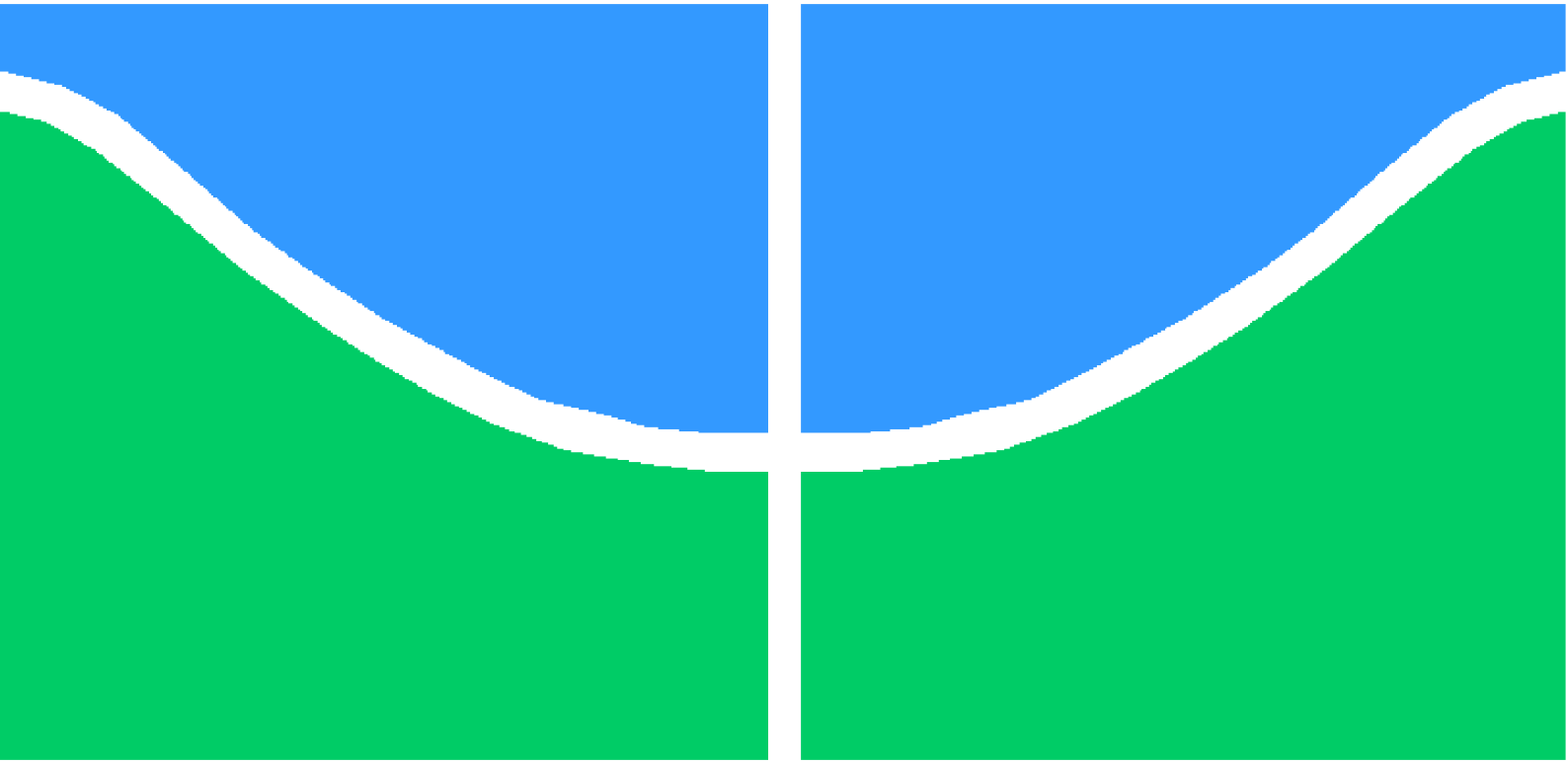

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Engenharia de Software

Pesquisa e desenvolvimento de técnicas de aprendizado de máquina contínuo

Autor: Cleber José de Castro Júnior

Professor Orientador: Dr. Nilton Correia da Silva

Brasília, DF

2022

Cleber José de Castro Júnior

Pesquisa e desenvolvimento de técnicas de aprendizado de máquina contínuo

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.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Dr. Nilton Correia da Silva

Brasília, DF

2022

Cleber José de Castro Júnior
Pesquisa e desenvolvimento de técnicas de aprendizado de máquina contínuo/ Cleber José de Castro Júnior. – Brasília, DF, 2022-
53 p. : il. (algumas color.) ; 30 cm.

Orientador: Dr. Nilton Correia da Silva

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2022.

1. Inteligëncia artificial. 2. MLOps. I. Dr. Nilton Correia da Silva. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Pesquisa e desenvolvimento de técnicas de aprendizado de máquina contínuo

CDU 02:141:005.6

Cleber José de Castro Júnior

Pesquisa e desenvolvimento de técnicas de aprendizado de máquina contínuo

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.

Trabalho aprovado. Brasília, DF, 23 de setembro de 2022 – Data da aprovação do trabalho:

Dr. Nilton Correia da Silva
Orientador

Dr. Fabricio Ataides Braz
Convidado 1

Dr. Jonathan Alis Salgado Lima
Convidado 2

Brasília, DF
2022

Agradecimentos

Agradeço a todos a todos os meus familiares e amigos que estiveram comigo nessa trajetória. Que me ajudaram a crescer não só na graduação, mas também, como pessoa. Agradeço também, a todos os professores que fizeram parte da minha formação, foram essências. Agradeço ao Prof. Dr. Nilton Correia da Silva por todos os ensinamentos, e por tornar possível esse trabalho. Agradeço ao Ailab, por tornar possível o profissional que sou hoje.

Resumo

No ciclo de vida de um modelo de aprendizado de máquina a atualização dos modelos criados se torna muito necessária, visto que novos dados podem estar sendo incluídos no decorrer do tempo. Porém, a técnica mais tradicional em aprendizado de máquina pressupõe um único momento de aprendizado. Isto é, dado um conjunto de dados de treinamento, executa-se um algoritmo de aprendizado de máquina no conjunto de dados para produzir um modelo que, embora possa resolver um determinado problema, com o decorrer do tempo e novos dados inseridos pode não ser mais assertivo. Um aprendizado de máquina que aprende continuamente, acumulando todo o conhecimento aprendido em momentos anteriores é importante para um ciclo de vida mais longo dos modelos implementados. O objetivo desta pesquisa é avaliar os principais métodos e técnicas focados em aprendizado contínuo, bem como, criar um sistema para dar suporte a aprendizado contínuo para sistemas de IA (Inteligência Artificial) voltados a processamento de linguagem natural aplicados a textos jurídicos.

Palavras-chave: DevOps, MLOps, Inteligência artificial.

Abstract

In the life cycle of a machine learning model, updating the models created becomes very necessary, as new data may be added over time. However, the most traditional technique in machine learning presupposes a single learning moment. That is, given a set of training data, a machine learning algorithm is run on the data set to produce a model that, although it can solve a given problem, with the lapse of time and new data entered may no longer be assertive. Machine learning that learns continuously, accumulating all the knowledge learned in previous moments is important for a longer life cycle of the implemented models. The objective of this research is to evaluate the main methods and techniques focused on continuous learning, as well as create a system to support continuous learning for AI (Artificial Intelligence) systems aimed at natural language processing applied to legal texts.

Key-words: DevOps, ML Ops, Artificial intelligence.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ML	Aprendizado de Máquina
IA	Inteligência Artificial
MLOps	*Machine Learning Operations*
DevOps	*Development Operations*
CLI	*Command-line Interface*
API	Interface de programação de aplicações
JSON	*JavaScript Object Notation*

Sumário

1 Introdução

1.1 Contextualização

O aprendizado de máquina tem sido fundamental para os avanços da análise de dados e inteligência artificial. Algoritmos de Machine Learning (ML) têm sido usados em quase todas as áreas da ciência da computação e em muitas áreas das ciências naturais, engenharia e ciências sociais. As aplicações práticas são ainda mais difundidas. É seguro dizer que, sem algoritmos de ML eficazes, muitas indústrias não teriam florescido, por exemplo, comércio digital e pesquisa na web (CHEN ZHIYUAN; LIU, 2016).

Atualmente o paradigma dominante para aprendizado de máquina consiste em executar um algoritmo de ML em um determinado conjunto de dados para gerar um modelo, esse modelo é então aplicado em tarefas de desempenho da vida real. Chamamos esse paradigma de aprendizagem isolada, porque não considera nenhuma outra informação relacionada, nem o conhecimento previamente aprendido ou até mesmo novos conhecimentos. O problema fundamental com este paradigma de aprendizagem isolado é que ele não tem memória, ele não retém e nem acumula o conhecimento aprendido no passado para usar em uma aprendizagem futura.

Considerando também que tudo muda constantemente e, portanto, a rotulagem deve ser feita continuamente, isso torna a tarefa mais difícil, pois o atual paradigma de aprendizagem isolado provavelmente não é adequado para a construção de um sistema verdadeiramente inteligente, mas apenas adequado para resolver problemas em domínios estreitos e com pouca mudança (LIU, 2017).

Com isso, o aprendizado de máquina demanda um ciclo de vida que o retroalimente e que o atualize para novos contextos, visto que sistemas com algoritmos de ML estão constantemente se atualizando e aprimorando. Como solução, um sistema capaz de realizar entregas e atualizações contínuas, trazendo conceitos desenvolvidos em práticas ágeis e Development Operations (DevOps) no contexto de aprendizado de máquina, conceito denominado Machine Learning Operations (MLOps).

1.2 Problema e Motivação

O desenvolvimento de modelos de aprendizado de máquina pode ser visto como um processo semelhante ao estabelecido para o desenvolvimento de software tradicional. A principal diferença entre os dois está na forte dependência entre a qualidade de um modelo de aprendizado de máquina e a qualidade dos dados usados para treinar ou realizar

avaliações (RENGGLI CEDRIC; RIMANIC, 2021).

Um aprendizado de máquina que aprende continuamente, acumulando o conhecimento aprendido em momentos anteriores é importante para um ciclo de vida mais longo de modelos implementados. Mesmo com vários sistemas já criados utilizando os conceitos do MLOps, um sistema com retreinamento utilizando novos dados e rótulos, é escasso. Assim, como o controle de versão de modelos de ML, apesar de existirem soluções, as mesmas não resolvem totalmente os problemas. O mesmo vale para a coleta de dados, processamento de dados, engenharia de recursos, rotulação de dados, *design* de modelos, treinamento e otimização de modelos, implantação de *endpoint* e monitoramento.

1.3 Objetivos

1.3.1 Objetivo Geral

Avaliar os principais métodos e técnicas focados em aprendizado contínuo, levantando seus pontos fortes e pontos fracos, bem como suas lacunas para um sistema desejável. Em seguida modelar e implementar um sistema para dar suporte a aprendizado contínuo para sistemas de IA voltados a processamento de linguagem natural aplicados a textos jurídicos.

1.3.2 Objetivos Específicos

- Levantar principais soluções de aprendizado contínuo;
- Comparar soluções de aprendizado contínuo, definindo pontos fortes, pontos fracos e pontos faltantes;
- Modelar um sistema de aprendizado de máquina contínuo;
- Implementar a solução levantada nos pontos anteriores;
- Validar o sistema implementado para modelos de processamento de linguagem natural.

2 Métodos e Materiais

Para a metodologia de desenvolvimento foi utilizado o método comparativo, que está centrado em estudar semelhanças e diferenças (PRODANOV C. C.; FREITAS, 2013), realizando comparações com o objetivo de verificar semelhanças e explicar divergências. Dado o contexto do tema escolhido o método definido para a pesquisa foi o comparativo para uma primeira parte. Inciaremos com uma comparação e em seguida, com tudo já levantado e comparado, seguimos para um método experimental, onde implementaremos um sistema de MLOps.

2.1 Fases da Pesquisa

Para a elaboração de uma pesquisa científica , é imprescindível conhecer os procedimentos e percursos a serem realizados (PRODANOV C. C.; FREITAS, 2013), assim, como podemos ver na Figura 1 a pesquisa será dividida em quatro grandes fases:

- fase da formulação e do planejamento da pesquisa;
- fase de coleta de dados e a busca de informações sobre o tema;
- fase da experimentação e desenvolvimento;
- fase de formulação da redação do texto final da pesquisa divulgação dos resultados.

2.1.1 Formulação e Planejamento da Pesquisa

Nessa fase foi escolhido o assunto, ciclo de vida em ML. Aqui foi realizado o levantamento do material bibliográfico, a elaboração do problema de investigação e pela delimitação das questões que determinam os objetos de estudo, com a investigação das produções bibliográficas relacionadas ao assunto estudado e o posterior recolhimento dessas fontes de informação. Aqui foi definido o foco em levantar as soluções existentes para o ciclo de vida de aprendizado de máquina contínuo, e também, foi definida a implementação de um sistema para dar suporte a aprendizado de máquina contínuo.

2.1.2 Coleta de Dados

De posse do tema e de uma boa base acerca do tema, foi procurado na biblioteca, através de fichários, catálogos, *abstracts*, uma bibliografia sobre o assunto, a qual forneceu os dados essenciais para a elaboração do trabalho. Selecionadas as obras, procedemos, em seguida, à localização das informações necessárias.

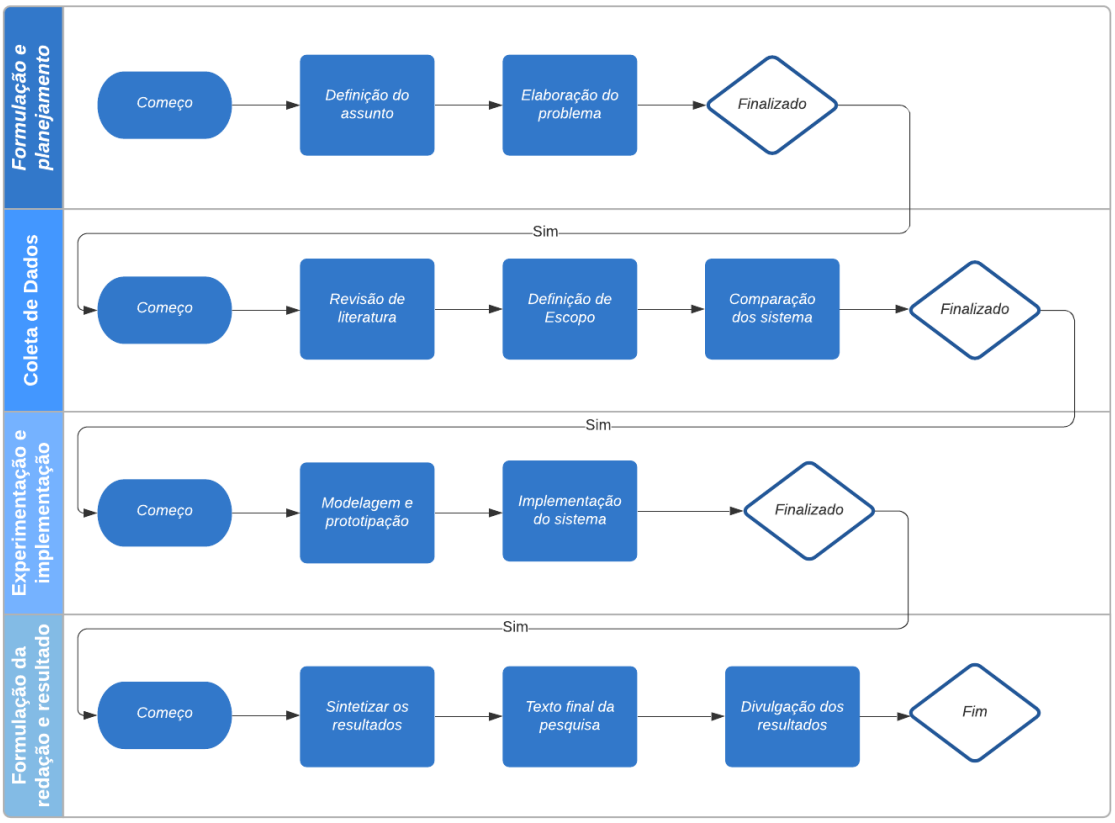

Figura 1 – Fluxograma das fases de pesquisa

2.1.2.1 Revisão de Literatura

Nessa fase foi pesquisado na literatura o que já foi feito sobre o tema, as soluções já existentes, trazendo uma maior referência sobre os aspectos já abordados, e as lacunas existentes na literatura. Aqui, descobriu-se o MLOps e várias ferramentas existentes que auxiliam no ciclo de vida de um projeto de aprendizado de máquina.

Durante essa fase foram levantadas as três ferramentas de MLOps para serem avaliadas e testadas, e em seguida comparadas, bem como o próprio MLOps, seus conceitos e categorias.

2.1.2.2 Definição de Escopo

Antes de iniciar a modelagem do sistema a ser desenvolvido é necessário definir a limitação de escopo. Decidimos implementar um sistema para modelos de NLP (Processamento de linguagem natural). O sistema terá foco em treinamento e retreinamento, além disso, terá implementação para controle de versão de modelos, endpoint de implantação, monitoramento, e estrutura de pipeline para implementação de modelos.

2.1.2.3 Comparação das Ferramentas

Com as soluções para o fluxo de vida de ML levantadas, agora comparamos suas funcionalidade, afim de, descobrir seus pontos fortes e fracos, e, conseguir definir uma ferramenta mais adequada ao problema com base no que já existe, e nas lacunas deixadas por elas.

2.1.3 Implementação do Sistema

Nesta fase será implementado o sistema de suporte a aprendizado de máquina contínuo. Aqui, também, será definida a arquitetura, as linguagens e *frameworks* adotadas, bem como qualquer comportamento mais específico do sistema. Terá como resultado o próprio produto final da pesquisa. Dos pontos já definido para o sistema, ele será implementado na linguagem de programação python, por ser uma das mais utilizadas e maior quantidade de bibliotecas de ML disponíveis.

2.1.4 Formulação da Redação do Texto Final da Pesquisa

Esse é o momento em que temos condições de sintetizar os resultados obtidos com a pesquisa. É retornado ao problema inicial lançado na introdução, revendo as principais contribuições que ele trouxe à pesquisa. Essa seção deve responder aos questionamentos que balizaram o estudo, de forma coerente com o que foi apresentado na seção introdutória (PRODANOV C. C.; FREITAS, 2013).

Divulgação dos resultados conseguidos com o estudo praticado para a comunidade científica e aos profissionais de sua área de atuação, intitula-se fase de exposição do trabalho final (PRODANOV C. C.; FREITAS, 2013).

2.2 Materiais

2.2.1 Gitlab CI/CD

É uma ferramenta de integração contínua e *deploy* contínuo integrada ao Gitlab. A escolha foi feita por conta do número de funcionalidades e especialmente pela integração

com o Kubernetes. A configuração é feita por meio de um arquivo chamado ".gitlab-ci.yml". Nesse arquivo, podemos definir múltiplas configurações desde *stages*, *branches* que executarão cada *stage*,*tags* para realização de *deploy* contínuo, etc. Será utilizado para automatizar submissões de modelos.

2.2.2 Docker

Ferramenta que permite a criação de ambientes virtuais, utilizando para isso contêineres Linux. Um contêiner é um conjunto de processos isolados do restante do sistema, o que permite virtualização em nível de sistema operacional.

2.2.3 Mongo DB

É um software de banco de dados orientado a documentos livre, de código aberto e multiplataforma, escrito na linguagem C++. Classificado como um programa de banco de dados *NoSQL*, o MongoDB usa documentos semelhantes a JSON com esquemas.

2.2.4 Portainer

Plataforma de gerenciamento de contêineres com interface de usuário intuitiva e práticas codificadas que ajudam as organizações a criação contêineres de forma rápida e eficiente.

2.2.5 Git

Git é um sistema de controle de versão distribuído gratuito e de código aberto projetado para lidar com tudo, desde projetos pequenos a muito grandes com velocidade e eficiência.

2.2.6 Python

Linguagem de programação de alto nível muito utilizada no contexto de ML.

2.2.7 Flask

Flask é um *framework* web escrito em Python, para criar aplacações Web.

2.2.8 Lucid Chart

Lucid chart é uma plataforma proprietária baseada em web que permite aos usuários colaborar na elaboração, revisão e compartilhamento de gráficos e diagramas. É utilizado para diagramação de arquitetura do sistema, e diagramas de sequencia de métodos.

2.2.9 DigitalOcean

Plataforma de serviços de computação em nuvem, com vários serviços muito escaláveis, será utilizada para disponibilização e testes do sistema a ser criado.

3 Referencial Teórico

3.1 MLOps

O MLOps é uma cultura e uma prática de engenharia de aprendizado de máquina que visa unificar o desenvolvimento de sistemas de ML e a operação de sistemas de ML. A prática de MLOps significa que você defende a automação e o monitoramento de todos os passos da construção do sistema de ML, inclusive integração, teste, lançamento, implantação e gerenciamento de infraestrutura (MLOPS. . . , 2020). Geralmente, MLOps, como um conceito, é focado no aprendizado de máquina em produção.

O conceito de MLOps se baseia na cultura DevOps, que é uma prática comum no desenvolvimento e na operação de sistemas de software em larga escala. Essa prática oferece benefícios como encurtar os ciclos de desenvolvimento, aumentar a velocidade de implantação e as versões confiáveis (MLOPS. . . , 2020). Um sistema de aprendizado de máquina é um sistema de software. Sendo assim, práticas semelhantes se aplicam para garantir que você crie e opere sistemas de ML de maneira confiável em escala. Porém, um sistema de ML possui algumas diferenças significativas, é um sistema que os testes do sistema vão muito além de testes unitários e de integração, é preciso avaliar a qualidade do modelo após o treinamento. Em produção, um modelo de ML pode diminuir a sua performance com o tempo, tornando necessária uma ação. Na Figura 2 podemos visualizar onde o MLOps se encontra dentro da cultura DevOps, da engenharia de dados e do aprendizado de máquina, sendo assim uma junção dos três.

Os sistemas de Machine Learning podem ser distribuídos em oito categorias diferentes:

- coleta de dados;
- processamento de dados;
- engenharia de recursos;
- rotulagem de dados;
- design de modelos;
- treinamento e otimização de modelos;
- implantação de *endpoint* e monitoramento.

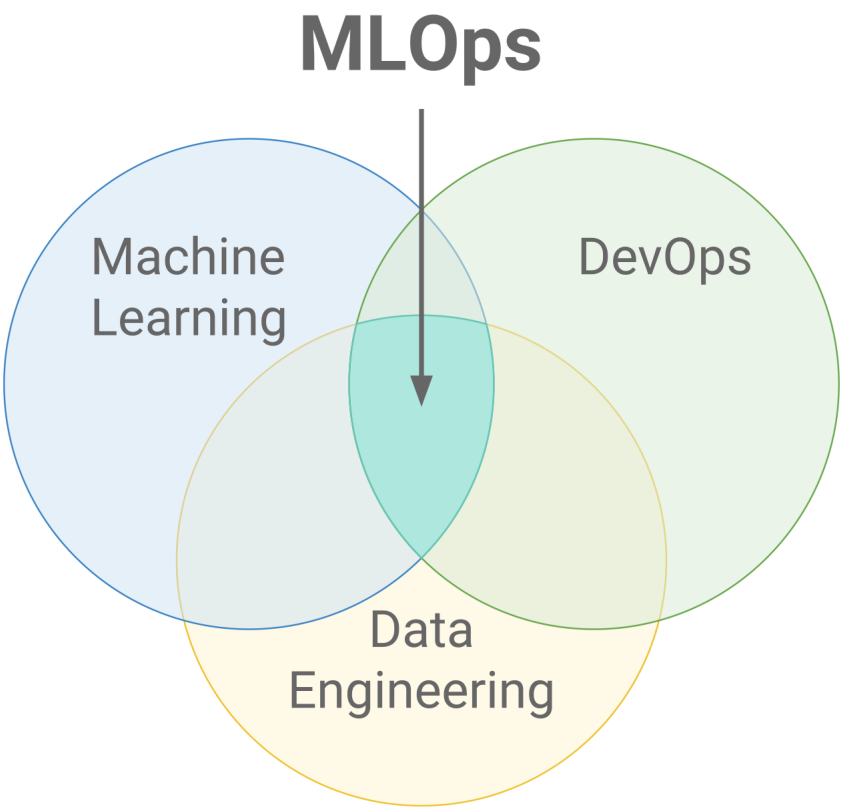

Figura 2 – MLOps, DevOps e Engenharia de Software

Fonte: (MLOPS. . . , b).

Cada passo no ciclo de vida de aprendizagem de máquina é construído em seu próprio sistema, mas requer interconexão.

3.2 Sistemas de MLOps

Comparar os sistemas de aprendizado de máquina e MLOps não é trivial, pois esses produtos são complexos. Geralmente, as diferenças entre as plataformas só podem ser totalmente percebidas com testes do mundo real com um caso de uso real. Uma maneira comum de comparar sistemas é comparar recursos. Muitos desses sistemas são muito parecidos em recursos em uma primeira vista, mas na prática variam enormemente (MLOPS. . . , a). Para levantar o foco de cada sistema, foi levado em consideração como cada um se posiciona sobre suas propostas. Foram levantados seis sistemas como podemos ver na Figura 3.

Para a pesquisa, foram selecionados três, como vemos na Tabela 1, sistemas que se apresentaram mais completos nas categorias que são foco deste trabalho, na parte de modelagem, treinamento e deploy. E todos possuem fácil acesso, pois são de código aberto.

Nome	coleta de dados	processamento de dados	design de modelos	treinamento e otimização de modelos	deploy e monitoramento
Kubeflow		x	x	x	x
MLFlow		x	x	x	x
DVC					
Metaflow		x	x	x	
Airflow	x	x		x	
Seldon					x

Figura 3 – Sistemas de MLOps

São eles o *Kubeflow*, o *MLFlow* e o *Metaflow*. Outro fator de escolha está na forma em que eles funcionam, tanto o *Kubeflow* quanto o *MLFlow* possuem interface gráfica para controle dos modelos, mas se diferenciam na sua forma de instalação, enquanto o primeiro exige uma complexa instalação via *kubernetes*, o segundo possui uma instalação mais simples. Já o *Metaflow* se diferencia não possuindo interface gráfica, mas com uma estruturação via linha de comando bem definida. Assim, poderemos verificar um sistema com uma instalação robusta e complexa, uma mais simples mas com a maior parte dos recursos da primeira, e um que funciona de forma completamente diferente dos anteriores.

Tabela 1 – Principais sistemas de MLOps

Nome	**Foco**
Kubernetes	Comunidade, Extensão, Ciclo de Vida
MLFlow	Experimentação, Controle de versão
Metaflow	Pipelines

3.2.1 Kubeflow

O Kubeflow é uma plataforma criada para aprimorar e simplificar o processo de implantação de fluxos de trabalho de *machine learning* no Kubernetes. Usando o Kubeflow, fica mais fácil gerenciar uma implantação distribuída de ML, colocando componentes no pipeline de implantação, como os componentes de treinamento, atendimento, monitoramento e registro em *log* em contêineres no *cluster* Kubernetes (INTRODUCTION..., 2021).

O objetivo do Kubeflow é abstrair os aspectos técnicos do gerenciamento de um *cluster* Kubernetes para que se possa aproveitar rapidamente o poder do Kubernetes e os benefícios da implantação de produtos em uma estrutura de microsserviços.

O Kubeflow possui uma série de componentes para o fluxo de trabalho em ML, se destacam:

- Serviços para geração e gerenciamento de Jupyter Notebooks. Usado para ciência de dados interativa e experimentos com fluxos de trabalho de ML;
- Pipelines, uma plataforma para construir, implantar e gerenciar fluxos de trabalho de ML de várias etapas com base em contêineres Docker;
- Oferece vários componentes que podem ser usados para criar treinamentos de ML, ajuste de hiper-parâmetros e atendimento de cargas de trabalho em várias plataformas.

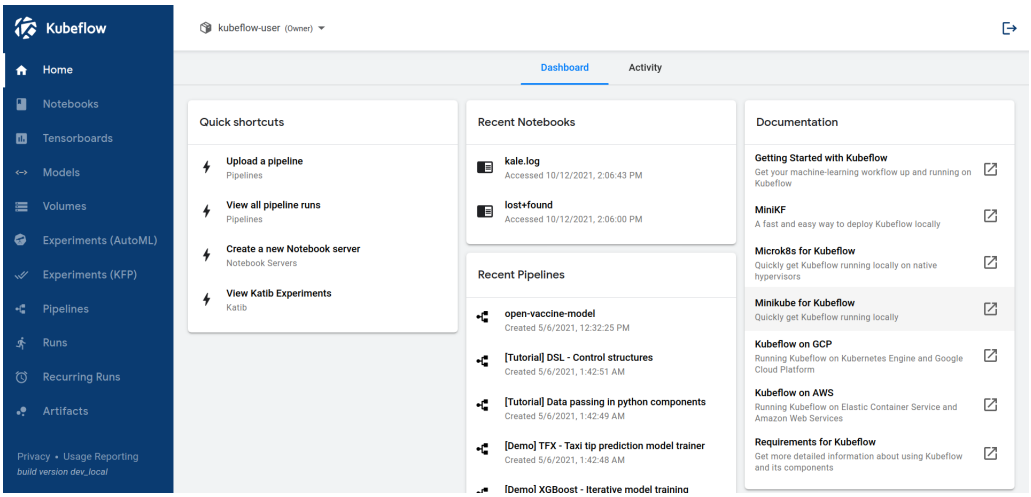

Figura 4 – Interface do Kubeflow

Dentro das funcionalidades principais do Kubeflow, a gestão de Jupyter Notebook e as Pipelines se destacam.

3.2.1.1 Jupyter Notebook

Existem vários benefícios de integrar notebooks Jupyter no Kubeflow para ambientes corporativos. Esses benefícios incluem:

Integra-se bem com o resto da infraestrutura no que diz respeito à autenticação e controle de acesso. Habilitando o compartilhamento de bloco de notas mais fácil em toda a organização. Os usuários podem criar contêineres de notebook ou *pods* diretamente no *cluster*, em vez de localmente em suas estações de trabalho. Os administradores podem fornecer imagens de notebook padrão para sua organização e configurar o controle de acesso baseado em função, segredos e credenciais para gerenciar quais equipes e indivíduos podem acessar os blocos de notas.

Ao agrupar blocos de anotações Jupyter no Kubeflow, você pode usar a biblioteca Fairing para enviar trabalhos de treinamento usando TFJob. O *job* de treinamento pode ser executado em um único nó ou distribuído no mesmo *cluster* do Kubernetes, mas não

dentro do *pod* de notebook em si. O envio do trabalho com a biblioteca Fairing torna os processos como a conteinerização do Docker e a alocação de *pods* claros para os cientistas de dados. No geral, os notebooks hospedados pelo Kubeflow são melhor integrados com outros componentes, enquanto fornecem extensibilidade para imagens de notebook.

3.2.1.2 Kubeflow Pipelines

Uma plataforma simples para criar e implantar fluxos de trabalho de aprendizado de máquina em contêineres no Kubernetes. Facilitam a implementação de pipelines de *Machine Learning* de nível de produção sem se preocupar com os detalhes de baixo nível do gerenciamento de um *cluster* Kubernetes. Um componente principal do Kubeflow e também é implantada quando o Kubeflow é implantado.

Uma pipeline é uma descrição de um fluxo de trabalho de ML, incluindo todos os componentes no fluxo de trabalho e como eles se combinam na forma de um grafo. Cada componente da pipeline é um conjunto independente de códigos empacotados como imagens do Docker. Inclui a definição das entradas (parâmetros) necessárias para executar a pipeline e as entradas e saídas de cada componente. Um componente é uma etapa do fluxo de trabalho. Consistem em:

- Código do cliente: o código que fala com os terminais para enviar tarefas;
- Código de tempo de execução: o código que executa o trabalho real e geralmente é executado no *cluster*;
- Uma interface de usuário (UI) para gerenciar e rastrear experimentos, trabalhos e execuções;
- Notebooks para interagir com o sistema.

Por exemplo, um componente pode ser responsável pelo pré-processamento de dados, transformação de dados, treinamento de modelo e assim por diante. Depois de desenvolver a pipeline, você pode carregá-la e compartilhá-la na interface do usuário dos Kubeflow Pipelines.

3.2.1.3 Vantagens do Kubeflow

Kubeflow é indicado para uma equipe (dentro de uma organização) iniciando sua jornada nativa na nuvem com o K8s, que pode querer aproveitar a consistência oferecida pelo Kubeflow para cargas de trabalho de ML para novos projetos. Para equipes de pesquisa ou institutos que desejam minimizar a complexidade do gerenciamento de uma infraestrutura para um cientista de dados ou pesquisador e, em vez disso, fornecem uma interface limpa e consistente que facilita a configuração com apenas alguns cliques.

3.2.2 MLFlow

MLflow é uma plataforma de código aberto para gerenciar o ciclo de vida de aprendizado de máquina de ponta a ponta, incluindo experimentação, reprodutibilidade, implantação e registros de modelos. A ferramenta lida com quatro funções principais, o Tracking, Project, Model e Registry.

3.2.2.1 Tracking

O *Tracking* é um módulo do MLflow que tem como objetivo armazenar todos os registros de métricas, parâmetros, *tags* e artefatos de cada execução individual do código de um modelo, que é chamada pela ferramenta de *run*.

A cada execução é possível registrar os artefatos localmente, por convenção no diretório */mlruns* ou em um banco de dados. De acordo com a documentação, o MLflow suporta, para o banco de dados, dialetos mysql, mssql, sqlite, e postgresql (MLFLOW. . . , c).

Para definir o servidor de rastreamento, é necessário previamente definir o diretório de armazenamento dos experimentos, que por default é *mlruns/*, o URI e a porta para receber a interface disponibilizada pelo MLflow.

A figura 5 ilustra a dinâmica do tracking, com os experimentos, runs, métricas, parâmetros, diretórios, entre outros.

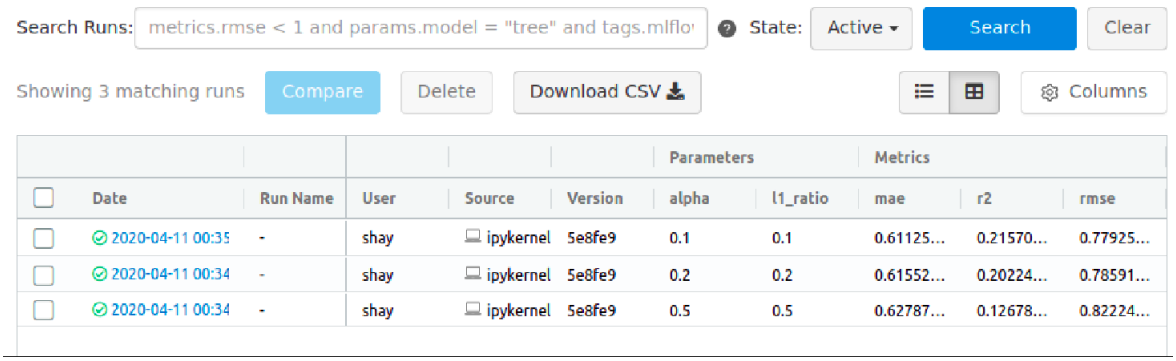

Figura 5 – MLFow Tracking

3.2.2.2 Project

Este módulo permite que as execuções individuais, com os arquivos gerados previamente, possam ser empacotados, reproduzidos e replicados em outros ambientes, como para pessoas da mesma equipe, por exemplo. Isso é possível devido a uma geração do MLflow de arquivos contendo réplicas de configurações do ambiente original, com instruções para futuras réplicas (MLFLOW. . . , b). Podemos ver o funcionamento na figura 6.

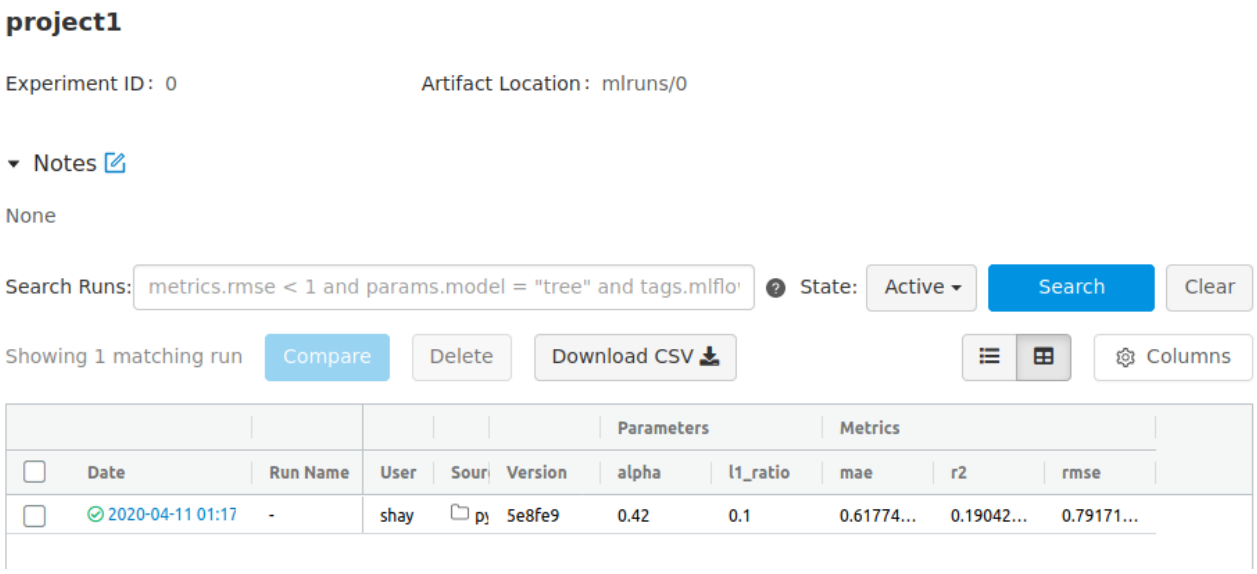

Figura 6 – MLFow Project

3.2.2.3 Model

Um modelo MLflow é um formato padrão para empacotar modelos de aprendizado de máquina que pode ser usado em uma variedade de ferramentas de downstream - por exemplo, entrega em tempo real por meio de uma API REST ou inferência em lote no Apache Spark. O formato define uma convenção que permite salvar um modelo em diferentes "sabores" que podem ser entendidos por diferentes ferramentas de downstream (MLFLOW..., a).

3.2.2.4 Registry

O Registry é um módulo com o objetivo de centralizar os modelos que queremos manter, como os que obtiveram os melhores resultados de uma forma segura. Ele permite o fornecimento de informações sobre a versão do modelo, em que etapa este modelo se encontra (em produção, por exemplo) e diversas anotações, além de ser mais facilmente publicada para outras pessoas.

3.2.3 Metaflow

Metaflow é uma biblioteca Python amigável que ajuda cientistas e engenheiros a construir e gerenciar projetos de ciência de dados da vida real. O Metaflow foi originalmente desenvolvido na Netflix para aumentar a produtividade dos cientistas de dados que trabalham em uma ampla variedade de projetos, desde estatísticas clássicas até aprendizado profundo de última geração. Metaflow fornece uma API unificada para a pilha de infraestrutura necessária para executar projetos de ciência de dados, do protótipo à pro-

dução (WHAT. . . ,). Utilizando de uma estrutura de pipeline, ele define todos os processos necessários em código. Possui as seguintes quatro funções principais:

- Fornece uma API altamente utilizável para estruturar o código como um fluxo de trabalho, ou seja, como um grafo direcionado de etapas (usabilidade);
- Persista um instantâneo imutável de dados, código e dependências externas necessárias para executar cada etapa (reprodutibilidade);
- Facilita a execução das etapas em vários ambientes, desde o desenvolvimento à produção (escalabilidade , prontidão para produção);
- Registre metadados sobre execuções anteriores e torne-os facilmente acessíveis (usabilidade , reprodutibilidade).

3.2.3.1 Fluxo

Um fluxo é a menor unidade de computação que pode ser programada para execução. Normalmente, um fluxo define um fluxo de trabalho que extrai dados de uma fonte externa como entrada, os processa em várias etapas e produz dados de saída.

O usuário implementa um fluxo criando uma subclasse FlowSpece implementando etapas como métodos. Além das etapas, um fluxo pode definir outros atributos relevantes para o agendamento, como parâmetros e acionadores de dados.

3.2.3.2 Grafo

Metaflow infere um grafo direcionado (normalmente acíclico) com base nas transições entre as funções de etapa. Requer que as transições sejam definidas para que o grafo possa ser analisado estaticamente a partir do código-fonte do fluxo. Isso torna possível traduzir o grafo para execução por tempos de execução que suportam apenas grafos definidos estaticamente.

3.2.3.3 Etapa

Uma etapa é a menor unidade de computação recuperável. É implementado pelo usuário como um método decorado com o decorador @step em uma classe de fluxo.

Uma etapa é um ponto de verificação . O Metaflow tira um instantâneo dos dados produzidos por uma etapa que, por sua vez, é usado como entrada para as etapas subsequentes. Portanto, se uma etapa falhar, ela pode ser retomada sem executar novamente as etapas anteriores.

Poder retomar a execução é um recurso poderoso. Seria conveniente poder retomar a execução em qualquer linha de código arbitrária. A principal razão pela qual o ponto

de verificação é feito no nível da etapa em vez de no nível da linha é a sobrecarga do estado de salvamento. O usuário é encorajado a manter os passos pequenos, mas não tão pequenos que a sobrecarga se torne perceptível.

3.2.3.4 Decoradores

O comportamento de uma etapa pode ser modificado com decoradores. Tags são o principal mecanismo para estender o Metaflow. Por exemplo, um decorador pode capturar exceções, implementar um tempo limite ou definir requisitos de recursos para uma etapa. Uma etapa pode ter muitos decoradores arbitrários, implementados como decoradores Python.

3.2.3.5 Step Code

O código da etapa é o corpo de uma etapa. Ele implementa a lógica real de fluxo de negócios. É possível implementar várias ligações de linguagem, por exemplo, R, para Metaflow, de modo que apenas a linguagem do código da etapa seja alterada, enquanto toda a funcionalidade central, implementada em Python, permanece intacta.

Todas as variáveis de instância, por exemplo *self.x*, usadas no código da etapa, tornam-se artefatos de dados que são persistidos automaticamente. Variáveis de pilha, por exemplo x, não são persistentes. Essa dicotomia permite que o usuário controle a sobrecarga do ponto de verificação escolhendo explicitamente entre variáveis persistentes e não persistentes no código da etapa.

3.3 Comparação dos sistemas

MLflow e Kubeflow são líderes de categoria nas plataformas de aprendizado de máquina de código aberto, mas são muito diferentes. Para simplificar, o Kubeflow resolve a orquestração da infraestrutura e o rastreamento de experimentos com o custo adicional de ser bastante exigente para configurar e manter, enquanto o MLflow resolve bem o rastreamento de experimentos (e controle de versão do modelo).

O Kubeflow atende aos requisitos de equipes maiores responsáveis pelo fornecimento de soluções de ML de produção personalizadas. Essas equipes costumam ter funções mais especializadas e os recursos necessários para gerenciar a infraestrutura do Kubernetes. O MLFlow, por outro lado, atende melhor às necessidades dos cientistas de dados que procuram se organizar melhor em torno de experimentos e modelos de aprendizado de máquina. Para essas equipes, a facilidade de uso e configuração costuma ser o principal fator. Já o Metaflow cria uma estrutura de pipeline muito robusta e simples de ser executada, apesar de não possuir uma interface gráfica, o que não deixa tão acessível todo os metadados, registros e rastreamentos que ele faz.

Um sistema ideal teria a escalabilidade e facilidade de executar experimentos do Kubeflow, com a organização e controle de versão do MLFlow, junto com a estrutura de pipelines do Metaflow. Com isso podemos definir o que seria o sistema a ser implementado, utilizando os pontos fortes de cada ferramenta e inserindo o que mais fez falta, que foi uma abordagem para um retreinamento. Nenhum dos sistemas enfatiza o fato da inserção de novos dados, ou até mesmo a avaliação do modelo em produção.

4 Pontos Propostos

Neste capítulo serão desenvolvidos os pontos propostos dos capítulos anteriores, os quais foram utilizados para o comparativo das ferramentas de MLOps. Assim, será tratado em cada seção um ponto da aplicação.

4.1 Proposta inicial

Um sistema para o ciclo de vida de ML que disponibilize recursos para as oito diferentes categorias dentro do MLOps divido em três grandes módulos. Um primeiro que será o centro de tudo, uma API REST responsável pelo treino e retreino de modelos, controle de versão dos modelos executados, predição, e monitoramento das predições executadas. O segundo módulo, uma estrutura em python, que disponibilizará a estrutura de pipeline necessária para um treinamento de novos modelos, nele existirão pré-definições para a execução dos modelos implementados (semelhante ao que o Metaflow cria de estrutura para um modelo). E por fim o terceiro módulo, responsável por conectar os modelos criados ao módulo principal onde será executado o treinamento, será uma CLI atuando na submissão dos modelos criado, assim o sistema poderá ser integrado à ferramentas de integração contínua com Gitlab CI, Github Actions, entre outros. A figura 7 apresenta o fluxo de treinamento de um novo modelo.

4.2 Proposta final

Para a proposta final da aplicação o conceito original foi mantido. Toda a ideia de escrever o código do modelo e submeter via *git*, para em seguida utilizar pipelines de integração contínua que iniciam um treinamento no servidor central. Esse sistema possui 5 módulos, como vemos a seguir:

- Repositório de controle de versão de dados;
- Jupyter Notebook para experimentação de modelos e manipulação dos dados;
- Repositório de Modelo com arquivo python que define o modelo, e CI/CD para execução do modelo no servidor;
- Servidor central para treinamento e controle dos modelos;
- Servidor de inferência e monitoramento;

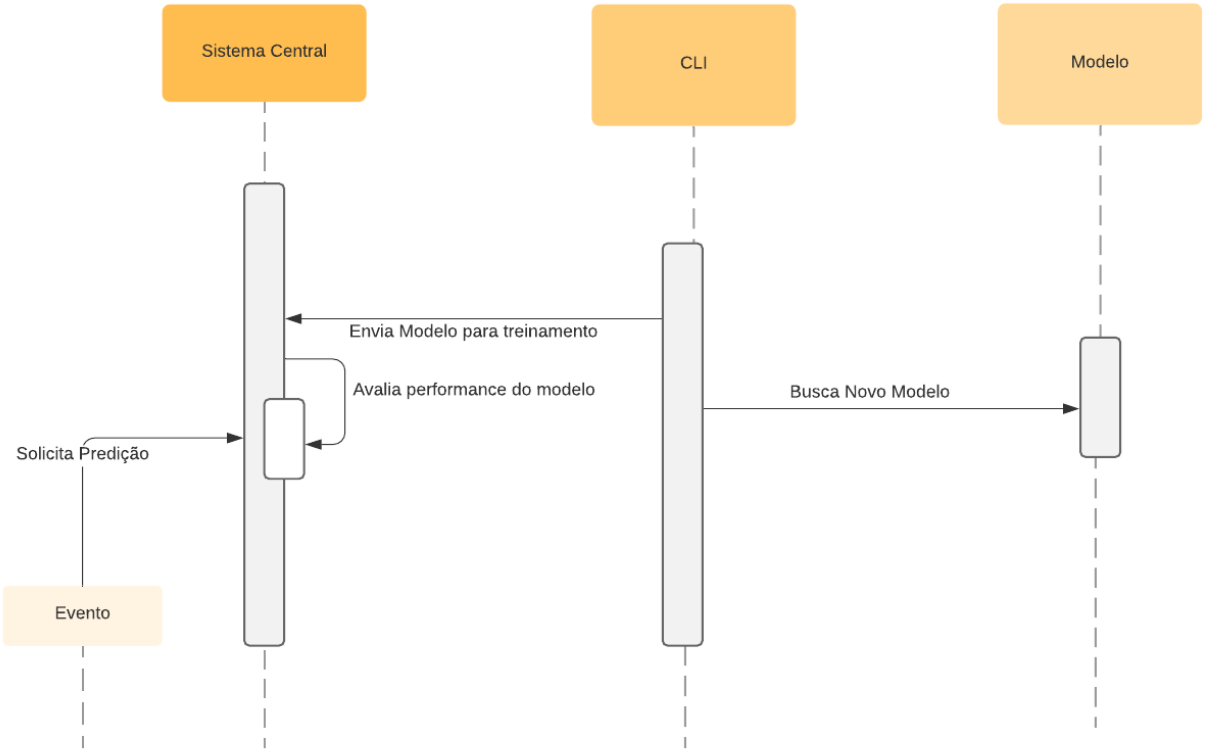

Figura 7 – Diagrama de sequência do sistema projetado inicialmente, fluxo de novo treinamento

4.3 Controle de versão de dados

Para o desenvolvimento de soluções de IA, um dos pontos de maior importância é o dado. Por muitas vezes é o ponto decisivo para a viabilidade de um projeto. As principais preocupações acerca do dado estão relacionadas à extração, o armazenamento, e o controle de versão.

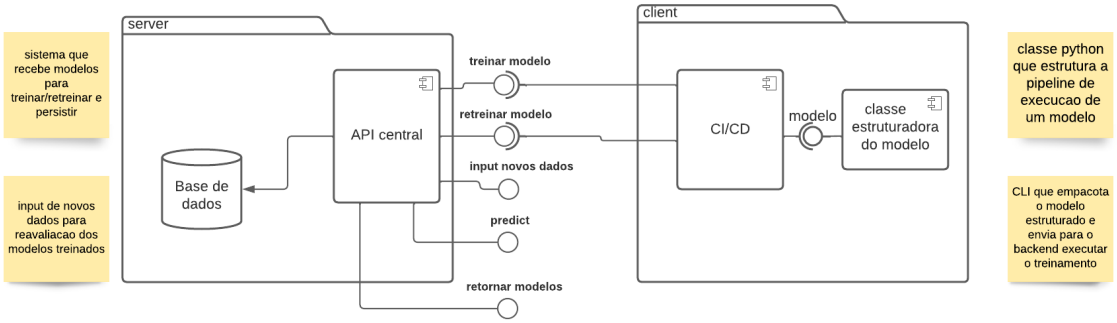

Figura 8 – Diagrama de componentes inicial

Para o desenvolvimento do projeto, a parte de extração não está no escopo planejado, já o armazenamento utilizará da biblioteca pandas para leitura e manipulação dos dados, que podem ser de diversos formatos como parquet, csv, json. E ainda, o controle de versão será feito pela biblioteca Data Version Control (DVC), que é uma ferramenta de gerenciamento de experimentos de dados e ML que aproveita o conjunto de ferramentas de engenharia existente como o Git.

4.4 Jupyter Notebook

Ainda sobre o dado, uma necessidade muito importante para o engenheiro de IA é a possibilidade de manipular o dado com diversas bibliotecas. Como todo o sistema foi pensado para ser versátil a diferentes modelos e dados, também será disponibilizado um Jupyter Notebook com o repositório do DVC montado em sua pasta principal. Dessa forma, o responsável por tratar o dado não terá limitação sistêmica. Podendo assim, executar qualquer procedimento para manipular o dado e em seguida salvar o novo dado com o controle de versão.

Além de manipulação do dado, também será configurado para conectar com o servidor central, onde os modelos principais estão sendo armazenados e versionados. Com isso, pode-se executar testes manuais nos modelos treinados caso seja necessário, ou até comparar os modelos existentes.

4.5 Design de modelos

Logo após as etapas de dados concluída, seguimos para a etapa de design de modelos. Nela definimos todos os arquivos necessários para o treinamento. Nessa etapa é configurada a pipeline que gera uma imagem docker com os arquivos de execução do modelo e suas dependências. Em seguida, executa o deploy para o servidor central, onde o modelo é treinado. O repositório de modelos possui os seguintes arquivos:

- model.py: arquivo python com todos os processos de execução do modelo, tais como leitura dos dados, treinamento;
- requirements.txt: arquivo com as dependências de execução do modelo em questão;
- .gitlab-ci.yml: arquivo que define as configurações de *build* e *deploy* do modelo;
- Dockerfile: configuração da imagem docker que define a forma de execução do modelo.

Dessa forma, para criar um novo modelo basta criar um repositório com esses arquivos e executar a pipeline. Assim, é gerada uma imagem docker com o modelo definido no arquivo model.py e, logo em seguida, a execução do deploy inicia o treinamento. Ao final, o modelo é disponibilizado na interface do servidor.

4.6 Servidor central

Uma etapa essencial do sistema é o servidor central, que é o centro da aplicação. Nele todos os modelos executados são salvos, junto com seus metadados, e também, o controle dos mesmos. Aqui podemos controlar qual versão fica em produção, ou staging, e temos todo o histórico de execuções.

Inicialmente a ideia era implementar essa aplicação para o servidor central. Porém, o grande problema seria na compatibilidade com bibliotecas de ML. Dessa forma foi definido o uso do *MLFlow* para cumprir esse papel. Com isso, o sistema se torna mais versátil, por conta da grande compatibilidade do *MLFlow* com bibliotecas de ML.

Para configurar o *MLFLow*, está sendo utilizado o *server* do mesmo. Tudo configurado em uma imagem docker, configurada e disponibilizada com integração contínua pelo Gitlab CI.

4.7 Inferência e monitoramento

Por fim, feito o treinamento e deploy do modelo, chegamos a etapa final de uma iteração no sistema. A inferência e o monitoramento do modelo é feita por uma API desenvolvida em Python, com a biblioteca Flask. Nesta API podemos executar inferência de um modelo em produção, em homologação, ou até mesmo um modelo específico utilizando o ID fornecido pelo *MLFlow*. Além disso, após uma inferência que o usuário da aplicação considere incorreta, ele pode enviar o resultado esperado de volta para a api. Esse processo alimenta uma nova base de dados, que pode ser utilizada para um treinamento futuro, dessa forma o sistema se retroalimenta. E também, o usuário tem acesso a uma rota com métricas relacionadas à assertividade do modelo.

A API possui as seguintes rotas:

- Inferência de modelos em produção - POST - /api/models/production/<model-name>;
- Inferência de modelos em homologação - POST - /api/models/staging/<model-name>;
- Inferência de modelos em EXPERIMENTAÇÃO - POST - /api/models/runs/<run-id>;
- Recuperar novos dados salvos - GET - /api/export/<model-name>;
- Inserir novo dado após inferência incorreta - PUT - /api/export/<model-name>;
- Exportar novos dados para o DVC - POST - /api/export/<model-name>;
- Monitorar métricas de assertividade do modelo - GET - /api/health/<model-name>

5 Resultados

5.1 Sistema

O sistema foi implementado com sucesso, com quase tudo que foi inicialmente planejado. As exceções ficam por parte do retreinamento automático, e também a coleta de métricas para viabilizar uma avaliação de retreinamento automático. Para esse caso o sistema dispõe de funcionalidades para coleta de novos dados, e visualização do desempenho do modelo em produção. Dessa forma o usuário pode executar um novo treinamento com novos dados, porém não executado de forma automática.

O restante da aplicação está disponível. Dispondo de etapas de experimentação de dados, treinamento e desenvolvimento e inferência de modelos como podemos ver na Figura 9.

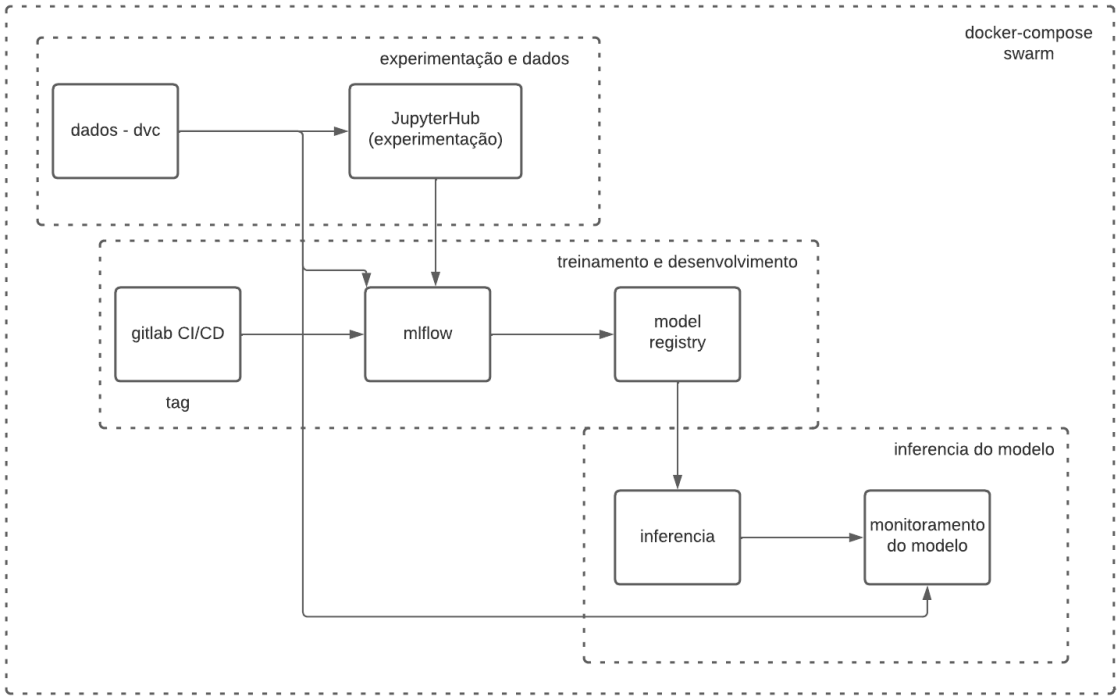

Figura 9 – Diagrama de componentes do sistema

5.2 Fluxo da aplicação

1. Na Figura 10 vemos a estrutura de arquivos definidos para o DVC, nele estão metadados dos dados e Jupyter Notebooks utilizados para manipulação dos mesmos.

Name	Last commit	Last update
.dvc	add dvc template	2 weeks ago
data	add dvc template	2 weeks ago
models	add dvc template	2 weeks ago
notebooks	add dvc template	2 weeks ago
results	add dvc template	2 weeks ago
src	add dvc template	2 weeks ago
.dvcignore	add dvc template	2 weeks ago
.gitignore	add dvc template	2 weeks ago
.pre-commit-config.yaml	add dvc template	2 weeks ago
.pylintrc	add dvc template	2 weeks ago
LICENSE	add dvc template	2 weeks ago
README.md	add dvc template	2 weeks ago
metadata.yaml	add dvc template	2 weeks ago
requirements.txt	update requirements	10 hours ago

Figura 10 – Repositório de controle de versão de dados

2. Em seguida, podemos ver o JupyterLab 11 onde está sendo feito testes e manipulações dos dados que logo em seguida são mapeados pelo DVC 12.

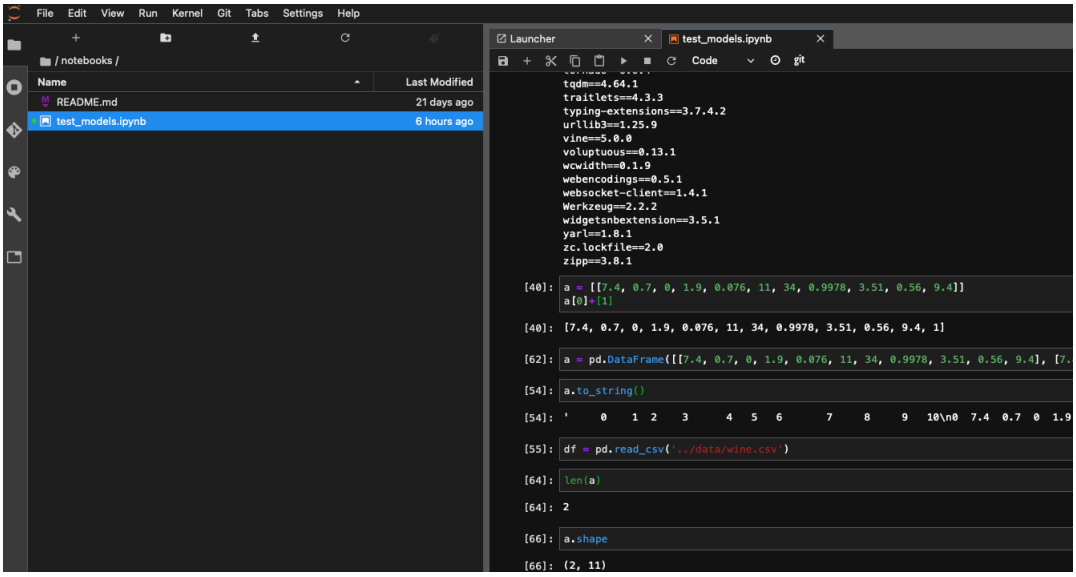

Figura 11 – JupyterLab

3. O passo seguinte é a criação de um modelo que é feita no Gitlab, podemos ver na Figura 13 todos os arquivos necessários para gerar uma versão do modelo via docker, fazer o deploy pela pipeline e, por fim, conectar com o *MLFlow* para salvar os dados do

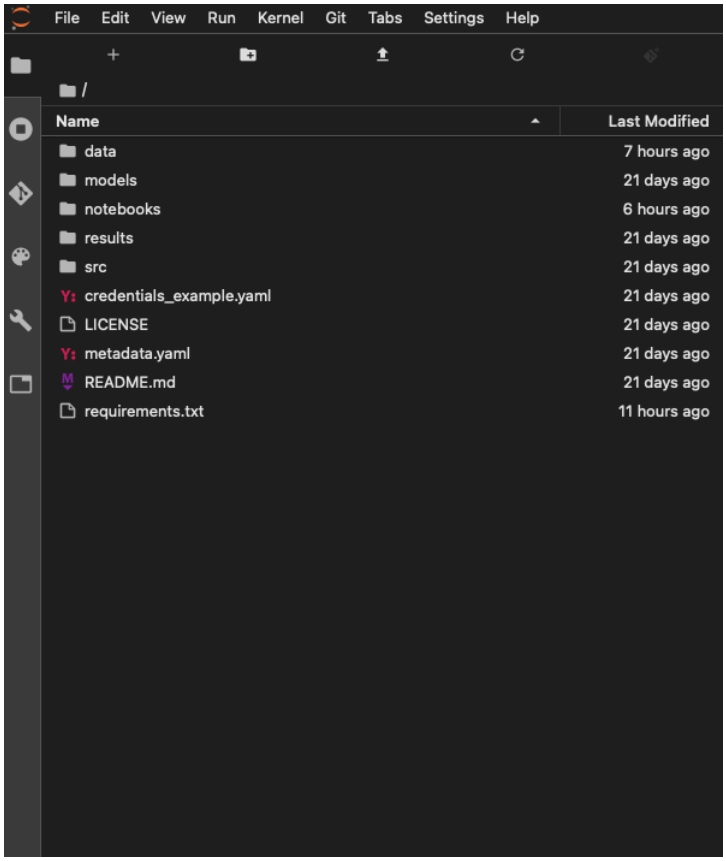

Figura 12 – Pastas do DVC no JupyterLab

modelo.

Name	Last commit	Last update
.gitlab-ci.yml	initial commit	12 hours ago
Dockerfile	initial commit	12 hours ago
README.md	Initial commit	13 hours ago
model.py	initial commit	12 hours ago
requirements.txt	initial commit	12 hours ago

Figura 13 – Repositório de modelo no Gitlab

4. Para a execução do modelo no ambiente é necessário executar a pipeline do modelo, as pipelines foram definidas para serem executadas em criação de TAGs no padrão esperado (v0.0.0). Podemos ver na Figura 14 a execução de uma pipeline.

5. Em seguida a imagem Docker gerada é executada no servidor, figura 15.

6. Após a execução do container o modelo já aparece disponível na interface do *MLFlow*, onde é possível visualizar suas métricas e manuseá-lo para um ambiente como o de produção 16.

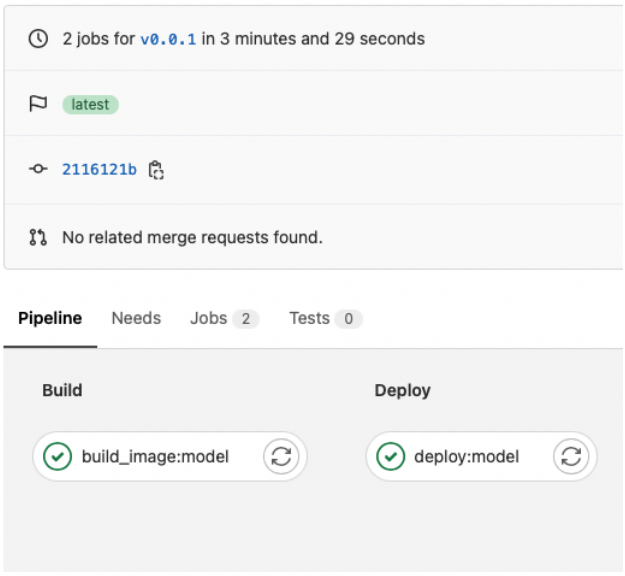

Figura 14 – Pipeline de modelo no Gitlab CI/CD

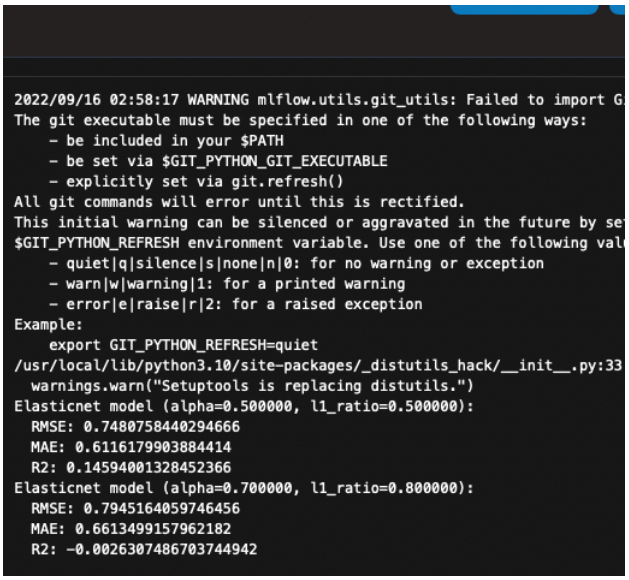

Figura 15 – Execução do modelo no servidor

7. Por fim, podemos utilizar o modelo via API como vemos nas Figuras 17 e 18.

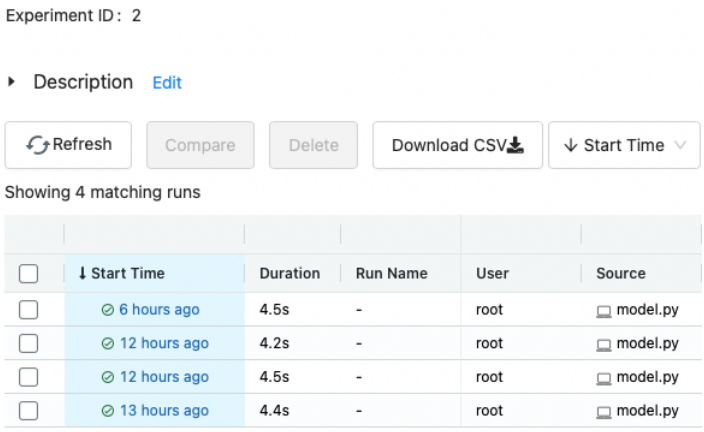

Figura 16 – Dashboard do *MLFlow* com o novo modelo salvo

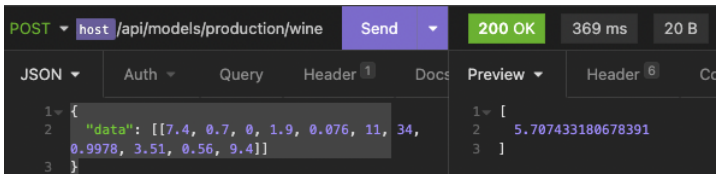

Figura 17 – API de inferência e monitoramento

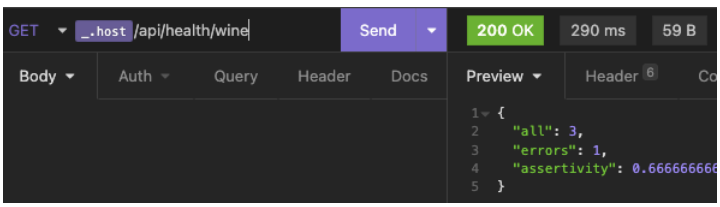

Figura 18 – API de inferência e monitoramento, métrica de assertividade

6 Conclusão

A aplicação de MLOps no ciclo de vida de ML se mostrou muito útil, visto que muitos processos podem ser automatizados. E também, podemos ter um fluxo organizado e rastreável. O sistema implementado consegue acompanhar praticamento do início ao fim de um ciclo de ML, contribuindo com treinamento e a retroalimentação de modelos. Contudo, ainda há o que evoluir no sistema para entrar em cenários de uso profissionais onde necessita de uma robustez maior. O projeto se encontra na organização mlops-system [1].

6.1 Trabalhos futuros

6.1.1 Retreinamento automático

O retreinamento dos modelos, atualmente, só funciona de forma manual. Foram criados recursos para que o modelo se retroalimente, porém, falta uma definição de métrica para o momento de retreino sem a intervenção humana. Isso seria interessante pois o sistema ficaria mais completo, podendo funcionar por mais tempo sem necessitar de uma nova intervenção e pipeline.

6.1.2 Melhor camada de manipulação de dados

O sistema, atualmente, executa métodos simples para todo o tratamento de dados e não entrou na extração de dados. Uma evolução interessante seria adicionar recursos para a extração e definir uma manipulação de dados mais robusta, como criar funções pré definidas para automatizar parte do processo.

6.1.3 Orquestração e distribuição de *containers*

Uma execução em cluster Kubernetes é muito interessante para a arquitetura da aplicação. Executar o treinamento dos modelos de forma distribuída tende a ser muito interessante para usos mais intensos. Nesse caso o servidor central ficaria em um cluster, sendo responsável pela execução dos serviços e pela gestão de recursos. Provavelmente seria a melhor solução de implantação, pois, se trata uma arquitetura escalável.

[1] https://gitlab.com/mlops-system

7 Cronograma

7.1 Cronograma referente ao semestre 2021.1

Tabela 2 – Cronograma 2021.1

Atividades	**Julho**	**Agosto**	**Setembro**	**Outubro**	**Novembro**
Definir problema, objetivo	x				
Pesquisa bibliográfica	x	x	x		
Definição do Escopo		x			
Revisão da literatura		x	x		
Comparação de ferramentas			x	x	
Planejamento do sistema				x	
Escrita do TCC			x	x	
Revisão de Texto					x
Defesa do TCC					x

7.2 Cronograma referente ao semestre 2021.2

Tabela 3 – Cronograma 2022.1

Atividades	**Junho**	**Julho**	**Agosto**	**Setembro**
Modelagem do sistema	x	x		
Prototipação do sistema	x	x		
Implementação do sistema		x	x	x
Análise de dados obtidos			x	x
Escrita do TCC				x
Revisao do TCC				x
Defesa do TCC				x

Referências

CHEN ZHIYUAN; LIU, B. *Lifelong Machine Learning*. 1. ed. ed. USA: Morgan Claypool Publishers, 2016. ISBN 978-1-627-05501-7, 978-1-627-05877-3. Citado na página 19.

INTRODUCTION to Kubeflow. 2021. <https://www.kubeflow.org/docs/about/kubeflow/>. Acessado: 2021-09-30. Citado na página 29.

LIU, B. Lifelong machine learning: a paradigm for continuous learning. Department of Computer Science, University of Illinois at Chicago, Chicago IL 60607, USA, 2017. Citado na página 19.

MLFLOW Models. <https://www.mlflow.org/docs/latest/models.html>. Acessado: 2021-09-30. Citado na página 33.

MLFLOW Projects. <https://www.mlflow.org/docs/latest/projects.html>. Acessado: 2021-09-30. Citado na página 32.

MLFLOW Tracking. <https://www.mlflow.org/docs/latest/tracking.html>. Acessado: 2021-09-30. Citado na página 32.

MLOPS: pipelines de entrega contínua e automação no aprendizado de máquina. 2020. <https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning?hl=pt-br>. Acessado: 2021-09-30. Citado na página 27.

MLOPS Platforms Compared. <https://valohai.com/mlops-platforms-compared/>. Acessado: 2021-09-30. Citado na página 28.

MLOPS Working Group. <https://about.gitlab.com/company/team/structure/working-groups/mlops/>. Acessado: 2021-09-30. Citado na página 28.

PRODANOV C. C.; FREITAS, E. C. *Metodolodia do trabalho científico: Métodos e Técnicas da Pesquisa e do Trabalho acadêmico*. 2. ed. ed. Novo Hamburgo – RS: Editora Feevale, 2013. Citado 2 vezes nas páginas 21 e 23.

RENGGLI CEDRIC; RIMANIC, L. G. N. M. K. B. W. W. Z. C. A data quality-driven view of mlops. Microsoft Research, 2021. Citado na página 20.

WHAT is Metaflow. <https://docs.metaflow.org/introduction/what-is-metaflow>. Acessado: 2021-09-30. Citado na página 34.