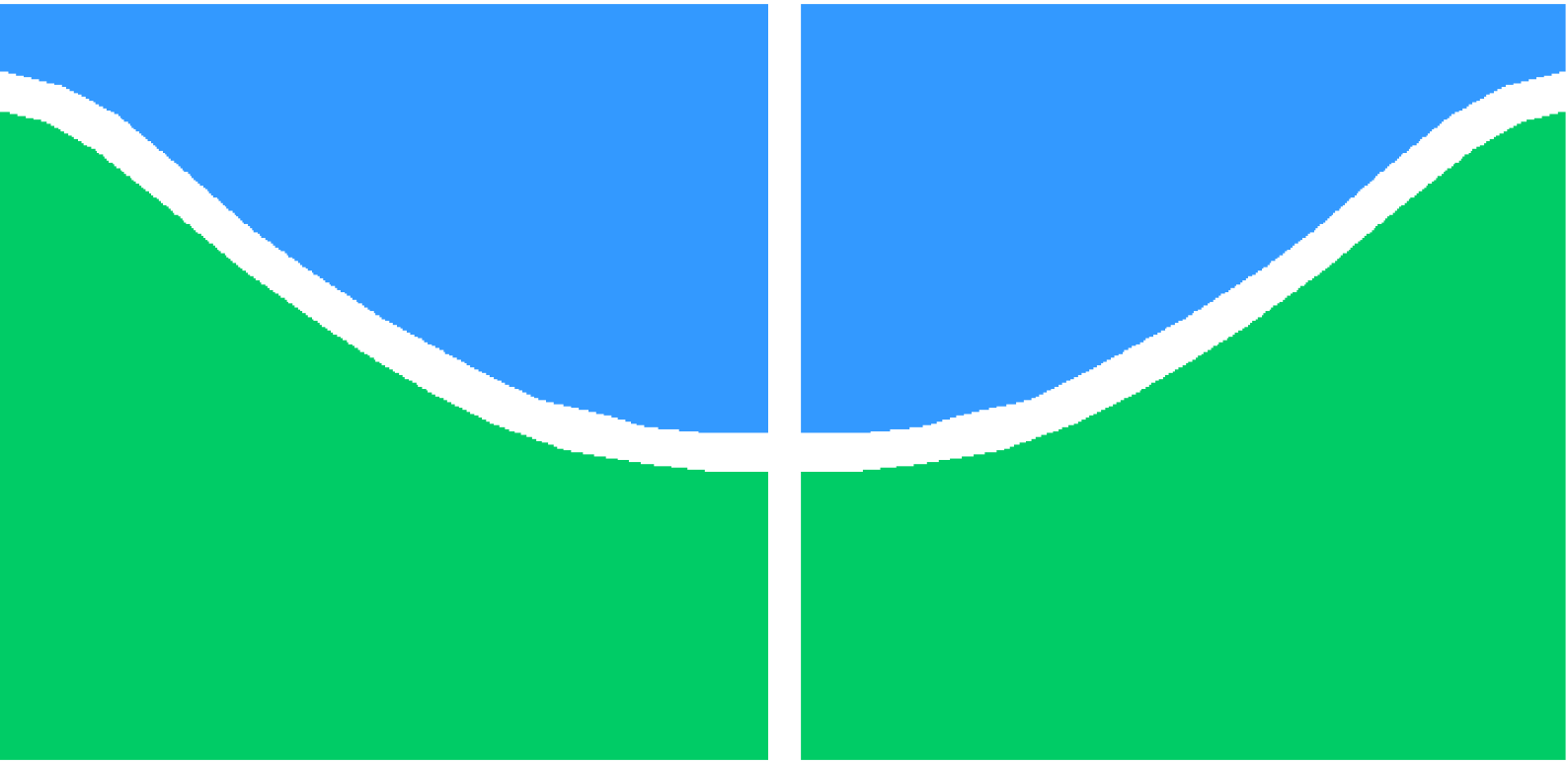


Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

PASSOS E TECNOLOGIAS UTILIZADAS PARA CRIAÇÃO DE UM SERVIÇO DE TRADUÇÃO AUTOMÁTICA DE IDIOMAS, NUMA INFRAESTRUTURA SEM SERVIDOR

Autor: Geovanne Santos Saraiva

Orientador: Fabricio Ataides Braz

Brasília, DF

2021

Geovanne Santos Saraiva

PASSOS E TECNOLOGIAS UTILIZADAS PARA CRIAÇÃO DE UM SERVIÇO DE TRADUÇÃO AUTOMÁTICA DE IDIOMAS, NUMA INFRAESTRUTURA SEM SERVIDOR

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Fabricio Ataides Braz
Orientador

Nilton Correia da Silva
Convidado 1

Bruno Cesar Ribas
Convidado 2

Brasília, DF
2021

Agradecimentos

Agradeço aos professores do curso de Engenharia de Software por apresentarem projetos e tecnologias que nos tiram da zona de conforto, preparando-nos para as tarefas que o mercado de trabalho necessita e com isso nos fazem crescer como profissionais. A minha família, principalmente minha mãe e avós, por sempre me motivarem, estarem ao meu lado e nunca me deixarem desistir.

"... o êxito na educação é consequência de três elementos indissociáveis: o Trabalho, a Solidariedade e a Perseverança.
(PESTALOZZI)

Resumo

Com o avanço no desenvolvimento de trabalhos científicos na área de aprendizado de máquina, vemos que serviços suportados por essa tecnologia estão inseridos em vários contextos do cotidiano que buscam resolver problemas complexos como a tradução automática de idiomas, tema sobre o qual será abordado neste trabalho. A revisão bibliográfica esclarece os paradigmas voltados para a tarefa de tradução, além de formas de arquitetar e disponibilizar o serviço de Inteligência Artificial(IA). Nos passos metodológicos estão definidos as etapas a serem seguidas para a escolha do modelo tradutor, da infraestrutura do *software* e da codificação do sistema. Foi observado que para a tarefa de tradução, os modelos de redes neurais artificiais que utilizam o mecanismo de atenção são os que obtém melhores resultados, sendo o diferencial a capacidade de analisar o contexto da frase antes de decodificar o texto para o idioma de destino. O projeto foi implementado utilizando ferramentas que auxiliem na criação de um *serverless*, que possibilita embarcar soluções de IA sem a necessidade de gerenciar um servidor, diminuindo o tempo gasto no desenvolvimento. As soluções com suporte de IA e o uso do *serverless* proporcionam sistemas com potencial para se obter alta vantagem competitiva, pois de um lado é possível resolver problemas com alto grau de dificuldade, e de outro disponibilizá-los para os usuários com maior rapidez e escalabilidade nas demandas das requisições.

Palavras-chaves: *machine learning serverless. machine translator. neural machine translation. transformer. serverless.*

Abstract

With the advance in the development of scientific works in machine learning area, we receive that services supported by this technology are embedded in several everyday contexts that seek to solve complex problems such as automatic language translation , a theme that will be addressed in this work. The bibliographic review clarifies the paradigms aimed at the task of translation, as well as ways of designing and provide the Artificial Intelligence(AI) service. In the methodological steps are defined the stages to be followed for the choice of the translator model, the software infrastructure and system coding. It was observed that for the translation task, the models of artificial neural networks that use the attention mechanism are the ones that obtains better results, being the differential the ability to analyze the context of the phrase before decoding the text into the target language. The project was implemented utilizing tools that help in the creation of a serverless, that allows embark AI solutions without the need to manage a server, reducing the time spent in development. The solutions with AI support and the use of serverless provide systems with potential to obtain a high competitive advantage, because on the one hand it is possible to solve problems with a high degree of difficulty, and on the other hand to make them available to users with greater speed and scalability in the demands of requisitions.

Palavras-chaves: *machine learning serverless. machine translator. neural machine translation. transformer. serverless.*

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

NMT	Neural Machine Translation
AWS	Amazon Web Service
CLI	Command Line Interface
IaC	Infrastructure as Code
HCL	HashiCorp Configuration Language
BaaS	Back end as a service
FaaS	Function as a service
JVM	Java Virtual Machine
MLOps	Machine Learning Operations
IA	Inteligência Artificial
PLN	Processamento de Linguagem Natural

Sumário

1 Introdução

Este documento apresenta a consolidação do trabalho de conclusão de curso em Engenharia de Software. Ao longo das próximas páginas será apresentado o contexto em que este trabalho está inserido, dando destaque para temas como o desenvolvimento de sistemas de informação com suporte em IA, bem como as questões inerentes ao ciclo de criação de tais produtos, que redunda em sua operação em produção.

Depois terá uma revisão bibliográfica com o intuito de esclarecer conceitos e processos associadas ao desenvolvimento do *software* tradutor, passando por 3 temas principais: infraestrutura *serverless* (RAJAN, 2018), paradigmas de tradução automática (HUTCHINS, 1955) e *Machine Learning Operations (MLOps)* (ASHMORE; CALINESCU; PATERSON, 2021).

Além disso, também é demonstrado como a execução deste trabalho foi coordenada por meio dos métodos e materiais empregados.

Ademais, os resultados gerados pelo trabalho serão embasados na análise dos dados coletados pela pesquisa de literatura, caracterizando a solução desenvolvida, incluindo os detalhes da sua construção, da arquitetura e do fluxo inerente ao seu ciclo de MLOps.

Por fim, haverá algumas considerações finais à respeito da experiência de desenvolvimento do trabalho para conclusão de curso, além de revisitar de forma breve os resultados obtidos.

1.1 Contexto

Atualmente, algoritmos de IA vêm sendo utilizados com maior frequência nos ecossistemas empresariais e acadêmicos, muito pela eficiência dos novos métodos de aprendizado de máquina junto com o avanço das ferramentas de desenvolvimento, que facilitam a implementação do *software* e faz com que seja possível arquitetar soluções em diversas áreas, como os campos de visão computacional (SZELISKI, 2011) e *Processamento de Linguagem Natural (PLN)* (CHOWDHURY, 2003).

A IA envolve um agrupamento de conceitos e mecanismos matemáticos que buscam fazer com que as máquinas mimetizem capacidades humanas ligadas à inteligência, por exemplo, o raciocínio e a habilidade de análise para a tomada de decisão(MCCARTHY, 2004).

A aplicação de IA têm obtido resultados bem expressivos em vários contextos da área de PLN, sendo um deles a tradução, a qual as palavras são substituídas mecanica-

mente de um idioma para outro com o uso de um *software*. Assim, podendo ser descrita como um procedimento em que o significado do conteúdo em texto/áudio de origem é decodificado, em seguida é recodificado no idioma de destino (ARNOLD et al., 1993).

A tradução automática pode ser executada utilizando diferentes abordagens. O projeto deu maior ênfase no *Neural Machine Translation (NMT)*, que abrange os tipos de tradutores em que uma rede neural artificial (HOPFIELD, 1988) é usada para prever uma saída sequencial numérica utilizando uma outra sequência numérica como entrada (KOEHN, 2017).

Essa área de conhecimento (NMT) é o que possui maior assertividade na função de traduzir (BOJAR et al., 2016), principalmente por utilizar os fundamentos de atenção presentes nos transformadores, mecanismos que pesam a influência de diferentes partes dos dados de entrada, faz com que a sequência não precise ser processados em ordem e o contexto que confere significado a palavra seja identificado (VASWANI et al., 2017).

Após a investigações dos paradigmas de PLN para realizar o serviço de tradução, vem uma tarefa cada vez mais estratégica representada pela implantação de tais sistemas/modelos em produção. Já existem materiais que orientam este tipo de engenharia (PALEYES; URMA; LAWRENCE, 2021), que vão mais a fundo nas etapas de MLOps, que são os conjuntos de práticas que visam implantar e manter modelos de aprendizado de máquina de maneira confiável e eficiente (ASHMORE; CALINESCU; PATERSON, 2021).

Porém, quando se decide ir da teoria para a prática encontra-se uma outra barreira, a dificuldade de achar tutorias e documentações que englobam a ação de colocar um modelo de IA em produção. Isso ocorre nas etapas de utilização e integração do serviço na nuvem, da configuração da infraestrutura e das arquiteturas que facilitem essas etapas.

Assim como tem acontecido com a disseminação do uso de IA, algo parecido pode ser observado quanto ao uso de arquiteturas de *software* mais inovadoras. O uso da infraestrutura *serverless*(RAJAN, 2018) mostrou ser uma opção bastante atrativa para fazer parte da solução, principalmente pela necessidade de embarcar um modelo de aprendizado de máquina numa estrutura que ofereça redução dos custos, facilite o desenvolvimento e simplifique o processo de disponibilização do produto. Portanto, o detalhamento conceitual que envolve o *serverless* será feito na revisão bibliográfica.

Assim, quando ocorre a união das soluções com suporte de PLN e da infraestrutura *serverless*, nota-se que a IA viabiliza resultados para problemas não triviais e que o *serverless* facilita colocar o serviço em produção, diminuindo a utilização do número de tecnologias e das etapas de codificação. Entretanto, temos ai a combinação de duas perspectivas novas e que juntas fazem ser necessário uma investigação teórica aprofundada.

1.2 Objetivo Principal

O propósito principal deste trabalho é implementar uma solução que disponibilize uma Application Programming Interface (API) (SOUZA et al., 2004) de tradução em uma infraestrutura denominada *serverless*. A solução por ser muito extensa fez com que fosse necessário modularizar a proposta em subobjetivos, que serão apresentados no tópico 1.3

1.3 Objetivos Específicos

- É preciso fazer uma pesquisa nas áreas de tradutores automáticos, dos tipos de *serverless* e de como funciona o ciclo MLOps.
- Haver uma análise do que foi pesquisado para a criação do desenho arquitetural.
- Escolher a linguagem de programação, as bibliotecas e ferramentas de integração que são utilizadas para sistemas tradutores.
- Implementar o *software* e chegar numa análise do que foi planejado e executado.

1.4 Organização do Trabalho

Este trabalho está organizado nos seguintes capítulos:

- Capítulo 2 - Revisão Bibliográfica: tem como objetivo interpretar e analisar estudos relevantes a respeito de três tópicos: *serverless*, tradutor automático e MLOps.
- Capítulo 3 - Materiais e Métodos: apresenta o plano metodológico de forma mais detalhada e caracteriza o objeto de estudo.
- Capítulo 4 - Análise de Resultados: nesta seção, o que foi definido na metodologia será abordado de maneira aprofundada, mostrando como foi desenvolvido aquele tópico junto com a análise.

2 Revisão Bibliográfica

Neste capítulo, será feito uma revisão dos tópicos de tradução automática, *serverless* e ciclo MLOps. Com o objetivo de interpretar e analisar estudos e técnicas a respeito dos três assuntos que juntos formam a base arquitetural do trabalho proposto.

2.1 Tradutor Automático

A tarefa alvo escolhida deste trabalho, é a tradução automática de idioma. Em se tratando desse tema, a primeira atividade foi descobrir quais propostas técnicas apresentam melhores resultados. Os tradutores seguem um processo no qual as palavras são substituídas mecanicamente de um idioma para outro com o uso de um *software*. Primeiro o significado do dado de origem é decodificado, em seguida é re codificado no idioma de destino (ARNOLD et al., 1993).

No entanto, isso não é tão simples quanto parece, tornar frases e sentenças inteiras abrangentes no idioma a ser traduzido exige mais do que apenas substituir as palavras, porque nem todas as palavras têm uma equivalente em outro idioma e muitas delas têm mais de um significado.

A tradução feita por computador é uma tarefa relativamente antiga. A partir da década de 1970, surgiram projetos para este desafio. Até hoje, temos três principais abordagens:

- *Statistical Machine Translation (SMT)* (HUTCHINS, 1955): o SMT performa com eficiência em traduções básicas, porém sua desvantagem é que ele não leva em consideração o contexto, o que implica que a tradução pode normalmente estar errada.
- *Rule Based Machine Translation (RBMT)* (ARNOLD et al., 1993): As regras linguísticas podem ser escritas manualmente sendo aproveitadas na maioria dos domínios, só que é difícil lidar com interações das regras em grandes sistemas e expressões idiomáticas.
- *Neural Machine Translation (NMT)* (BISHOP, 2006): Utiliza redes neurais artificias para a tarefa de tradução.

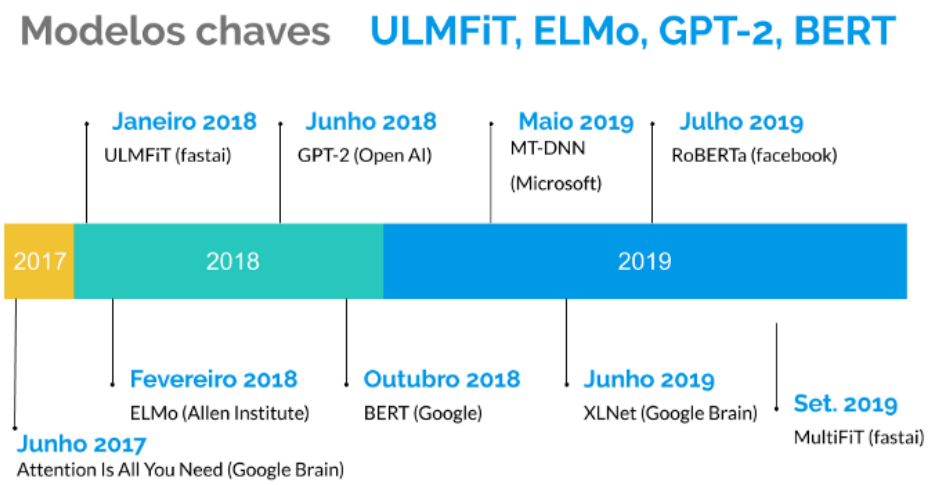
A SMT, com o advento dos modelos *International Business Machines (IBM)*, abriu caminho para propostas baseadas em frases e sintaxe. Esses métodos têm mostrado bastante progresso em muitos pares de idiomas e foram implantados com sucesso em sistemas de grande escala, como Google tradutor até 2014.

Em 2015, tivemos o primeiro experimento bem sucedido com aplicação de redes neurais (BISHOP, 2006) para a tarefa de tradução, representando o que há de mais evoluído nesta área, como verificado na campanha de avaliação de pesquisa *World Machine Translation (WMT)* (BOJAR et al., 2016), ocupando o lugar da frente nesta tarefa devido à sua facilidade de aprendizado e seu desempenho notável na tradução dos principais idiomas do mundo.

Os experimentos que tiveram início no ano de 2015, representaram uma verdadeira mudança de paradigma, uma vez que os resultados anteriores foram ultrapassados de maneira contundente. Atualmente, o NMT de última geração é baseado em um modelo codificador/decodificador sequencial com um mecanismo de atenção (VASWANI et al., 2017), tendo basicamente algumas variações na arquitetura da rede neural. Tanto é verdade que a área passou a ser tratada como NMT (STAHLBERG, 2020).

Depois de muitos anos com as pesquisas voltadas para outras abordagens de tradução, o cenário direcionou para as redes neurais por elas estarem se aprimorando, principalmente em 2017 e 2018, com a publicação do artigo acadêmico ULMFiT (HOWARD; RUDER, 2018), o qual a ideia era de treinar um modelo de classificação de textos com diferentes conjuntos de dados. Nessa época ainda, não era utilizado modelos baseados em atenção, mesmo assim conseguiram reduzir o erro para esta respectiva atividade em 18-24%. A partir deste momento, a IA aplicada ao PLN explodiu com o número de papéis científicos publicados sobre esse assunto. Todas as grandes empresas de tecnologia como Google e Facebook publicam modelos de *deep learning* (LECUN; BENGIO; HINTON, 2015) cada vez mais eficientes nas tarefas propostas.

Figura 1 – Modelos de Deep Learning



Fonte: "<https://medium.com/@pierre_guillou/processamento-de-linguagem-natural-pln-com-deep-learning-panorama-das-tend-40375c2303c3>"(2019)

A Figura 1 mostra uma linha do tempo com os principais modelos voltados para o PLN até o ano de 2019.

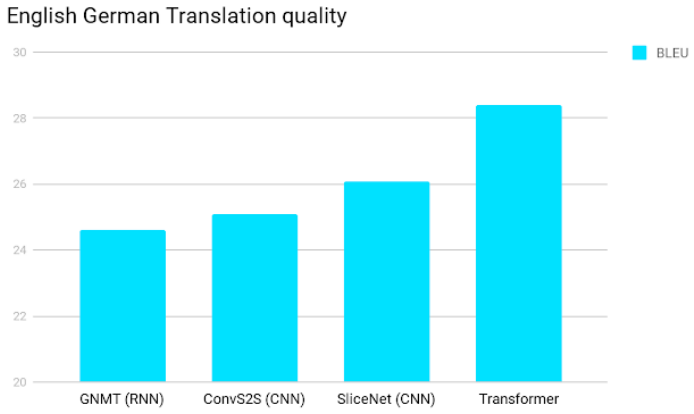
Google IA revelou uma nova arquitetura de rede neural chamada Transformador que lançou em 2017. A equipe afirmou que o Transformador funcionava melhor do que abordagens líderes, como redes neurais recorrentes e modelos convolucionais. De acordo com o artigo (VASWANI et al., 2017), o mecanismo de atenção considera a relação entre as palavras independentemente de onde elas são colocadas em uma frase, executando um número pequeno, mas constante de etapas escolhidas empiricamente. Em cada etapa, é aplicado uma relação entre todas as palavras da frase usando o mecanismo de auto atenção.

Até o ano de 2017, a maioria dos modelos de linguagem eram baseados em *Recurrent Neural Network (RNN)* (ZAREMBA; SUTSKEVER; VINYALS, 2015). RNN processa dados sequencialmente da esquerda para a direita ou da direita para a esquerda, palavra por palavra, até acessar a última célula. Entretando, o RNN não é muito eficiente no tratamento de longas sequências, pois ler uma palavra por vez gera várias etapas antes da tomada decisão, o modelo tende a esquecer o conteúdo de posições distantes.

O *Long Short Term Memory (LSTM)* (HOCHREITER; SCHMIDHUBER, 1997) oferece uma ligeira melhoria em relação ao RNN. O LSTM alavanca um mecanismo denominado *Gate* (GERS; SCHMIDHUBER; CUMMINS, 2000) para determinar quais informações a célula precisa lembrar e quais esquecer. Também, pode-se eliminar o problema de desaparecimento do gradiente. Porém, apesar das melhorias alcançadas pelo LSTM, não é suficiente quando comparado aos que usam o mecanismo de atenção.

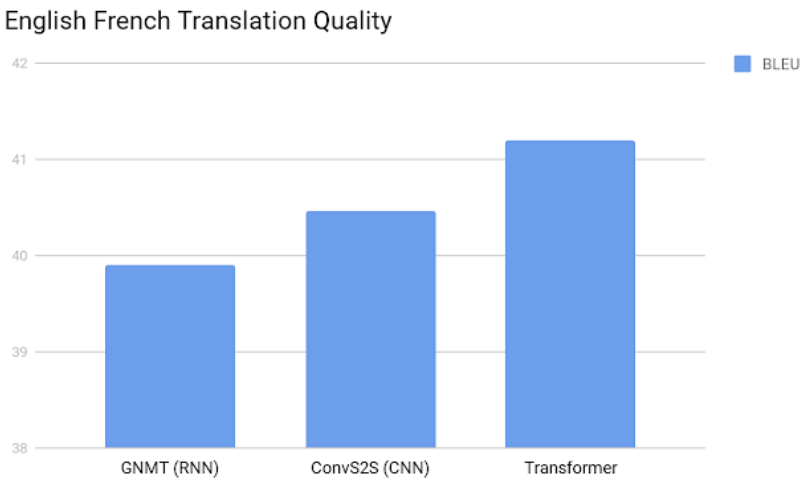
Na Figura 2 é mostrado que o transformador supera os modelos recorrente e convolucional na proposta de tradução acadêmica de inglês para alemão e inglês para francês. Dados produzidos na Conferência WMT (BOJAR et al., 2016).

Figura 2 – Inglês para o Alemão



Fonte:
"<https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html>"(2017)

Figura 3 – Inglês para o Francês



Fonte:
"<https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html>"(2017)

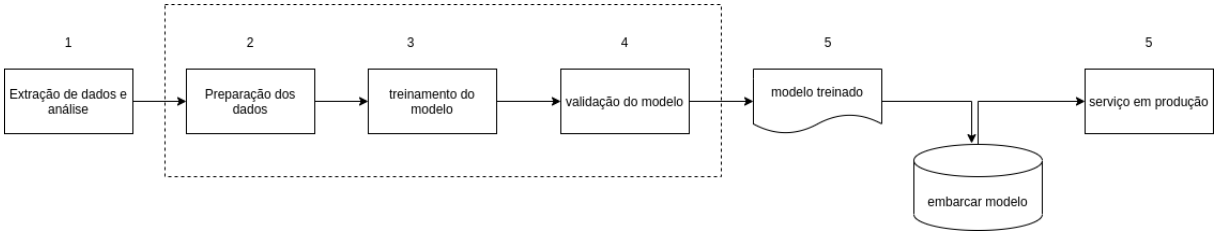
Agora que foi visto qual abordagem de PLN apresenta melhores resultados no contexto de tradução e qual mecanismo é o mais satisfatório dentro da área de *deep learning*, no capítulo de análise será apresentado os critérios de escolha do modelo tradutor.

2.2 MLOps

O MLOps é uma cultura e prática na engenharia de aprendizado de máquina que visa unificar o desenvolvimento e a operação de sistemas, permitindo a colaboração entre as equipes a fim de manter o controle sobre diferentes versões de modelos, controle de experimentos de um mesmo problema, monitoramento, dentre outros. A prática de MLOps significa que o projeto visa a automação e o monitoramento de todos os passos da construção do sistema, inclusive integração, teste, lançamento, implantação e gerenciamento de infraestrutura (ASHMORE; CALINESCU; PATERSON, 2021).

Além do desafio de criar um modelo, é necessário elaborar um sistema de aprendizado de máquina integrado e operá-lo continuamente em produção, para isso é necessário unificar três áreas no processo, a engenharia de dados, a criação e validação do modelo e embarcar esse serviço numa infraestrutura para produção. Esses passos e o aprofundamento deles podem ser checados na Figura 4 e nos passos abaixo.

Figura 4 – MLOps Pipeline



Fonte: Elaborada pelo autor.

- 1) Extração de dados: seleciona e integra os dados relevantes de várias fontes que serão utilizados para o treinamento do modelo, logo após, realiza-se uma análise de dados exploratória para compreender os materiais disponíveis para a criação do modelo.

- 2) Preparação de dados: isso envolve a limpeza dos dados, divisão das bases em treinamento, validação e teste. Também é possível aplicar transformações de dados e engenharia de atributos ao modelo.

- 3) Treinamento de modelo: implementação de diferentes algoritmos com os dados já preparados para o treino dos modelos de *deep learning*. Além disso, os algoritmos implementados ficam sujeitos ao ajuste de hiper parâmetros para atingir o melhor desempenho.

- 4) Avaliação do modelo: a avaliação é feita com dados de teste que o algoritmo jamais teve contato, a saída deste passo é um conjunto de métricas utilizadas para interpretar a eficácia do modelo. A validação também acontece nesta etapa, confirmando que o modelo é adequado e não será preciso nenhum ajuste no momento da implantação.

- 5) Exibição do modelo: o modelo validado é implantado em um ambiente de destino para exibir previsões. No caso deste trabalho, uma API.

Além disso, existem ferramentas que auxiliam no controle desse fluxo, entre eles estão:

- mlflow: uma plataforma de código aberta para gerenciar o ciclo de vida dos modelos de aprendizado de máquina. Facilita as etapas de treinamento, rastreamento e produção. O *software* foi organizado para funcionar com as bibliotecas, linguagens e estruturas mais recentes e utilizadas no mercado.

- metaflow: tem o objetivo similar ao mlflow,trabalha com qualquer biblioteca de aprendizado de máquina e se integra aos serviços da *Amazon Web Service (AWS)*.

O que foi visto acima, é o fluxo padrão de MLOps, como cada projeto tem necessidades próprias, na parte de análise será mostrado como esse ciclo foi adaptado, incluindo as tecnologias que possibilitam essas modificações.

2.3 Serverless

A computação em nuvem surgiu após o início da virtualização das infraestruturas de *software* e *hardware* e ganhou importância na última década como um campo novo e empolgante devido à sua grande influência na redução de custos, diminuição da latência, melhoria da escalabilidade e diminuição do gerenciamento do lado do servidor.(JONAS et al., 2019)

Os autores em (JONAS et al., 2019), abordaram vários desafios ao gerenciar um ambiente de nuvem por um usuário, como disponibilidade, balanceamento de carga, escalonamento automático, segurança e monitoramento.

Esses desafios levaram à introdução de outro modelo de computação em nuvem, que é chamado de *serverless* (JONAS et al., 2019), esse se diferencia da nuvem tradicional por não ser uma arquitetura monolítica, sendo orientada por eventos, logo cada componente lida com diferentes partes de dados e é executado de forma independente (PARRES-PEREDO; PIZA-DAVILA; CERVANTES, 2019). O serviço *serverless* abrange duas áreas, o *Backend as a Service (BaaS)* e o *Function as a Service (FaaS)*. O BaaS tem como característica descrever aplicativos que incorporam de forma significativa, ou total, aplicações e serviços hospedados em nuvem de terceiros para gerenciar a lógica e o estado do servidor.

Já o FaaS fornece uma plataforma que permite aos clientes desenvolverem, executarem e gerenciarem funcionalidades de aplicativos, sem a complexidade de construir e manter a infraestrutura normalmente associada ao desenvolvimento e lançamento de um sistema. É executado em contêineres de computação, sem estado, comumente conhecidos como funções orientadas a eventos (RESEARCH. . . ,).

Em vez de ter um servidor sempre em execução, os desenvolvedores implantam seus aplicativos na nuvem como funções, em seguida o provedor de nuvem assume a responsabilidade de gerenciar, dimensionar e fornecer diferentes recursos para garantir o bom funcionamento dessas funções.

3 Materiais e Métodos

Neste capítulo, será definido os métodos que foram executados para o trabalho de conclusão de curso, junto com a descrição dos materiais utilizados.

3.1 Métodos

A enumeração dos passos metodológicos foram divididos em duas partes. A primeira se refere ao trabalho de conclusão de curso 1, com o intuito de aprofundar nas pesquisas em IA, mais especificamente em *deep learning*, analisando arquiteturas e processos que permitem colocar este tipo de serviço em produção.

A segunda parte, refere-se ao trabalho de conclusão de curso 2, com o foco maior no desenvolvimento do sistema, onde todo o material pesquisado anteriormente será utilizado como base para a criação do tradutor.

1. Investigar e escolher o modelo de aprendizado de máquina para tradução automática de texto.(1)
2. Definir o *pipeline* MLOps.(1)
3. Definir a arquitetura *serverless* e as tecnologias.(1)
4. Implementar a *framework serverless* e embarcar o modelo tradutor no serviço.(2)
5. Construir o componente *backend* do tradutor.(2)
6. Montar o arquivo Infrastructure as Code (IaF) (HÜTTERMANN, 2012) da plataforma nuvem escolhida, disponibilizando a API pro ambiente de produção.(2)

3.2 Ferramentas e tecnologias utilizadas

- *Serverless Framework*: ajuda no desenvolvimento e disponibilização do tradutor no *AWS Lambda Function*, sendo um CLI que oferece uma estrutura de automação dos processos do ciclo de vida do *serveless*.
- AWS *Lambda*: é o *serveless* onde o código roda, não necessita administrar o servidor, é uma ferramenta escalável e executa seu código somente quando demandado.
- Efsync: sincroniza os arquivos e dependências no *AWS-EFS filesystem*.
- Terraform: ferramenta IaC que permite o uso do HCL para descrever e construir infraestrutura nativas, seguras e eficientes.

- AWS-EFS: é um serviço totalmente gerenciado que facilita a configuração, a escalabilidade e a otimização dos custos de armazenamento de arquivos na *Amazon Cloud*.
- *HuggingFace*: é uma biblioteca que oferece o estado da arte das implementações arquiteturais relacionadas ao PLN.
- *Python*: é uma linguagem de programação de alto nível, interpretada de *script*, imperativa, orientada a objetos, de tipagem dinâmica e forte.

4 Análise

Este capítulo tem como finalidade elucidar os tópicos citados na metodologia e revisão bibliográfica, detalhando os passos do desenvolvimento do projeto, decisões arquiteturais e ciclo de vida do sistema tradutor, além de explicar os arquivos de código.

4.1 Escolha do modelo tradutor

Com o aprofundamento teórico feito na seção 2.1, é visto que nos últimos anos, especialmente do ano de 2017 em diante, os modelos de *deep learning* com o mecanismo de atenção são os que produzem resultados mais satisfatórios para tradução.

Porém, é necessário que haja uma investigação mais a fundo na área de NMT, pois já existem alguns modelos tradutores disponibilizados por empresas, sendo preciso fazer uma seleção desses exemplares.

Foram elicitados 3 plataformas que oferecem modelos transformadores para tradução, eles estão listados abaixo:

- *HuggingFace*.
- *Amazon Lex*.
- *Watson Language Translator*.

Os critérios de escolha da plataforma que oferece o modelo tradutor são:

- C1. Oferecer diferentes implementações arquiteturais para tradutores.
- C2. Documentação da arquitetura neural.
- C3. Serviço oferecido gratuitamente.
- C4. Possibilidade de acessar e modificar as arquiteturas oferecidas.
- C5. Integração com projetos em diferentes linguagens de programação.

Tabela de comparação:

Tabela 1 – Plataformas de Modelos Transformadores

Plataforma	C1	C2	C3	C4	C5
HuggingFace	X	X	X	X	X
Amazon Lex					X
Watson Language Translator					X

A tabela de comparação mostra que o *HuggingFace* supre todos os critérios e melhor se encaixa como parte da solução, sendo uma ferramenta *open source*, que oferece à comunidade de forma gratuita modelos pré-treinados que implementam o estado da arte dos transformadores relacionados ao PLN (WOLF et al., 2020), com mais de 1000 exemplos disponíveis em diversas áreas.

O *HuggingFace* diferencia-se das outras 2 opções pelas regras de negócio, basicamente os outros serviços levantados já vem com o tradutor pronto, tirando a autonomia do desenvolvedor de escolher e versionar modelos já criados, além de não ser gratuito o uso.

Agora, chegamos na etapa de coleta dos modelos oferecidos pelo *HuggingFace*, para isso foi utilizado uma *string* de busca "*Neural Machine Translation*", cujos exemplos encontrados estão listados abaixo.

- *BertGeneration*, modelo feito para tarefas de sequência a sequência usando as etapas de codificação e re codificação, como proposto no artigo (ROTHE; NARAYAN; SEVERYN, 2020).
- *Mt5* é uma variante multilíngue do T5, com um desempenho de última geração em muitos *benchmarks* de PLN (XUE et al., 2021).
- *FSTM* é um modelo disponibilizado pelo empresa Facebook para atuar em dois pares de idiomas, inglês para alemão e inglês para russo (NG et al., 2019).
- *MBart-50* é uma versão melhorada do MBart-25, criado pelo Facebook IA e faz a tradução de 50 idiomas (LIU et al., 2020).

Além disso, os critérios de seleção para essa etapa são:

- C1. Modelo que seja específico para tradução e facilita a implementação.
- C2. Modelo que já venha pré-treinado, faz com que não seja necessário treinar com outros dados.

- C3. Modelo com documentação teórica e prática de como aplicá-lo.
- C4. Utilize transformadores.

Tabela 2 – Modelos de NMT

Modelos	C1	C2	C3	C4
BertGeneration		X	X	X
Mt5 X		X	X	X
FSTM	X	X	X	X
MBart-50 X	X	X	X	X

Os que melhor encaixaram nos requisitos de seleção acima foram a MBart-50 e a FSTM, porém a Mbart-50 oferece tradução para mais pares de idiomas, implementando uma arquitetura padrão sequência para sequência com um codificador bidirecional (LIU et al., 2020).

4.2 Definição da Pipeline MLOps

A Seção 4.1 mostra que a plataforma *HuggingFace* fornece o modelo tradutor já pré-treinado, isso impacta no ciclo geral de MLOps fazendo com que seja necessário alguns ajustes nas etapas iniciais.

Por isso, as ferramentas mlflow e metaflow citadas na Seção 2.2, que dão suporte na parte de extração de dados até a validação do modelo, mas como essas fases já estão supridas pelo *HuggingFace*, é hora de pensar na tecnologia que auxilie em embarcar o modelo na nuvem, que seria justamente a etapa 5 da Figura 4.

A problemática que envolve embarcar um modelo de NMT, vem da dificuldade de armazenar os arquivos, isso ocorre nas principais empresas que oferecem o serviço *serverless*, como a *Amazon Web Service (AWS)*, *Google Cloud Functions* e *Azure Functions*. Por exemplo, o Tensorflow e o Pytorch, que são bibliotecas de código aberto para aprendizado de máquina, necessitam de espaço de memória para armazenagem, além de uma máquina que processe e carregue os modelos de NMT.

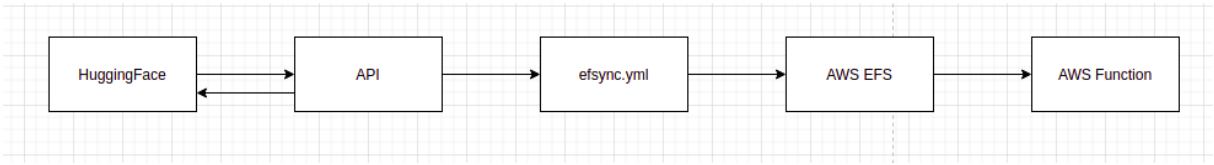
Antes de procurar a ferramenta que integra as dependências do projeto, primeiro é preciso escolher a plataforma *serverless*, a Seção 4.3 demonstra o processo de seleção da AWS.

Analisando a arquitetura do serviço representada na Figura 6, a tecnologia de integração tem de atuar em carregar os arquivos na *Amazon Elastic File System (AWS EFS)*.

Foi encontrado um *Command Line Interface(CLI)* chamado *efsync*, que sincroniza os arquivos locais ou do S3 automaticamente na AWS EFS, permitindo que seja instalado dependências diretamente no sistema de arquivos EFS e usado posteriormente na função *AWS Lambda*.

O processo aprimorado de MLOps está representado na Figura 5, primeiro um *script* que está dentro da API de tradução é executado para carregar o modelo do *HuggingFace*, depois o arquivo *efsync* integra as dependências do projeto na *AWS EFS*, por fim, a *AWS Function*, que armazena a lógica da rota de tradução, utiliza o que está na AWS EFS.

Figura 5 – MLOps Adaptado



Fonte: Elaborada pelo autor.

4.3 Arquitetura Serverless

Após a revisão de litetatura feita sobre o *serverless*, sabe-se que ela é uma ramificação da computação em nuvem e que se divide em FaaS e BaaS. Fazendo uma comparação do serviço tradutor que é implementado neste trabalho, verifica-se que o projeto se encaixa no molde FaaS, pois a ideia é fazer um microsserviço, uma funcionalidade que responda a eventos e que possa virar um módulo de um *software* maior caso desejado.

Por isso, é necessário uma pesquisa para a escolha da empresa que oferecerá o FaaS, sendo elas:

- A *AWS Lambda*, um serviço FaaS fornecido pela AWS.
- O FaaS da *Azure Cloud*, fornecida pela Microsoft.
- *Google Cloud Functions*, fornecido pela Google.

Logo em seguida, são criado os critérios de escolha do FaaS, que são:

- C1. Gratuidade até 10000 requisições no mês, que é o máximo de demanda prevista para ser suportada pelo tradutor.

- C2. Suportar que a lógica da rota seja implementada em *Python*.
- C3. Não ser necessário gerenciar e nem ter conhecimento em servidores para a implementação.

Tabela 3 – Comparação das plataformas Serverless

Plataformas	C1	C2	C3
AWS Lambda	X	X	X
Azure Cloud	X	X	
Google Cloud Functions	X	X	X

Como podemos ver na Tabela 3, as plataformas possuem vantagens bem parecidas suprindo quase todos os critérios de aceitação. Dependendo da demanda de requisições e do contexto do *software*, qualquer uma dessas opções será válida, sendo o *Google Cloud Functions* e *AWS Lambda* os que se encaixam melhor.

O FaaS da AWS foi escolhido por uma maturidade maior do autor em operar nesta plataforma. Notou-se que o produto não sofreria grande impacto nas fases de desenvolvimento e integração.

Partindo da escolha do FaaS, temos uma representação arquitetural na Figura 6 da *AWS Lambda*.

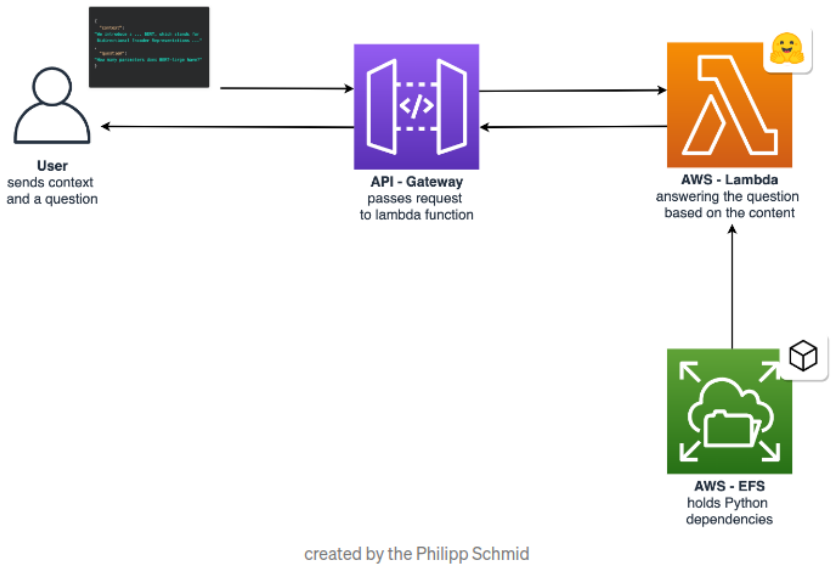


Figura 6 – Arquitetura da aplicação

O usuário acessa a rota de tradução, com isso o evento é gerado e a API *Gateway* direciona a requisição para a *AWS Lambda*, que carrega as dependências contidas na *AWS-EFS* e executa a lógica do modelo tradutor. No final, o retorno para o *front-end* será o texto traduzido ou um erro orientando o usuário do que pode ter ocorrido.

Com o escopo e escolhas arquiteturais maduras, é hora de apresentar os diagramas e requisitos do projeto.

4.3.1 Metas e Restrições de Arquitetura

Restrições	Descrição
Portabilidade	Por ser um FaaS voltado para receber requisições de um *front-end web* ou *mobile*, é necessário conexão com a internet e deve estar apta para uso nos navegadores Mozilla Firefox, Google Chrome, Safari, etc.
Distribuição	A distribuição ocorre assim que a nova versão do software é enviada para produção. O usuário sempre acessa a última versão disponível.

4.3.2 Diagrama de Sequência

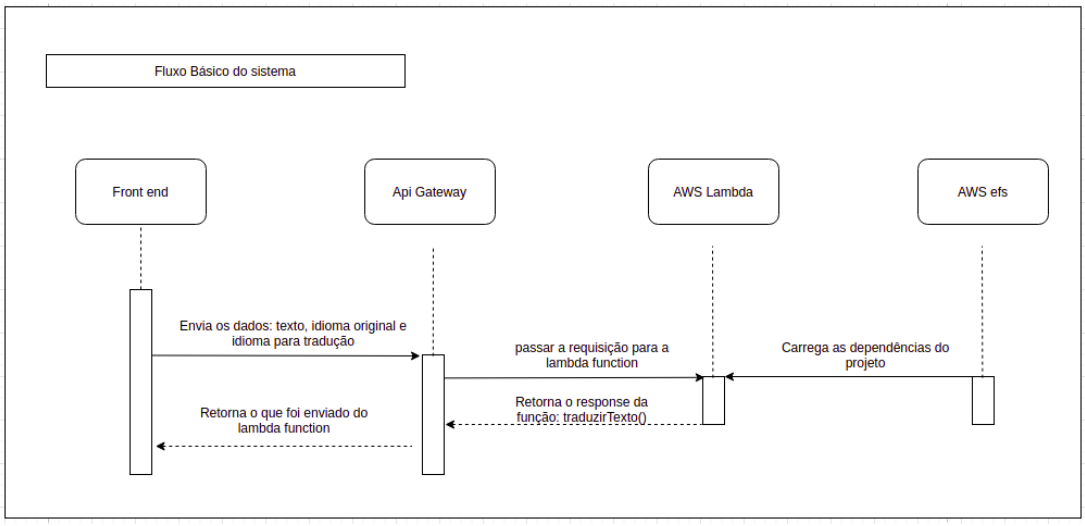


Figura 7 – Fluxo da Aplicação

4.3.3 Qualidade

Os seguintes itens conferem ao sistema aspectos de qualidade, bem como a descrição da abordagem realizada para satisfazer esses aspectos. São estes aspectos legais e reguladores, como as normas estabelecidas para bom funcionamento do sistema, requisitos do projeto, padrões de confiabilidade e desempenho.

4.3.4 Requisitos Funcionais

RF01 - Traduzir o texto enviado pela requisição com o idioma escolhido pelo usuário.

4.3.5 Requisitos Não Funcionais

Os requisitos não funcionais são divididos principalmente em dois tópicos:

- RNF01 - Confiabilidade.

Disponibilidade: O sistema estará disponível no modo 24/7 (24 horas por dia, 7 dias por semana).

Suportabilidade: O sistema deverá suportar uma demanda até 50 requisições simultâneas sem perder desempenho no processamento.

- RNF02 -Desempenho.

Tempo de Resposta: O *serverless* tem de responder às requisições do *front-end* em um tempo limite de 2 segundos.

4.4 Embarcar Modelo de Deep Learning

Para embarcar o modelo de *deep learning*, é necessário que primeiro seja criado um arquivo *main.tf* para definir a infraestrutura, para isso o Terraform é utilizado, nele precisamos colocar o provedor da nuvem junto com as credenciais, um sistema de arquivos AWS EFS, um ponto de acesso e um destino de montagem para poder usá-lo na função *AWS Lambda*.

Figura 8 – Arquivo *main.tf*

```
# provider
provider "aws" {
  region                  = "eu-central-1"
  shared_credentials_file = "~/.aws/credentials"
  profile                 = "adm_serveless"
}

# get all available availability zones

data "aws_vpc" "default" {
  default = true
}

data "aws_subnet_ids" "subnets" {
  vpc_id = data.aws_vpc.default.id
}

# EFS File System

resource "aws_efs_file_system" "efs" {
  creation_token = "MyFS"
}

# Access Point

resource "aws_efs_access_point" "access_point" {
  # file_system_id = "fs-17a91b4c"
  file_system_id = aws_efs_file_system.efs.id
}

# Mount Targets

resource "aws_efs_mount_target" "efs_targets" {
  for_each = data.aws_subnet_ids.subnets.ids
  subnet_id = each.value
  # file_system_id = "fs-17a91b4c"
  file_system_id = aws_efs_file_system.efs.id
}

#
# SSM Parameter for serverless
#

resource "aws_ssm_parameter" "efs_access_point" {
  name      = "/efs/accessPoint/id"
  type      = "String"
  # value     = "fsap-0af758658b12d8af5"
  value     = aws_efs_access_point.access_point.id
  overwrite = true
}
```

Fonte: Elaborado pelo autor.

Após a etapa anterior, a AWS EFS está pronta para receber as dependências que o *serverless* necessita, por isso foi criado um outro arquivo chamado efsync.yaml, que utiliza o CLI Efsync e nele são estruturados os requisitos do projeto, os dados da AWS EFS e do perfil da *Amazon*. Boa parte das informações presentes neste arquivo, foram consultadas utilizando o CLI da Amazon.

Figura 9 – Arquivo *efsync.yaml*

```
#standard configuration
efs_filesystem_id: fs-58a61903 # aws efs filesystem id
subnet_Id: subnet-3582cb5f # subnet of which the efs is running in
ec2_key_name: efsync-asd913fjgq3 # required key name for starting the ec2 instance
clean_efs: all # Defines if the EFS should be cleaned up before. values: `'all'`,`'pip'`,`'file'` uploading
# aws profile configuration
aws_profile: adm_serveless  # aws iam profile with required permission configured in .aws/credentials
aws_region: eu-central-1 # the aws region where the efs is running

# pip dependencies configurations
efs_pip_dir: lib # pip directory on ec2
python_version: 3.8 # python version used for installing pip dependencies -> should be used as lambda runtime afterwads
requirements: requirements.txt # path + file to requirements.txt which holds the installable pip dependencies
```

Fonte: Elaborado pelo autor.

Agora só falta criar o *script* para carregar o modelo pré-treinado do *Huggingface* na aplicação, isso está presente no arquivo get_model.py, detalhado na Figura 10. Além disso, à biblioteca *transformers* precisa ser colocada no requirements.txt.

Figura 10 – Arquivo que carrega o MBart-50

```
from transformers import MBartForConditionalGeneration, MBart50TokenizerFast

def get_model():
  """Loads model from Hugginface model hub"""
  try:
    model = MBartForConditionalGeneration.from_pretrained("facebook/mbart-large-50-many-to-many-mmt", use_cdn=True)
    model.save_pretrained('./model')
  except Exception as e:
    raise(e)

def get_tokenizer():
  """Loads tokenizer from Hugginface model hub"""
  try:
    tokenizer = MBart50TokenizerFast.from_pretrained("facebook/mbart-large-50-many-to-many-mmt", src_lang="en_XX")
    tokenizer.save_pretrained('./model')
  except Exception as e:
    raise(e)

get_model()
get_tokenizer()
```

Fonte: Elaborado pelo autor.

4.5 Back-End

Nesta parte do desenvolvimento é criado os passos para pegar a frase a ser traduzida junto com o idioma de destino. Sendo a rota acionada por um evento do tipo POST, faz com que seja pego dados do *front-end* e posteriormente esses dados passem para a função de tradução, na Figura 11 esse método se chama *translation_pipeline*.

Explicando mais a fundo o método tradutor, ele é do tipo 1:n, ou seja, o idioma base é o português e o usuário escolhe o idioma destino, que é uma das 50 opções listadas abaixo:

Tabela 4 – Idiomas Suportados

Árabe	Tcheco	Alemão	Inglês	Espanhol
*E*stoniano	Finlandês	Francês	Gujarati	Hindi
*I*taliano	Japonês	Cazaque	Coreano	Lituano
*L*etão	Birmanês	Nepalês	Holandês	Romeno
*R*usso	Sinhala	Turco	Vietnamita	Chinês
*A*frikaans	Azerbaijani	Bengali	Persa	Hebraico
*C*roata	Indonésio	Georgiano	Khmer	Macedônio
*M*alayalam	Mongol	Marathi	Polonês	Pashto
*E*sloveno	Sueco	Suaíli	Tâmil	Telugu
*T*ailandês	Tagalo	Ucraniano	Urdu	Galego

Abaixo temos as figuras com a implementação da lógica do *back-end*.

Figura 11 – Handler do Back-end

```
# initializes the pipeline
translation_pipeline = serverless_pipeline()

def handler(event, text):
    try:
        # loads the incoming event into a dictonary
        body = json.loads(event['body'])
        # uses the pipeline to translate the text
        translation =translation_pipeline(language_target=body['language_target'], text=body['text'])
        return {
            "statusCode": 200,
            "headers": {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*',
                "Access-Control-Allow-Credentials": True

            },
            "body": json.dumps({'answer': translation})
        }
    except Exception as e:
        print(repr(e))
        return {
            "statusCode": 500,
            "headers": {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*',
                "Access-Control-Allow-Credentials": True
            },
            "body": json.dumps({"error": repr(e)})
        }
```

Fonte: Elaborado pelo autor.

Figura 12 – Função tradutora

```
def serverless_pipeline(model_path='./model'):
    """Initializes the model and tokenzier and returns a predict function that can be used as pipeline"""
    tokenizer = MBart50TokenizerFast.from_pretrained(model_path)
    model = MBartForConditionalGeneration.from_pretrained(model_path)
    def predict(text, language_target):
        """predicts the translation text on an given language target."""
        model_inputs = tokenizer(text, return_tensors = "pt")
        generated_tokens = model.generate(**model_inputs, forced_bos_token_id = tokenizer.lang_code_to_id [language_target])
        translation = tokenizer.batch_decode(generated_tokens, skip_special_tokens=True)
        return translation
    return predict
```

Fonte: Elaborado pelo autor.

4.6 Infrastructure as Code

Por fim, agora temos a configuração do último arquivo antes do *deploy*, nele o objetivo é definir os detalhes de cada parte do serviço, conectando com o ponto de acesso da AWS EFS, onde estão as dependências do projeto. Também é mostrado como ocorre o acionamento da função, e depois coloca-se os dados do provedor da nuvem, tudo isso pode ser visto na Figura 13.

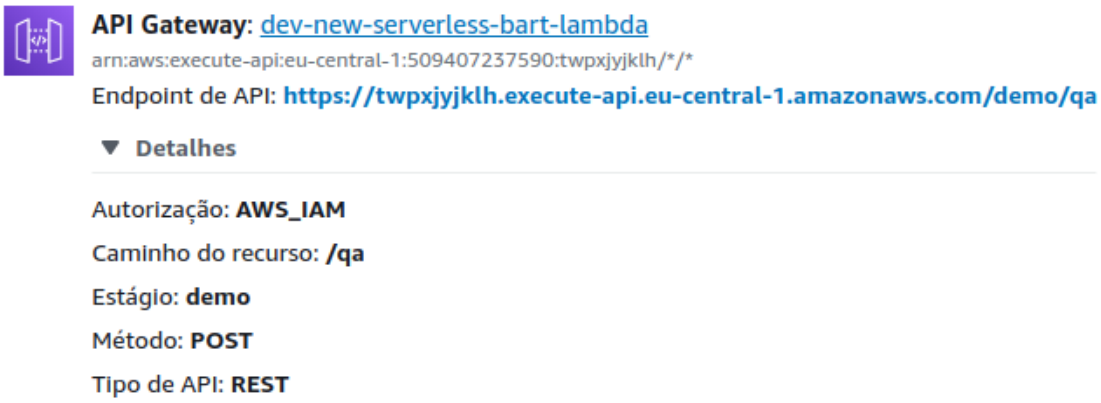
Figura 13 – Estrutura do Serviço

```yaml
service: new-serverless-bart-lambda

plugins:
  - serverless-pseudo-parameters

custom:
  efsAccessPoint: ${ssm:/efs/accessPoint/id}
  LocalMountPath: /mnt/efs
  efs_pip_path: /lib

provider:
  name: aws
  runtime: python3.8
  region: eu-central-1
  memorySize: 3008 # optional, in MB, default is 1024
  timeout: 60 # optional, in seconds, default is 6
  environment: # Service wide environment variables
    MNT_DIR: ${self:custom.LocalMountPath}
    EFS_PIP_PATH: '${self:custom.LocalMountPath}${self:custom.efs_pip_path}'
  iamManagedPolicies:
  # iam.role.managedPolicies:
    - arn:aws:iam::aws:policy/AmazonElasticFileSystemClientReadWriteAccess

package:
  exclude:
    - test/**
    - lib/**
    - terraform/**
    - node_modules/**
    - .vscode/**
    - .serverless/**
    - .pytest_cache/**
    - __pychache__/**

functions:
  translatingtext:
    handler: handler.handler
    fileSystemConfig:
      localMountPath: ${self:custom.LocalMountPath}
      arn: 'arn:aws:elasticfilesystem:${self:provider.region}:#{AWS::AccountId}:access-point/${self:custom.efsAccessPoint}'
    vpc:
      securityGroupIds:
        - <your-default-security-group-id>
      subnetIds:
        - <your-default-subnet-id>
        - <your-default-subnet-id>
        - <your-default-subnet-id>
    events:
      - http:
          path: qa
          method: post
```

Fonte: Elaborado pelo autor.

Após o *deploy* é possível ver no perfil da AWS que a função *Lambda* e o arquivo de dependências AWS EFS foram criados.

Figura 14 – Dados AWS EFS



Fonte: Elaborado pelo autor.

Figura 15 – Dados AWS *Lambda*

	Nome	ID do sistema de arquivos	Criptografado	Tamanho total	Tamanho na categoria Standard/One Zone	Tamanho na categoria Standard-IA/One Zone-IA	Taxa de transferi provisio (MiB/s)
○	MyFS	fs-7aa7d1ce	Criptografado	6.00 KiB	6.00 KiB	0 bytes	-

Fonte: Elaborado pelo autor.

5 Conclusão

Com as etapas seguidas neste trabalho de conclusão de curso, desde a revisão bibliográfica até a implementação do tradutor, percebeu-se que na última década o aprendizado de máquina evoluiu exponencialmente, renovando os antigos métodos usados no PLN. Fazendo com que fosse disponibilizado modelos de *deep learning* que conseguem melhores resultados na tarefa de tradução, além das bibliotecas e ferramentas que diminuem as etapas de MLOps, como as fases de pré-treinamento e validação do modelo matemático transformador.

Juntando isso ao conhecimento de *serverless*, e sabendo criar e manutenir a infraestrutura na plataforma AWS, o sistema é disponibilizado seguindo processos reduzidos e simplificados, que podem ser definidos na seguinte ordem: primeiro, carregar o modelo de aprendizado de máquina; segundo, construir a lógica da rota que faz a tradução, e por último, tendo os arquivos de infraestrutura criados é necessário somente fazer o *deploy* do código na AWS.

O tradutor produzido é acionado por eventos, especificamente por requisições do tipo POST, elaborado para funcionar como um microsserviço. Oferece suporte para os 50 idiomas propostos pelo modelo MBart50, sendo a linguagem base o português.

Porém, para chegar ao produto final desse trabalho, ocorreram dificuldades, principalmente na fase de configuração da infraestrutura com a utilização do Terraform, pois foi necessário encontrar uma ferramenta que conectasse as dependências do projeto, que incluem as bibliotecas e o modelo tradutor com a função Lambda.

Durante o período de desenvolvimento, com a elaboração do fluxo de usuário, notou-se uma demanda que poderia gerar uma melhoria futura, que é justamente ser enviado na requisição do *front-end* o modelo tradutor a ser utilizado, ou seja, ao invés de carregar somente um modelo, o sistema disponibilizaria nas dependências do projeto diversos modelos de tradução. Isso acarretaria na refatoração dos critérios de escolha do modelo de *deep learning*, no desenho arquitetural, este último incluindo modificar os *scripts* e a lógica do *back-end*.

Com a finalização do tradutor, constatou-se que o serviço oferece suporte pros 50 idiomas elicitados anteriormente, além de estar disponível para os clientes *front-end*.

Referências

ARNOLD, D. et al. *Machine Translation: an Introductory Guide*. London: Blackwells-NCC, 1993. Disponível em: <http://www.essex.ac.uk/linguistics/clmt/MTbook/>. Citado 2 vezes nas páginas 20 e 23.

ASHMORE, R.; CALINESCU, R.; PATERSON, C. Assuring the machine learning lifecycle: Desiderata, methods, and challenges. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 54, n. 5, maio 2021. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3453444>. Citado 3 vezes nas páginas 19, 20 e 26.

BISHOP, C. M. *Pattern Recognition and Machine Learning*. [S.l.]: Springer, 2006. Citado 2 vezes nas páginas 23 e 24.

BOJAR, O. et al. *Results of the WMT16 Metrics Shared Task*. Berlin, Germany: Association for Computational Linguistics, 2016. 199–231 p. Disponível em: <https://aclanthology.org/W16-2302>. Citado 3 vezes nas páginas 20, 24 e 25.

CHOWDHURY, G. G. Natural language processing. *Annual Review of Information Science and Technology*, v. 37, n. 1, p. 51–89, 2003. Disponível em: <https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/aris.1440370103>. Citado na página 19.

GERS, F. A.; SCHMIDHUBER, J.; CUMMINS, F. Learning to forget: Continual prediction with lstm. *Neural Computation*, v. 12, n. 10, p. 2451–2471, 2000. Citado na página 25.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural Computation*, v. 9, n. 8, p. 1735–1780, 1997. Citado na página 25.

HOPFIELD, J. Artificial neural networks. *IEEE Circuits and Devices Magazine*, v. 4, n. 5, p. 3–10, 1988. Citado na página 20.

HOWARD, J.; RUDER, S. *Universal Language Model Fine-tuning for Text Classification*. 2018. Citado na página 24.

HUTCHINS, W. J. *Chapter 6 Machine translation: History of Research and Applications*. 1955. Citado 2 vezes nas páginas 19 e 23.

HÜTTERMANN, M. *Infrastructure as Code*. Berkeley, CA: Apress, 2012. 135–156 p. ISBN 978-1-4302-4570-4. Disponível em: <https://doi.org/10.1007/978-1-4302-4570-4_9>. Citado na página 29.

JONAS, E. et al. Cloud programming simplified: A berkeley view on serverless computing. *CoRR*, abs/1902.03383, 2019. Disponível em: <http://arxiv.org/abs/1902.03383>. Citado na página 28.

KOEHN, P. *Neural Machine Translation*. 2017. Citado na página 20.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *Nature*, v. 521, n. 7553, p. 436–444, 2015. Disponível em: <https://doi.org/10.1038/nature14539>. Citado na página 24.

LIU, Y. et al. Multilingual Denoising Pre-training for Neural Machine Translation. *Transactions of the Association for Computational Linguistics*, v. 8, p. 726–742, 11 2020. ISSN 2307-387X. Disponível em: <https://doi.org/10.1162/tacl_a_00343>. Citado 2 vezes nas páginas 32 e 33.

MCCARTHY, J. What is artificial intelligence? 2004. Citado na página 19.

NG, N. et al. *Facebook FAIR's WMT19 News Translation Task Submission*. 2019. Citado na página 32.

PALEYES, A.; URMA, R.-G.; LAWRENCE, N. D. *Challenges in Deploying Machine Learning: a Survey of Case Studies*. 2021. Citado na página 20.

PARRES-PEREDO, A.; PIZA-DAVILA, I.; CERVANTES, F. Building and evaluating user network profiles for cybersecurity using serverless architecture. In: *2019 42nd International Conference on Telecommunications and Signal Processing (TSP)*. [S.l.: s.n.], 2019. p. 164–167. Citado na página 28.

RAJAN, R. A. P. Serverless architecture - a revolution in cloud computing. In: *2018 Tenth International Conference on Advanced Computing (ICoAC)*. [S.l.: s.n.], 2018. p. 88–93. Citado 2 vezes nas páginas 19 e 20.

RESEARCH on Early Warning System of Power Network Overloading under Serverless Architecture. In: 2018 2ª Conferência IEEE sobre Energia Internet e Integração do Sistema de Energia (EI2). [S.l.: s.n.]. Citado na página 28.

ROTHE, S.; NARAYAN, S.; SEVERYN, A. Leveraging Pre-trained Checkpoints for Sequence Generation Tasks. *Transactions of the Association for Computational Linguistics*, v. 8, p. 264–280, 06 2020. ISSN 2307-387X. Disponível em: <https://doi.org/10.1162/tacl_a_00313>. Citado na página 32.

SOUZA, C. R. B. de et al. Sometimes you need to see through walls: A field study of application programming interfaces. In: *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*. New York, NY, USA: Association for Computing Machinery, 2004. (CSCW '04), p. 63–71. ISBN 1581138105. Disponível em: <https://doi.org/10.1145/1031607.1031620>. Citado na página 21.

STAHLBERG, F. *Neural Machine Translation: A Review and Survey*. 2020. Citado na página 24.

SZELISKI, R. *Computer vision algorithms and applications*. London; New York: Springer, 2011. Disponível em: <http://dx.doi.org/10.1007/978-1-84882-935-0>. Citado na página 19.

VASWANI, A. et al. Attention is all you need. *CoRR*, abs/1706.03762, 2017. Disponível em: <http://arxiv.org/abs/1706.03762>. Citado 3 vezes nas páginas 20, 24 e 25.

WOLF, T. et al. *HuggingFace's Transformers: State-of-the-art Natural Language Processing*. 2020. Citado na página 32.

XUE, L. et al. *mT5: A massively multilingual pre-trained text-to-text transformer*. 2021. Citado na página 32.

ZAREMBA, W.; SUTSKEVER, I.; VINYALS, O. *Recurrent Neural Network Regularization*. 2015. Citado na página 25.