np.array(query_embedding)
    chunk_embedding = np.array(chunk_embedding)

    # Direct cosine similarity calculation
    similarity = np.dot(query_embedding, chunk_embedding) /
(np.linalg.norm(query_embedding) *
np.linalg.norm(chunk_embedding))
    return similarity

# Function to aggregate top N chunks based on semantic
similarity
def aggregate_top_chunks(query_embedding, query, top_n=1000):
    scored_chunks = []

    for chunk_data in document_data:
        chunk_embedding = chunk_data['embedding']

```

```

        semantic_similarity =
calculate_semantic_similarity(query_embedding,
chunk_embedding)
        scored_chunks.append((chunk_data['text'],
semantic_similarity))

        # Sort chunks by their semantic similarity score, in
descending order
        scored_chunks.sort(key=lambda x: x[1], reverse=True)

        # Aggregate the top N chunks
        aggregated_context = " ".join([chunk[0] for chunk in
scored_chunks[:top_n]])
        return aggregated_context

@app.route("/", methods=["GET"])
def home():
    return render_template("index.html")

@app.route("/get_response", methods=["POST"])
def get_response():
    try:
        user_input = request.json.get("user_input")
        print(f"User Input: {user_input}") # Debugging

        # Embed the user's query using the sentence
transformer model and reduce its dimensionality
        query_embedding = create_query_embedding(user_input)

        # Aggregate the top 3 most relevant chunks based on
semantic similarity
        aggregated_context =
aggregate_top_chunks(query_embedding, user_input, top_n=300)

```

```

        print(f"Aggregated Context:
{aggregated_context[:100]}...") # Debugging

        # Pass the context back to the AI for a final response
        context_message = f"The user is asking about
'{user_input}'. The relevant context from the document is:
{aggregated_context}. Please limit your response to the
information provided in the context."

        final_response =
gemini_client.generate_content(contents=[{"role": "user",
"parts": [{"text": context_message}]}])

        # Safeguard against missing candidates or content
parts
        if final_response.candidates and
len(final_response.candidates) > 0:
            final_candidate = final_response.candidates[0]
            if hasattr(final_candidate, 'content') and
hasattr(final_candidate.content, 'parts'):
                final_part =
final_candidate.content.parts[0]
                if final_part and hasattr(final_part,
'text'):
                    bot_reply = final_part.text.strip()
                    print(f"AI Response: {bot_reply}") #
Debugging
                    return jsonify({"reply": bot_reply})

        return jsonify({"reply": "Nenhuma resposta encontrada
ou o contexto não é claro o suficiente."})

    except Exception as e:
        print(f"Error occurred: {e}") # Debugging

```

```
        return jsonify({"reply": "Ocorreu um erro ao  
processar sua solicitação. Por favor, tente novamente."}),  
        500  
  
if __name__ == "__main__":  
    app.run(debug=True)
```

Fonte: Elaborado pelos autores com o auxílio da ferramenta ChatGPT (2024).

APÊNDICE C – Código em Linguagem HTML.

Quadro 13 – Código Utilizado para a Criação de Ambiente HTML.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>CFN Chatbot</title>
  <link rel="stylesheet" href="/static/css/styles.css">
  <link rel="stylesheet" href="{{ url_for('static',
filename='css/styles.css', v='1.0') }}">
</head>
<body>

<div class="chat-container">
  <div class="chat-header">CFN Chatbot</div>

  <div id="chat-box" class="chat-messages">
    <div id="chat-log"></div>
  </div>

  <div class="chat-input">
    <input type="text" id="user-input" placeholder="Type a
message...">
    <button id="send-button">></button>
  </div>
</div>

<script src="/static/js/script.js"></script>
<script src="{{ url_for('static', filename='js/script.js',
v='1.0') }}"></script>

```

```
</body>  
</html>
```

Fonte: Elaborado pelos autores com o auxílio da ferramenta ChatGPT (2024).

APÊNDICE D – Código em Linguagem JavaScript.**Quadro 14 – Código Utilizado na Confecção da Caixa de Mensagens do Chatbot.**

```
document.getElementById('send-  
button').addEventListener('click', async () => {  
    const userInput = document.getElementById('user-  
input').value;  
    const chatLog = document.getElementById('chat-log');  
  
    console.log("Button clicked. User input:", userInput);  
  
    if (!userInput.trim()) {  
        console.log("Empty input. No message sent.");  
        return; // Avoid sending empty messages  
    }  
  
    // Create a user message bubble  
    const userMessage = document.createElement('div');  
    userMessage.classList.add('chat-message', 'user');  
    const userMessageContent = document.createElement('div');  
    userMessageContent.classList.add('message-content');  
    userMessageContent.textContent = userInput;  
    userMessage.appendChild(userMessageContent);  
    chatLog.appendChild(userMessage);  
  
    // Scroll to the latest message  
    chatLog.scrollTop = chatLog.scrollHeight;  
  
    // Clear input field  
    document.getElementById('user-input').value = '';  
  
    try {
```

```
// Send the message to the bot
const response = await fetch('/chat', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ message: userInput })
});

const data = await response.json();

// Create a bot message bubble
const botMessage = document.createElement('div');
botMessage.classList.add('chat-message', 'bot');
const botMessageContent =
document.createElement('div');
botMessageContent.classList.add('message-content');
botMessageContent.textContent = data.reply || 'No
response received. Please try again.';
botMessage.appendChild(botMessageContent);
chatLog.appendChild(botMessage);

// Scroll to the latest message
chatLog.scrollTop = chatLog.scrollHeight;

} catch (error) {
  console.error("Error fetching bot response:", error);
  const errorMessage = document.createElement('div');
  errorMessage.classList.add('chat-message', 'bot');
  const errorMessageContent =
document.createElement('div');
  errorMessageContent.classList.add('message-content');
  errorMessageContent.textContent = 'Failed to
communicate with the server.';
  errorMessage.appendChild(errorMessageContent);
  chatLog.appendChild(errorMessage);
```

```
        // Scroll to the latest message
        chatLog.scrollTop = chatLog.scrollHeight;
    }

    // Clear the input field and scroll to the latest
message
    document.getElementById('user-input').value = '';
    chatLog.scrollTop = chatLog.scrollHeight;
});
```

Fonte: Elaborado pelos autores com o auxílio da ferramenta ChatGPT (2024).

APÊNDICE E – Código em Linguagem CSS**Quadro 15 – Código Utilizado para Formatação da Caixa de Mensagens.**

```
/* Body styles - simplified */
body {
    font-family: Arial, sans-serif;
    margin: 0;
    background-color: #ffffff; /* background color */
    height: 100vh;
    box-sizing: border-box; /* Ensure padding is included in
the height/width */
}

/* Chatbox container at the bottom center */
.chat-container {
    width: 600px; /* Adjusted width for the chatbox */
    max-width: 100%;
    background-color: #fff;
    border-radius: 8px;
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.1);
    overflow: hidden;
    display: flex;
    flex-direction: column;
    border: 1px solid #ddd;

    /* Positioning at the bottom center */
    position: fixed;
    bottom: 0; /* Sticks to the bottom */
    left: 50%; /* Center horizontally */
    transform: translateX(-50%); /* Adjust horizontal centering
*/
    margin-bottom: 20px; /* Optional margin to give it some
space from the bottom */
}
```

```
}

/* Other styles (same as before) */

.chat-header {
    background-color: #075e54; /* WhatsApp header color */
    color: #fff;
    padding: 15px;
    text-align: center;
    font-weight: bold;
}

#chat-box {
    flex: 1;
    padding: 10px;
    background-color: #ece5dd;
    overflow-y: auto; /* Enable vertical scrolling */
    max-height: 300px; /* Limit height so scrolling can happen
*/
    scroll-behavior: smooth; /* Smooth scrolling effect */
    position: relative;
}

.chat-message {
    margin-bottom: 10px;
    display: flex;
    align-items: flex-start;
}

.chat-message.user .message-content {
    background-color: #dcf8c6; /* Light green for user message
bubble */
    color: #333;
    margin-left: auto;
```

```
        border-radius: 8px 8px 0 8px; /* Rounded corners with
pointed end */
        max-width: 75%;
        padding: 8px 12px;
        box-shadow: 0 1px 2px rgba(0, 0, 0, 0.1);
    }

.chat-message.bot .message-content {
    background-color: #fff; /* White for bot message bubble */
    color: #333;
    margin-right: auto;
    border-radius: 8px 8px 8px 0; /* Rounded corners with
pointed end */
    max-width: 75%;
    padding: 8px 12px;
    box-shadow: 0 1px 2px rgba(0, 0, 0, 0.1);
}

.message-content {
    position: relative;
    word-wrap: break-word;
    line-height: 1.4;
}

.chat-input {
    display: flex;
    border-top: 1px solid #ddd;
    padding: 10px;
    background-color: #f7f7f7; /* WhatsApp input area
background */
}

#user-input {
    flex: 1;
```

```
padding: 10px;
border: 1px solid #ddd;
border-radius: 20px;
outline: none;
font-size: 14px;
}

#send-button {
margin-left: 10px;
padding: 10px 15px;
background-color: #075e54; /* send button color */
color: #fff;
border: none;
border-radius: 50%;
cursor: pointer;
font-size: 16px;
display: flex;
align-items: center;
justify-content: center;
}

#send-button:hover {
background-color: #064a45;
}

/* Scrolling text at the top */
.vertical-scrolling-text {
height: 50px; /* Adjust height for the scrolling area */
overflow: hidden;
background-color: #075e54; /* green */
color: white;
font-size: 16px;
font-weight: bold;
padding: 10px 0;
```

```
position: fixed; /* Stay at the top */
top: 0;
width: 100%;
display: flex;
flex-direction: column;
justify-content: center;
align-items: center;
text-align: center;
}

.vertical-scrolling-text div {
  display: none;
  height: 100%;
  animation: vertical-scroll 6s infinite;
}

.vertical-scrolling-text div:nth-child(1) {
  display: block;
  animation-delay: 0s;
}

.vertical-scrolling-text div:nth-child(2) {
  display: block;
  animation-delay: 2s;
}

.vertical-scrolling-text div:nth-child(3) {
  display: block;
  animation-delay: 4s;
}

@keyframes vertical-scroll {
  0% { transform: translateY(100%); }
  33% { transform: translateY(0); }
```

```
66% { transform: translateY(0); }  
100% { transform: translateY(-100%); }  
}
```

Fonte: Elaborado pelos autores com o auxílio da ferramenta ChatGPT (2024).

APÊNDICE F – Arquivos para Criação de Imagem.

Quadro 16 – Criação de Arquivo Dockerfile.

```
# Use the latest official Python runtime as a parent image
(Python 3.12)
FROM python:3.12-slim

# Set the working directory in the container
WORKDIR /app

# Copy the current directory contents into the container at
/app
COPY . /app

# Install any necessary dependencies
RUN pip cache purge # Clears the pip cache to avoid corrupted
packages
RUN pip install --no-cache-dir -r requirements.txt

# Expose port 5000 (commonly used by Flask for development)
EXPOSE 5000

# Define environment variable for development
ENV FLASK_ENV=development

# Run the Flask app directly for development with live
reloading
CMD ["flask", "run", "--host=0.0.0.0", "--port=5000", "--
reload"]
```

Fonte: Elaborado pelos autores com o auxílio da ferramenta ChatGPT (2024).

Quadro 17 – Exemplo de Arquivo Requirements.txt.

```
Flask==2.1.2
openai==0.27.0
PyPDF2==3.0.0
    scipy==1.9.1
```

Fonte: Elaborado pelos autores.

Quadro 18 – Criação de Arquivo .env.

```
OPENAI_API_KEY= "your_api_key"
FLASK_ENV=development
```

Fonte: Elaborado pelos autores.

Quadro 19 – Alterações no Código Flask-App.

```
@app.route('/')
def index():
    """Serve the main chat HTML file."""
    return send_from_directory('/app', 'ChatJuruna.html')

@app.route('/static/<path:path>')
def send_static(path):
    return send_from_directory('/app/static', path)

if __name__ == '__main__':
    # Use the PORT environment variable set by Cloud Run,
    defaulting to 8080 if not available
    port = int(os.environ.get("PORT", 8080))
    app.run(host="0.0.0.0", port=port)
```

Fonte: Elaborado pelos autores com o auxílio da ferramenta ChatGPT (2024).

Quadro 20 – Alterações no Arquivo Dockerfile.

```
# Make port 8080 available to the world outside this
container
EXPOSE 8080

# Define environment variable for production
ENV FLASK_ENV=production

# Run the application using gunicorn on port 8080
CMD ["gunicorn", "-b", "0.0.0.0:8080", "app:app"]
```

Fonte: Elaborado pelo autor com o auxílio da ferramenta ChatGPT (2024).

Quadro 21 – Alteração no Arquivo .env.

```
FLASK_ENV=production
```

Fonte: Elaborado pelos autores.

APÊNDICE G – Perguntas e Respostas.

1. Quais são as funções do combate? (CGCFN-0-1, 2020, cap.1, p. 9-13)
2. Quais são os fundamentos das Ações Anticarro? (CGCFN-10.2, 2020, cap. 1, p. 3-6)
3. Mostre os tipos de reconhecimento e levantamento de praia (CGCFN-1-4, 2008, cap. 4, p. 86).
4. Descreva os tipos de transposição de curso d'água. (CGCFN-1-5, 2020, cap. 5, p. 16)
5. Como funciona o sistema de reembolso da ONU? (CGCFN-1-8.1, 2010, cap. 6, p. 2)
6. Composição da Força de Desembarque em uma operação de Evacuação de Não-Combatentes. (CGCFN-2-3, 2020, cap. 3, p. 1-3)
7. Caso uma emboscada seja feita pelo inimigo, cuja natureza seja irregular, como poderá ser revertida tal situação? (CGCFN-2-5, 2020, cap. 5, 10-18)
8. Qual proteção física será utilizada caso observado o nível de MOPP igual a 2? (CGCFN-10.3, 2020, cap. 7, p. 42)
9. O Estudo do Ambiente Operacional é constituído por quantas fases? (CGCFN-20, 2021, cap. 4, p. 3)
10. O que é a Matriz de Sincronização? (CGCFN-30, 2013, cap. 6, p. 5)
11. Como é composta uma Matriz de Sincronização? Dê um exemplo. (CGCFN-30, 2013, Anexo D, p. 1)
12. Como é composto o Estado-Maior do BtlInfFuzNav (CGCFN-31.1, 2020, cap. 2, p. 1)
13. O que seria o grupo TAT-LOG? (CGCFN-33, 2008, cap. 3, p. 14)
14. Como se dá o Reconhecimento de Engenharia? (CGCFN-33.1, 2020, cap. 6, p. 1)
15. Quais são as tarefas de um DZDbq? (CGCFN-33.2, 2020, cap.4, p. 4)
16. Cite quais são as necessidades fundamentais do ApAeAprx? (CGCFN-34.1, 2020, cap. 3, p. 6)
17. Os fogos de artilharia numa operação defensiva têm qual finalidade? (CGCFN-50.2, 2020, cap. 5, p. 1)
18. Qual é a composição e organização do CAI? (CGCFN-60, 2008, cap. 4, p. 9)

19. Como se faz a Estimativa de Perdas de pessoal? (CGCFN-60.4, 2020, cap. 4, p. 9)
20. Quais são as fases das Ações Ofensivas efetuadas durante um OMAU? (CGCFN-401, 2020, cap. 5, p. 6)

Tabela 1 - Perguntas Elaboradas de Acordo com os Manuais.

1. O documento não fornece informações específicas sobre as funções do combate.
2. Os fundamentos das ações anticarro (AC) envolvem a integração de todos os meios AC disponíveis com objetivos específicos, como localizar e engajar os blindados inimigos tão distantes das posições amigas quanto possível, reduzir ao máximo o poder de combate dos blindados inimigos antes do seu engajamento com as unidades de assalto, e neutralizar ou destruir os blindados que consigam engajar a ForDbq, com todas as armas disponíveis. Para ser bem-sucedido, o Plano Anticarro deve estar integrado com o Plano de Barreiras e o Plano de Apoio de Fogo, explorando as vantagens proporcionadas por esses planos. Adicionalmente, é preciso considerar que os blindados inimigos são mais eficientes quando atuam em conjunto com a infantaria, portanto, a dissociação dessas forças deve ser prevista nos planos de defesa. As armas AC devem ser utilizadas escalonadamente em profundidade, iniciando o mais distante possível. Na defensiva, deve existir uma força preparada para o primeiro engajamento terrestre com os blindados inimigos.
3. Os tipos de reconhecimento e levantamento de praia são: a) Reconhecimento de Praia: É aquele conduzido em situação de combate, em praias controladas pelo inimigo, por EqRecon lançadas a partir do mar, antes de uma OpAnf, a fim de obter dados sobre a praia e localizar obstáculos a serem destruídos. Pode ser diurno ou noturno. b) Levantamento de Praia: É aquele conduzido após o desembarque de uma ForDbq, sob condições de segurança, ou, ainda, aquele conduzido fora de uma situação de combate, a fim de obter dados hidrográficos acerca de praias designadas. Pode ser paralelo ou perpendicular.

4. O documento não fornece detalhes específicos sobre os tipos de transposição de curso d'água. Ele apenas menciona que alguns rios podem exigir planejamento prévio para travessia e que esse pode chegar a ser uma operação de transposição de curso d'água, um tipo de operação terrestre complexa e com planejamento específico. Além disso, informa que rios com mais de 100m de largura são vistos como obstáculos significativos.
5. O documento não fornece informações sobre o sistema de reembolso da ONU.
6. O documento não fornece informações sobre a composição da Força de Desembarque em uma operação de Evacuação de Não-Combatentes.
7. De acordo com o documento, uma emboscada feita por uma força irregular pode ser revertida através de várias medidas, tais como:
 1. Dissimulação: Informações sobre deslocamentos de comboios devem ser dadas apenas para aqueles que precisam saber e difundir informações falsas para confundir o inimigo.
 2. Iludir o inimigo: Varie a velocidade de marcha e a formação do comboio para impedir que o inimigo preveja seus movimentos.
 3. Falsa Impressão de Força: Dê a impressão de que há mais força do que realmente há. Isso pode desencorajar um ataque.
 4. Alternando Continuamente as Medidas de Segurança: Para que o oponente não possa prever e se aproveitar de ações previsíveis.
 5. Aprendendo com o inimigo: Conhecer os padrões de atuação dos integrantes da força irregular pode auxiliar na preparação de estratégias defensivas.
 6. Coleta de informações: Informações precisas sobre as atividades da força irregular podem ajudar a diminuir a eficácia de suas emboscadas.
 7. Instruir os militares: Os membros do comboio devem estar sempre em alerta e prontos para reagir ao primeiro sinal de uma emboscada.
 8. Patrulhas de Reconhecimento: Fazer patrulhas de reconhecimento para identificar possíveis pontos de emboscada antes da passagem do comboio.
 9. Observação Aérea: Observar o itinerário pelos ares para identificar qualquer movimento suspeito.
 10. Organização do comboio: Certifique-se de que as viaturas não estejam superlotadas, pois isso pode restringir a movimentação dos militares e comprometer a reação ao ataque. Coloque elementos de reconhecimento à frente do comboio e designe militares para a segurança de cada viatura.

8. A proteção física para o nível MOPP-2 inclui equipamento de proteção respiratória individual, que deve ser transportado pelo militar, enquanto a proteção corporal coletiva ainda está disponível na Instalação Logística Sumária (ILS). A proteção para pessoal e material também permanecem disponíveis na ILS.
9. O Estudo do Ambiente Operacional é constituído por quatro fases. O texto menciona várias esferas importantes para análise: o campo externo (que inclui alianças e relações políticas internacionais), aspectos econômicos (como capacidade industrial e situação financeira), aspectos psicossociais (como moral da população e características nacionais), aspectos científico-tecnológicos (como nível de desenvolvimento tecnológico e áreas de pesquisa) e aspectos climáticos (como variações de temperatura e padrões de precipitação). Dentro da climatologia, são destacados vários tipos de climas, incluindo equatorial, tropical, semiárido, subtropical e desértico.
10. A Matriz de Sincronização é uma ferramenta importante para retratar o encadeamento de diferentes atividades no tempo e no espaço, facilitando a visão global da cinemática da operação. Durante o Planejamento Preliminar de Missão (PPM), ela é apropriada para condução do confronto, quando devem ser elaboradas uma matriz para cada confronto entre Linha de Ação (LA) e Pontos de Impacto (PI). A matriz da LA selecionada como Decisão servirá para fornecer subsídios para a escrituração do Conceito da Operação. Este documento complementa as informações constantes do Conceito da Operação e seu objetivo é facilitar o acompanhamento das principais ações planejadas, além de auxiliar na visualização das ações que sejam indispensáveis para a execução da operação.
11. O documento não fornece informações sobre a composição de uma Matriz de Sincronização ou um exemplo dela.
12. O Estado-Maior do BtlInfFuzNav é dividido em Estado-Maior Geral e Especial. O Estado-Maior Geral é composto pelas Seções de Pessoal (S-1), de Inteligência (S-2), de Operações (S-3) e de Logística (S-4). Há também um Imediato (Imto) que chefiava o Estado-Maior, responsável por coordenar, controlar e supervisionar as atividades do mesmo. Além disso, o Estado-Maior do BtlInfFuzNav também é responsável por assessorar o Comandante

(Cmt), exercer a supervisão funcional das subunidades, realizar estudos e elaborar documentos pertinentes.

13. O grupo TAT-LOG é uma organização naval encarregada do controle em operações militares navais. As funções do TAT-LOG são temporárias e o grupo é dissolvido assim que a respectiva organização que representa se estabelece em terra. O grupo é composto geralmente por representantes das Seções de Operações e Logística e possui pessoal e equipamentos de comunicações suficientes. A sua principal função é apoiar operações de desembarque, seja por superfície ou por helicópteros, e suas tarefas incluem encaminhar solicitações da tropa em terra, assessorar o controle naval quanto à localização dos itens solicitados, manter arquivo das séries a serem desembarcadas e assessorar no caso de congestionamentos ou limitações.
14. O Reconhecimento de Engenharia é feito mesmo em terrenos de posse e conhecimento da tropa que realiza o movimento retrógrado. Isso é feito para a verificação do estado das estradas e pontes que pode ter sido alterado pela ação do inimigo, como bombardeios ou sabotagens, ou pelas condições meteorológicas. Além disso, locais críticos e pontos para instalação de barreiras são identificados durante esse reconhecimento.
15. As tarefas de um DZDbq (Destacamento de Zona de Desembarque) incluem:
 1. Preparar e lançar os painéis indicadores;
 2. Instalar os indicadores de direção de vento;
 3. Estabelecer e manter comunicações necessárias;
 4. Reconhecer e selecionar áreas adjacentes aos locais de desembarque para depósitos de suprimentos e outras instalações logísticas, Posto de Comando (PC) do DZDbq e posições de defesa para segurança das instalações logísticas;
 5. Dirigir e controlar os helicópteros demandando a ZDbq operada, bem como apoiar as unidades helitransportadas que nela desembarcam;
 6. Descarregar e carregar helicópteros;
 7. Carregar as redes de carga e paletas;
 8. Evacuar, em viagem de retorno, as baixas ocorridas;
 9. Manter os necessários registros de suprimentos recebidos, fornecidos e disponíveis;
 10. Prover e instalar meios de marcação dos locais de descarga e passagens para operações noturnas;
 11. Estabelecer a segurança local da ZDbq;
 12. Instalar e operar um centro de informações, mantendo uma carta de situação

- atualizada; 13. Coordenar as operações com outros DZDbq (caso existam); e 14. Manter ligação com as tropas que estão sendo apoiadas.
16. As necessidades fundamentais do ApAeAprx são: a) Identificação precisa da LC: A identificação precisa da LC (Linha de Contato) é fundamental para a realização do ApAeAprx. Isso deverá ser feito por meio de pontos notáveis no terreno que são facilmente identificáveis do ar. Se isso não for possível, a LC deverá ser demarcada por meios visuais, como fumígenos coloridos. b) Designação precisa de alvos: Nos planos de apoio de fogo, os alvos devem ser designados normalmente por coordenadas topográficas. A orientação final para o ataque aos alvos deverá ser baseada em acidentes do terreno e pode ser indicada por fumígenos. c) Centralização dos pedidos de apoio de fogo aéreo: A centralização dos pedidos de ApFAe em uma única agência é necessária para assegurar o atendimento às missões, de acordo com a disponibilidade de Anv (Aeronaves) e tripulações e também de acordo com as prioridades estabelecidas. d) Comunicações eficientes: O atendimento aos requisitos fundamentais das comunicações é indispensável na ligação entre as Anv que realizam o ApFAe e as agências coordenadoras/controladoras, para garantir um controle efetivo e flexível e preservar a segurança da tropa e das Anv. e) Resposta imediata: Em alguns momentos da operação, haverá a necessidade de resposta imediata aos pedidos de ApFAe. Essas situações exigem que sejam mantidas Anv no ar, para cumprir com rapidez os pedidos de apoio. Entretanto, sempre que possível, esses pedidos deverão ser atendidos por Anv em alerta no solo ou no convés, a fim de economizar combustível e diminuir o desgaste e a exposição da Anv e sua tripulação.
17. Na operação defensiva, os fogos de artilharia têm várias finalidades. Eles objetivam retardar e desorganizar o inimigo durante sua aproximação através da execução de fogos de longo alcance. Eles também buscam complementar e reforçar o efeito dos obstáculos colocados para deter o avanço do inimigo. Além disso, podem ser utilizados para destruir carros e infantaria inimiga e para amplificar os efeitos causados pelos obstáculos.
18. A composição e organização do CAI (Centro de Análise de Inteligência) é uma Organização por Tarefas, nucleada pelo Batalhão de Comando e

Controle (BtlCmdoCt). Sua composição básica é: - Encarregado do CAI; - Analista-Supervisor; - Analista de Inteligência; - Encarregado de Coordenação das Operações de Inteligência; - Auxiliar de Disseminação; e - Operador de Banco de Dados. Além dessa composição básica, o CAI pode receber reforços, que dependendo do vulto da operação, podem incluir: - Auxiliar do CAI; - Um ou mais Analistas (de Inteligência ou de outras áreas específicas); - Auxiliar do Encarregado de Coordenação das Operações de Inteligência; e - Um ou mais Operadores de Banco de Dados. O CAI também pode ser reforçado por equipes de especialistas, de acordo com os fatores da decisão (Missão, Inimigo, Terreno, Meios e Tempo disponível). Tais equipes podem incluir SIAL; Equipe de Interrogadores e Tradutores; Equipes de Ligação das Agências de Operações de Inteligência; Equipe de Observadores Aéreos, entre outros

19. A Estimativa de Perdas de pessoal se baseia em vários fatores. Primeiro, há diferentes tipos de perdas de pessoal que podem ocorrer, incluindo perdas em combate (por morte, ferimento, desaparecimento, captura ou internamento), perdas fora de combate (relacionadas a treinamento, doenças mentais, doenças gerais e ferimentos causados por condições climáticas e do terreno) e perdas administrativas (que incluem transferências, ausências, deserções e perdas por rodízio). Inicialmente, a Estimativa de Perdas pode ser baseada em dados estatísticos de campanhas passadas, mas esses dados devem ser atualizados para refletir as condições atuais. À medida que a campanha prossegue, cada escalão coleta seus próprios dados e os usa para fazer estimativas mais precisas. A Estimativa de Perdas também leva em consideração uma série de fatores que podem influenciar as perdas, como a localização das unidades e o tipo de operação na qual as forças estão engajadas. Não há duas situações exatamente semelhantes, então vários dados são usados para analisar a experiência de perdas em situações anteriores e determinar médias de perdas aplicáveis. O cálculo das taxas de perdas é apresentado no Anexo C do documento "PLANEJAMENTO DA SEÇÃO DE PESSOAL".

20. O documento fornecido não fornece informações sobre as fases das Ações Ofensivas efetuadas durante uma OMAU. (Obs: ao se trocar a sigla OMAU

por Operação Militar em Área Urbana a resposta dada pelo Chatbot “As fases das Ações Ofensivas efetuadas durante uma Operação Militar em Área Urbana são: isolamento, avanço e limpeza.”)

Tabela 2 - Respostas Geradas pelo Chatbot.

ANEXO A – Figuras dos Ambientes Computacionais.

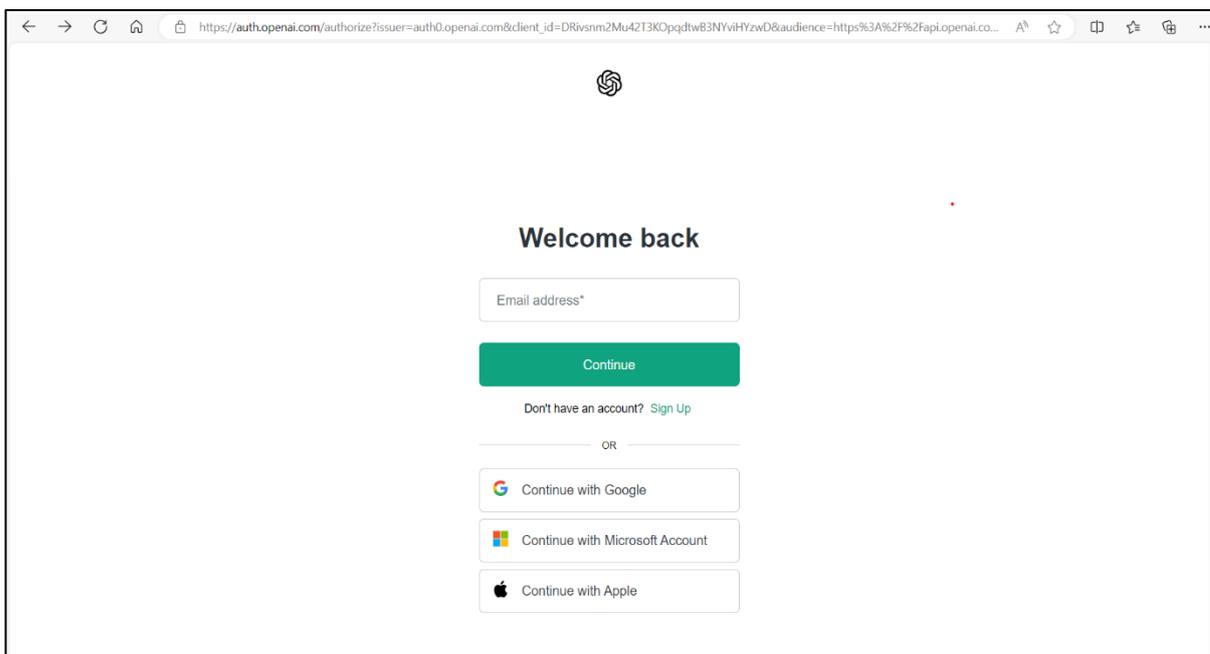


Figura 1 – Acesso à Plataforma da OpenAI.

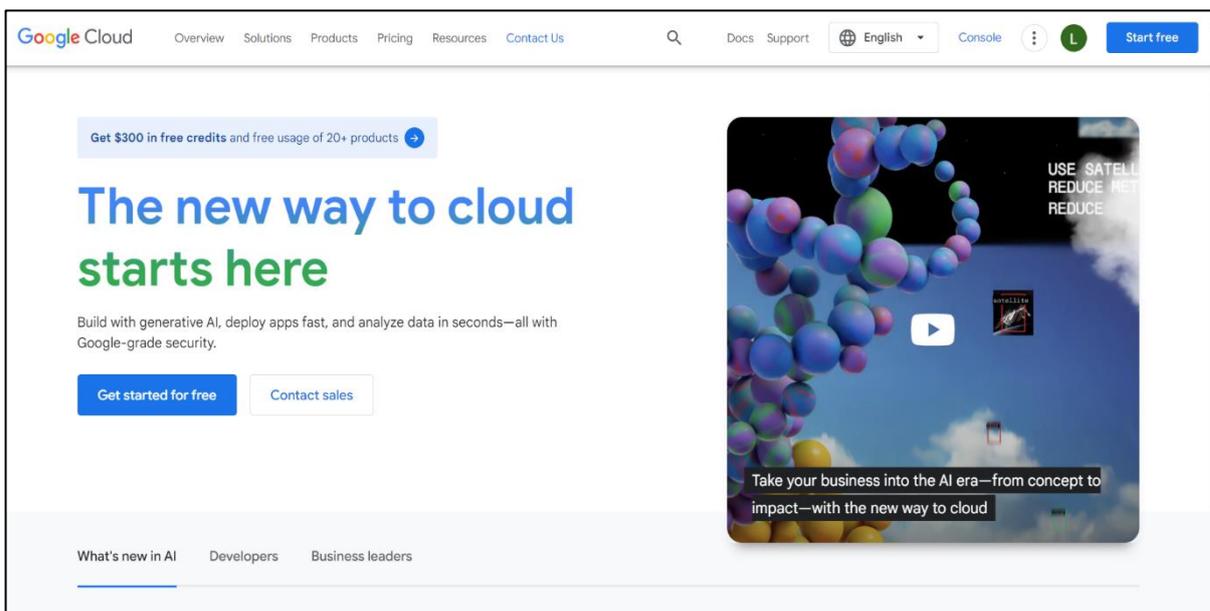


Figura 2 - Acesso à Plataforma Google Cloud.

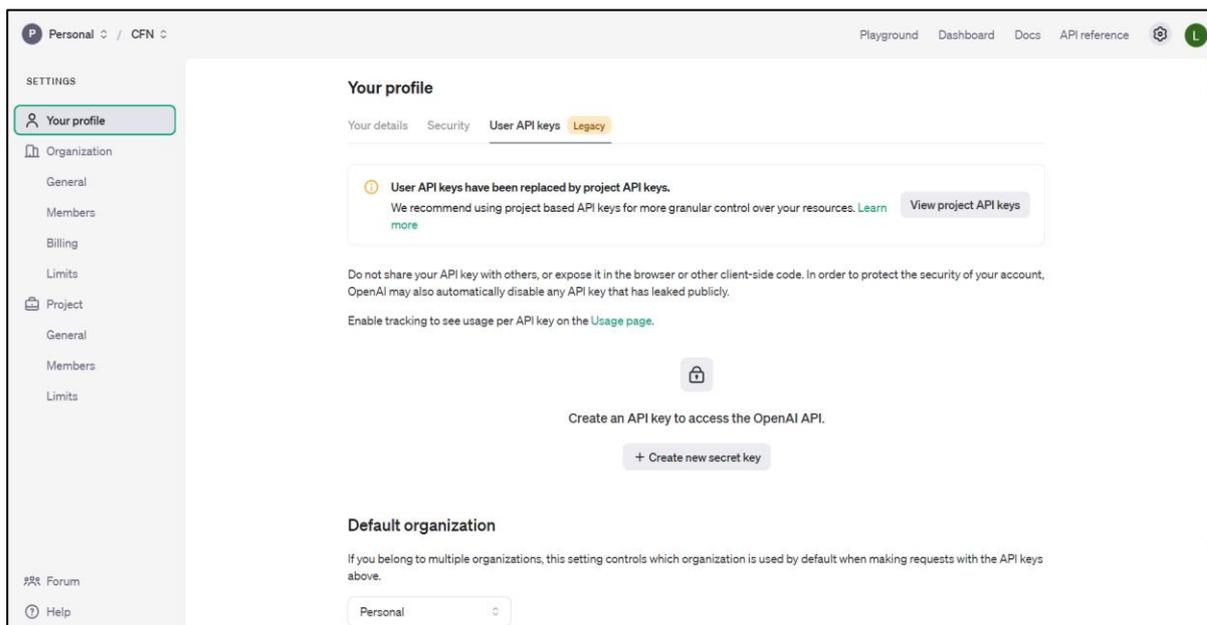


Figura 3 – Criação da API-Key da OpenAI.

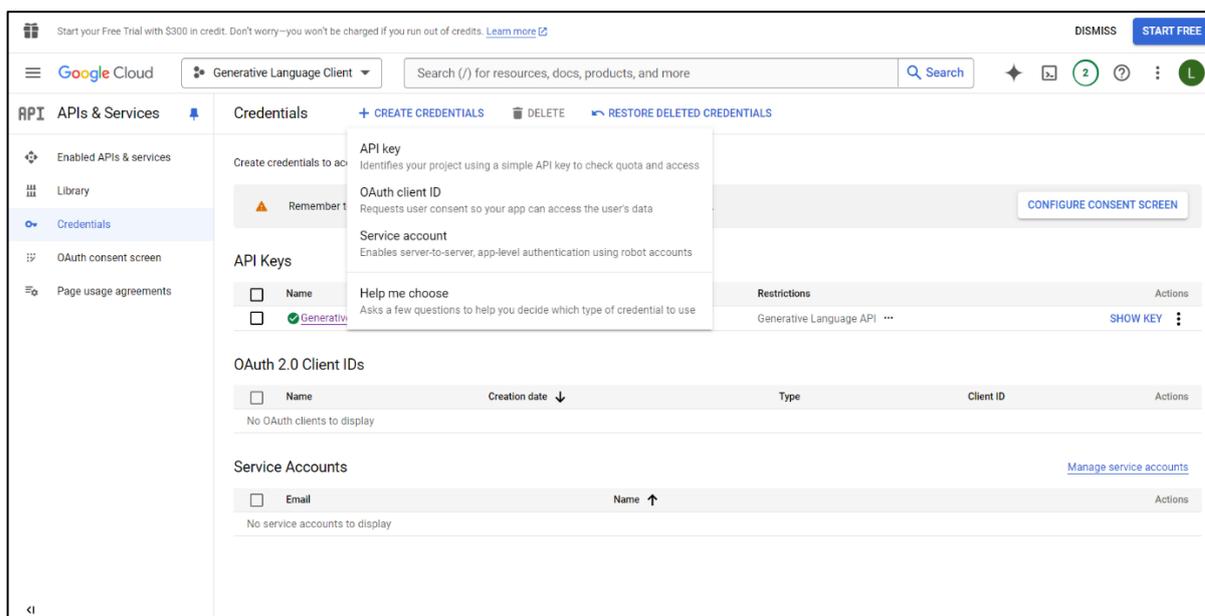


Figura 4 – Acesso ao Google Cloud (APIs & Services).

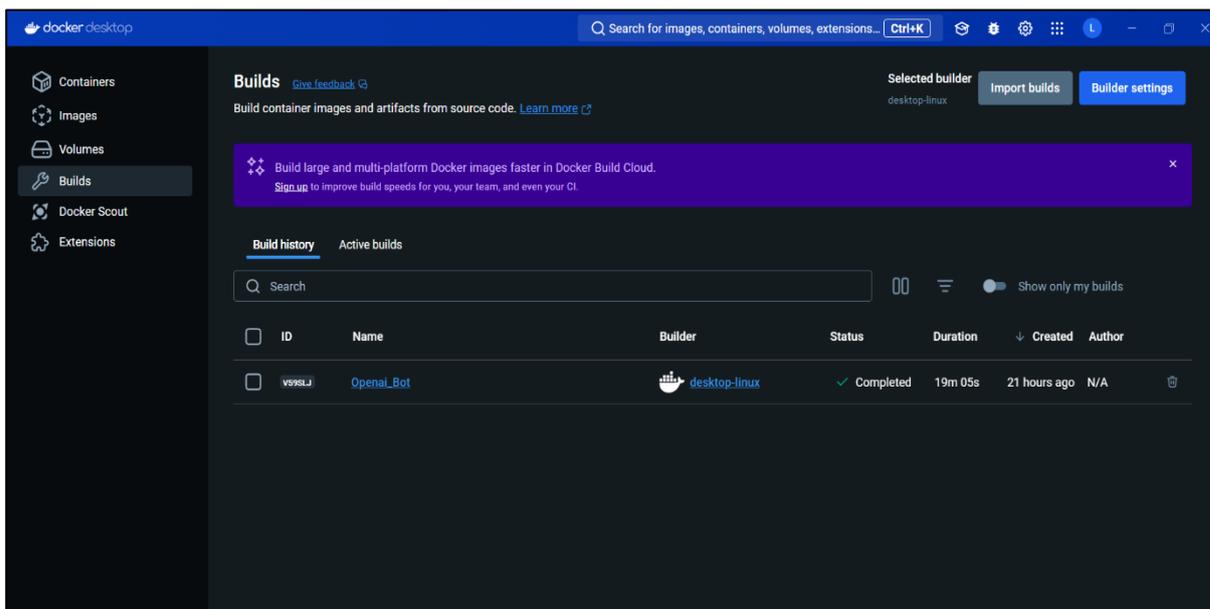


Figura 5 – Docker Build Criada em Docker Desktop.

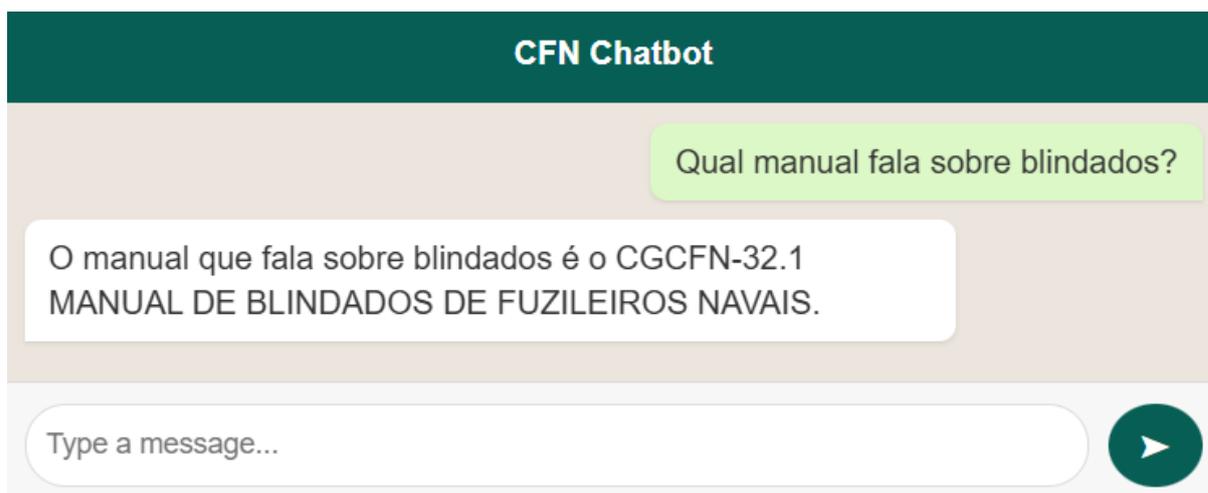


Figura 6 – Demonstração do Chatbot.

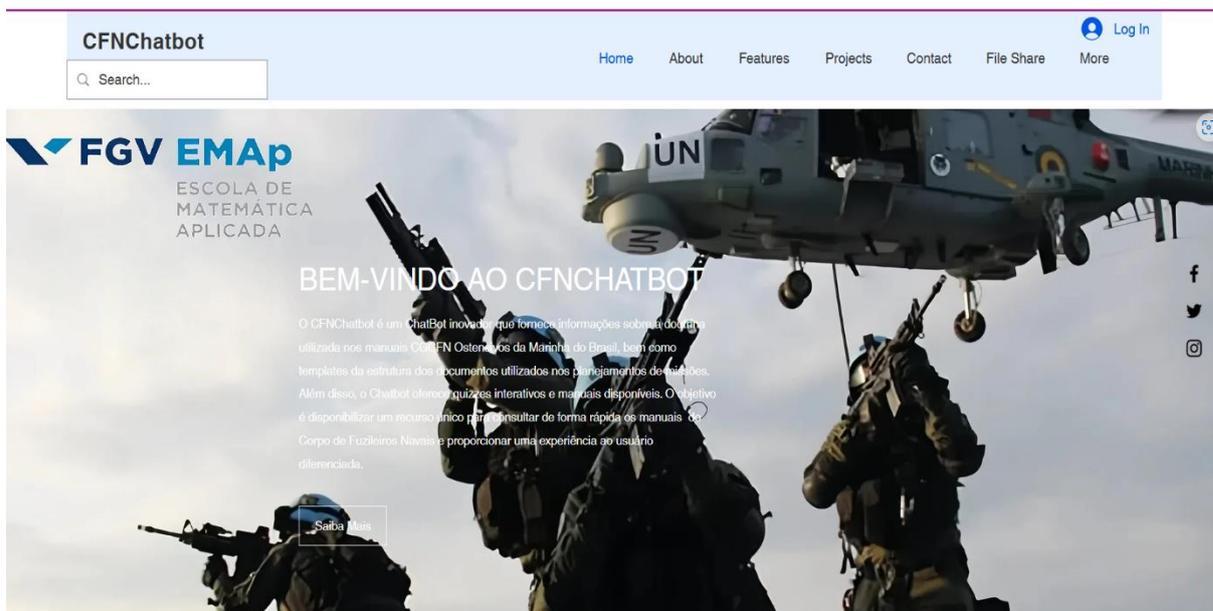


Figura 7 – Frontend Criado na Plataforma Wix.

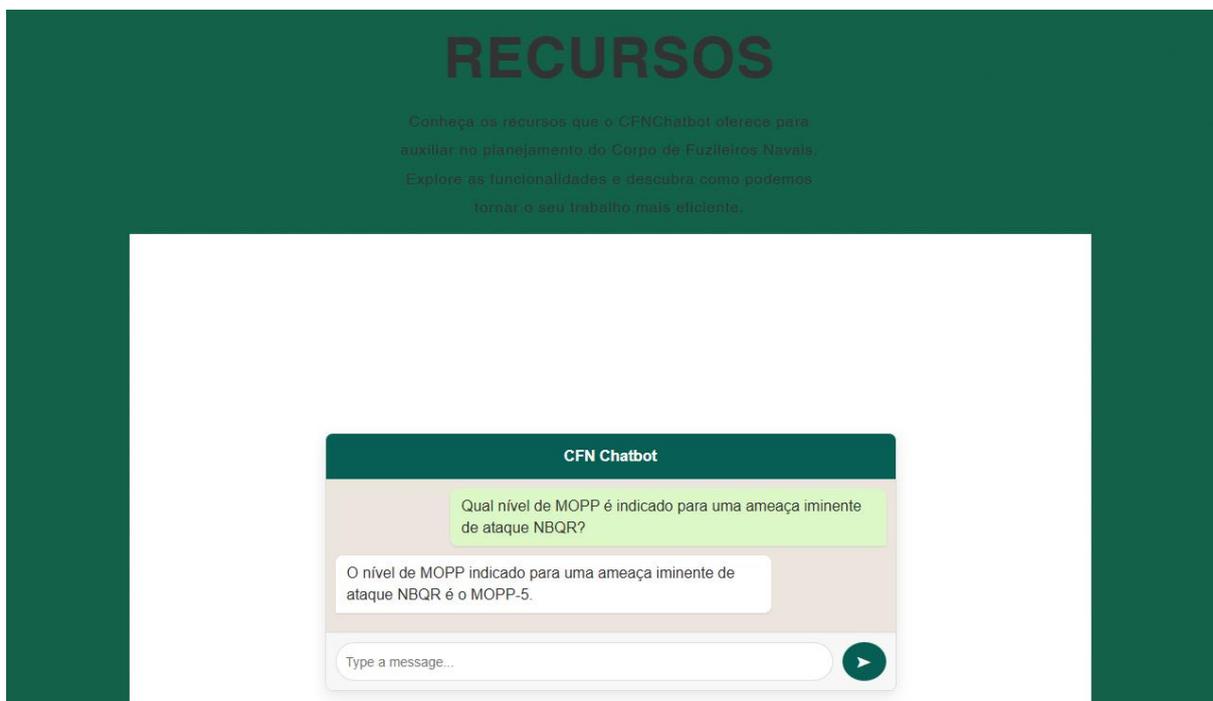


Figura 8 – Integração do URL Criado no Google Cloud com Wix Website.