Productionalization of an object detection system with MLOps Practices

*Abstract*—This paper discusses the challenges of using machine learning models in production and solution of adopting Machine Learning Operations (MLOps) practices. MLOps is a set of practices that has been used to improve the machine learning model development, deployment, and management processes. The paper discusses the advantages of using MLOps over manual machine learning and discusses the requirement specifications and system design for an object detection system with end-to-end functionality. The system also includes functional and non-functional requirements of the components, such as data engineering, model engineering, model testing and validation, model deployment, CI/CD pipelines, monitoring, and triggering. This paper also discusses the development cycle of machine learning models and MLOps practices for hosting, monitoring, and managing machine learning models. This paper also discusses the design and implementation of an object detection system using MLOps principles. In the future work, it discusses the addition of feature stores, AutoML, and version control to the proposed object detection system.

Keywords— ML (Machine Learning), MLOps (Machine learning operations), Amazon Sagemaker, Azure, Google Cloud Platform, Databricks, Docker, Kubernetes, CI/CD pipeline, Docker, Kubernetes, Flask, TensorFLow, TensorFlow Serving, MLFlow, KubeFlow, DataRobot, TensorFlow Extended, AutoML, Version control system.

# Introduction

Machine learning (ML) has emerged as a crucial method for maximizing the potential of data and enabling organizations to be more creative, effective, and viable. However, the performance of numerous useful ML applications in production contexts falls short of predictions [1]. As written by [2], the challenge of using machine learning models in production has persisted since the development of the first models. According to the [3], majority of real-world applications involve dynamic data. This suggests that to address feature drift, ML models must be retrained, or, in the worst case, the complete ML pipeline must be rebuilt. Data scientists, machine learning engineers, front end engineers, and production engineers attempted to find a way to cooperate and combine their knowledge to deploy models that are ready for production [2]. To solve ML production issues, a solid set of production techniques must be used. A potential candidate to describe these standardised production techniques is Machine Learning Operations (MLOps) [3].

In the past [1], the discipline of software engineering has seen the emergence of various software process models and development methodologies. The agile manifesto and waterfall are two prominent methodologies. The goal of these methodologies is to produce software products that are suitable for production. DevOps is a concept that was introduced in the years 2008 and 2009 and seeks to minimise problems in software development and deployment. DevOps involves the use of continuous integration and continuous delivery (CI/CD) pipelines, automated testing, and monitoring and feedback loops to enable fast and reliable software delivery. MLOps extends the principles and practices of DevOps to address the unique requirements of machine learning model development and deployment, with a focus on automation, collaboration, and continuous improvement [4]. According to the [5], Machine Learning Operations (MLOps) is a collection of procedures and guidelines for implementing and monitoring machine learning models in production. To ensure that machine learning models are deployed and managed efficiently and effectively, MLOps entails the integration of various technologies, including machine learning frameworks, deployment tools, and monitoring systems. Therefore, MLOps is a set of practices and tools used to improve the machine learning model development, deployment, and management processes.

Adopting MLOps practices can bring several benefits that can accelerate the time-to-market for ML projects. MLOps ensures a consistent and repeatable process by automating all aspects of the Machine Learning Development Cycle (MLDC) process, including model training, evaluation, versioning, and deployment [6]. Moreover, these benefits include increased productivity by providing self-service environments that give access to pre-validated data sets, saving time that would otherwise be wasted on missing or invalid data. Additionally, with the integration of CI/CD practices, MLOps enables fast and reliable deployment of the models with improved quality. By versioning all inputs and outputs, MLOps ensures auditability, offering clear insights into how the model was developed and deployed. Furthermore, MLOps practices can enforce policies that guard against model bias and keep track of changes in data statistical properties and model quality over time, ensuring high-quality data and models [7].

MLOps can pose several challenges during the implementation and it’s a difficult task to overcome them [8]. According to the [2], the challenge is in building reliable, effective MLOps pipelines (Entire machine learning development life cycle, from data preparation, model training to deployment and monitoring in production environments) with good compatibility. Moreover, collaboration on MLOps can be challenging because of the specialized skills required and the involvement of multiple teams with different priorities and objectives. Therefore, it's crucial to have cross-functional cooperation between data scientists, data engineers, development teams, and other stakeholders. [7]. Additionally, adopting MLOps techniques requires significant investments in resources, tools, and technology infrastructure. Another significant challenge is convincing stakeholders who don't fully comprehend MLOps of their return on investment (ROI) [9].

MLOps is enabled by multiple underlying technologies, such as cloud computing, containerization, version control, continuous integration and continuous delivery (CI/CD), monitoring, and logging. This is a concise explanation of these underlying technologies.

Cloud computing makes MLOps scalable, adaptable, and affordable. Cloud computing lets organisations immediately scale their computer resources. Cloud computing lets enterprises manage machine learning workflows including data processing, model training, and deployment. AWS, Azure, and Google Cloud Platform offer tools and services for deploying and managing machine learning models [10].

MLOps uses containers to simplify machine learning model deployment. Businesses can test their models in different environments regardless of architecture or dependencies. Docker and Kubernetes are popular in MLOps because they standardise packaging, deploying, and maintaining machine learning models [11][12].

Version control systems (VCS) like Git are prominent in MLOps for maintaining and tracking machine learning model changes. VCS lets organisations track code and data changes, manage machine learning model versions, and interact in real time [13]. VCS also allows firms to backtrack to prior machine learning model iterations, reducing errors and inconsistencies. [14].

MLOps procedures use modified CD and CI software development approaches. CI/CD automates machine learning model development, testing, and deployment, saving time and money [15][16]. Famous CI/CD pipeline tools are GitLab, GitHub Actions and Jenkins etc.

Monitoring and logging are crucial MLOps components that allow enterprises to monitor their machine learning models in real-world contexts. Monitoring analyses measurements and KPIs linked to machine learning model performance, whereas logging tracks events and actions related to its deployment and use. Monitoring and logging allow organisations to spot and fix issues in real time, ensuring their machine learning models perform properly and benefit end users [17] [14].

The format of this section of paper as follows: The second part of the paper will have background and considerations of MLOPs, in the third part I will design an object detection system at scale, fourth part will have conclusion and finally the last part will have a discussion on future work.

# Background and Considerations

## MLOps Workflow and Toolchain

MLOps is a new area of focus that aims to apply the principles of DevOps to machine learning workflows. Its purpose is to help organizations develop, deploy, and manage machine learning models in a manner that is both consistent and reliable, while also allowing for scalability [14]. MLOps workflow consists of three stages: design, model development, and operations.

Diagram

Description automatically generated

**Figure - 1** MLOps Workflow [18]

Above Figure-1 shows the iterative-incremental process in MLOps. The process involves breaking down the MLOps lifecycle into a series of smaller, more manageable stages, each consisting of an iterative cycle of activities. According to the [18], in first stage, designing a machine learning solution to solve the problem and the future development assessed. The design phase also includes prioritizing ML use cases, inspecting available data, specifying functional and non-functional requirements, designing the architecture of the ML-application, and establishing the serving strategy. In the second stage, model development, the goal is to build and verify a machine learning model to solve a real-world problem. During this phase, different machine learning algorithms are iteratively tested and refined. Data engineering and model engineering are also performed to create a stable and high-quality machine learning model that can be deployed in production. The primary focus of this phase is to ensure that the machine learning model can accurately and effectively solve the problem it was designed to address. Finally, in the last stage, the developed ML model is delivered in production. This phase includes practices like model deployment, testing, versioning, CI/CD pipelines, and monitoring. These three stages are interconnected, and a design choice made during the design stage will have an impact on the model development stage and then on the deployment choices made during this stage will impact on the operations stage.

The collection of tools and technologies used to implement the MLOps workflow are referred to as MLOps toolchains. In recent years, many tools have been developed to automate the machine learning phases. These toolchains automate all the three main stages of MLOps, includes design, data and model exploration, and operations to guarantee consistency and reproducibility [2]. As mentioned in [19], there are two types of toolchains available in the MLOps, end-to-end machine learning pipelines and isolated tools. End-to-end machine learning pipelines cover the entire project life cycle in a single cohesive solution. Some of the end-end machine learning pipelines are Amazon Web Services (AWS) SageMaker, Azure AI (Artificial Intelligence) Platform, Google Cloud AI (Artificial Intelligence) Platform, TensorFlow Extended (TFX) and Kubeflow etc. By using these pipelines developer can improve efficiency, consistency, and scalability of the machine learning models. On the other hand, isolated tools developed for specific life-cycle tasks, but each isolated task must then be integrated into whatever custom pipeline the developers are using. Each stage of MLOps workflow has several isolated toolchains include, Version Control Systems (VCS) such as Git, Data Version Control (DVC), data warehouses such as Google BigQuery, data preprocessing tools such as Pandas, NumPy and Scikit-learn, data management tools such as MLFlow and CometML, model training such as TensorFlow, PyTorch and Keras, CI/CD tools such as Jenkins and CircleCI, automated deployment tools such as Buildah, Docker and Kubernetes, and model monitoring tools such as Datadog and Grafana [19] [2], [20] [20]. The use of an end-to-end machine learning pipeline is frequently preferred over isolated tools because it provides single integrated pipeline, organizations can improve efficiency, consistency, reproducibility, scalability, automation, and collaboration [19].

## MLOps frameworks and their comparison

MLOps frameworks are software tools that makes easier to develop, deploy, manage, and monitor machine learning models in production environments. There are several frameworks in MLOPs that support the deployment of models into production. As mentioned in this Artificial Intelligence (AI) blog [18], MLOps’ commercial and open-source frameworks are readily available, and both offer automation and orchestration of the MLOps stages. The support and documentation offered by commercial offerings are usually comprehensive, and pre-built integrations with other services and infrastructure are also offered. They might also provide advanced features like managed infrastructure and automated machine learning. Open-source offerings, on the other hand, are frequently more flexible and adaptable, and they might have a more active developer community adding to their improvement. According to [12], well-established organisations like Amazon Web Services, Google, and Microsoft provide commercial MLOps frameworks such as Amazon Sagemaker, Google Cloud AI Platform, and Azure AI Platform. Despite that, open source MLOps frameworks like MLflow, Kubeflow, and TensorFlow Extended (TFX) have gained traction due to their customizable and adaptable nature. Selection of MLOps frameworks between commercial and open source depends on an organization's specific needs, priorities, and budget, as well as the complexity of the machine learning problem.

Following is the comparison of some of the famous MLOps frameworks which data scientists have been practising.

|  |  |  |  |
| --- | --- | --- | --- |
| Framework | Open Source | Interface | Launched Year |
| MLFlow | Yes | CLI/GUI | 2022 |
| KubeFlow | Yes | CLI | 2018 |
| Pachyderm | Yes | GUI | 2014 |
| DataRobot | No | GUI | 2012 |
| TensorFlow  Extended (TFX) | Yes | CLI/GUI | 2019 |

**Table – 1** Open source, Interface and Launched year comparison of MLOps Frameworks [2], [20]–[22]

|  |  |  |
| --- | --- | --- |
| Framework | Usage | Support |
| MLFlow | Managing end-to-end machine learning workflows includes, tracking, model versioning and deployment | Scikit-learn, Tensorflow,  PyTorch, Keras, Hugging face, Transformers, MLlib |
| KubeFlow | Provides an end-end machine learning platform for building, deploying, and managing scalable machine learning workflows on Kubernetes. | Tensorflow, PyTorch, Scikit-learn, KubeFlow Piplines |
| Pachyderm | Building scalable and data-driven machine learning pipelines that supports data versioning and lineage | Tensorflow, PyTorch, Scikit-learn, Pandas |
| DataRobot | Automates and standardise each stage of machine learning journey from data to value. It uses advanced algorithms to automate machine learning tasks. | Tensorflow, PyTorch, Keras Scikit-learn, Pandas |
| TensorFlow  Extended (TFX) | Building scalable and production-ready machine learning pipelines. | TensorFlow, Keras, TFLite |

**Table – 2** Usage and Support comparison of MLOps Frameworks [[2], [20]–[22]

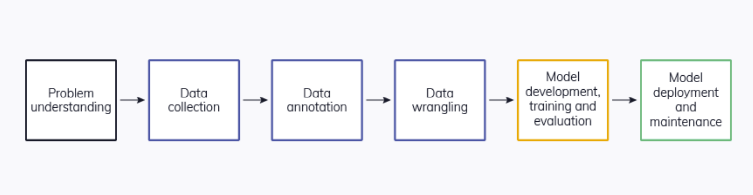
|  |  |  |
| --- | --- | --- |
| Framework | Advantages | Disadvantages |
| MLFlow | Transparency and standardization of ML lifecycle | It does not have multi-use environment.  Not perfect for deploying models to various platforms |
| KubeFlow | Integration of multiple frameworks | Manual setup and configuration challenges |
| Pachyderm | Supports distributed training and model serving | Limited support for some machine learning frameworks and tools |
| DataRobot | Easy to build machine learning model because of automation | Large volume of data could take a long time.  No connection between RDBMS database type like mysql. |
| TensorFlow  Extended (TFX) | The complete range of pipeline actions is available, including the ability to build, update, run, list, and delete pipelines. | It might not be appropriate for small-scale deployments or simple machine learning problems. |

**Table – 3** Pros and Cons of MLOps Frameworks [2], [20]–[22]

All the three tables Table 1, Table 2 and Table 3 shows the comparison of five well-known frameworks, MLFLow, KubeFlow, Pachyderm, DataRobot and TensorFlow Extended (TFX). From the Table 1, all the frameworks are open source except DataRobot. According to this e-commerce company [23], the cost of Datarobot for plan “Starter Pack for AutoML” is $98,000 per year. The second column of the table shows which framework has graphical user interface (GUI) and which has command-line interface (CLI). The third column shows the launched year of each framework. Table 2 shows the usage and mostly used supporting libraries and frameworks by all these frameworks. In Table 3, advantages and disadvantages of each framework has been discussed.

## Productionalisation and end-to-end manual/automatic cycles

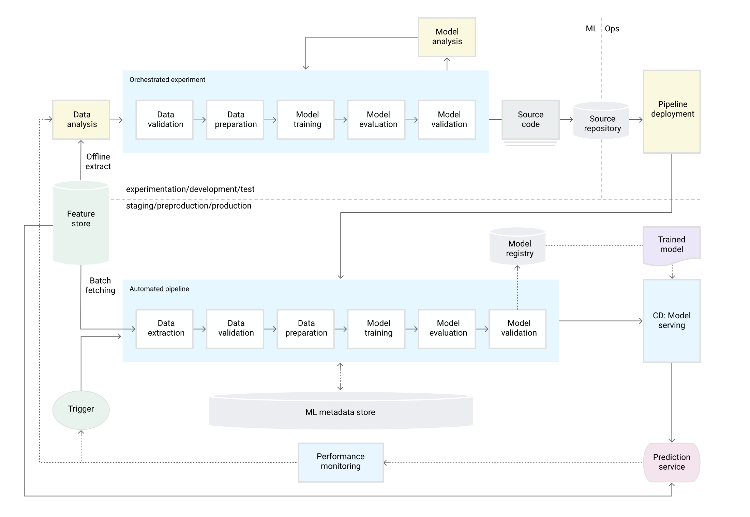
Productionalization is the development of a machine learning model from the  development phase to a stage where it can be deployed in a production environment. To do this, end-to-end pipelines consisting of data collection, data pre-processing, model training, model deployment, and model monitoring must be established [24]. There are two types of ML end-to-end pipeline, manual and automated cycles.



**Figure - 2** ML end-to-end manual pipeline [25]

Figure 2 shows the end-to-end manual ML cycle, includes data preparation, model training, evaluation and validation, and model deployment and maintenance. Each process must be completed manually and the transition from one step to the next. This process is typically driven by experimental code that data scientists directly write and run in notebooks until a usable model is created.

When deployed in the real world, machine learning models may encounter changes in the environment or data that were not present during training. In such situations, a manual procedure led only by data scientists may not be sufficient to guarantee the model continues to perform effectively. This is because models may need to be updated or retrained to accommodate these changes, and depending only on manual processes can be inefficient and time-consuming. Thus, it is essential to implement systems for monitoring and continuously enhancing the performance of machine learning models over time [5]. As a result, there has been a trend towards automating ML pipelines using tools and frameworks such as Kubeflow [26], MLflow [27], and TensorFlow Extended [28].



**Figure - 3** ML end-to-end automatic pipeline [5]

Figure 3 shows the end-to-end automatic cycle, focused on improving the speed, agility, and reliability of the machine learning (ML) development and deployment process. It involves automating the ML experiment steps to enable rapid iteration, implementing a continuous training (CT) process for the model in production, and ensuring an experimental-operational symmetry between the development and production environments. To achieve this, the code for the ML components and pipelines to be modularized, reusable, and containerized to ensure code reproducibility and component isolation. Additionally, the pipeline must continuously deliver prediction services based on new models that are trained on fresh data. Finally, the entire training pipeline is deployed to production, instead of just the trained model, which helps to maintain consistency and ensure the production environment is ready for new models. Overall, automating ML pipeline provides the foundation for achieving greater efficiency, scalability, and reliability in the ML development and deployment process [5].

## Feature stores and its roles and benefits within the ML pipeline

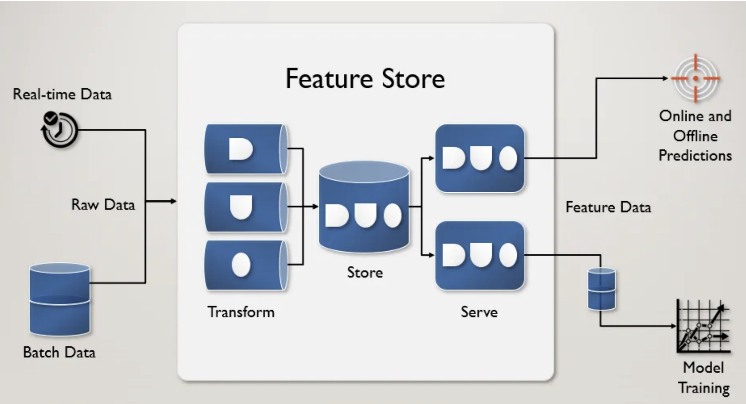
A feature store is a central location where you can standardize how features are defined, stored, and accessed for training and serving. A feature store needs an API (Application Programming Interface) that enables both training and serving tasks, as well as high-throughput batch serving and low-latency real-time serving for the feature values. The data scientists and other stakeholders can access feature store to standardise the process of creating and using machine learning models by providing a sole source of truth for features. The feature store helps to ensure that all stakeholders are using the same data and definitions, reducing the risk of errors or inconsistencies. It also makes it easier to reuse features across multiple models and applications, which can save time and improve efficiency [5]. Below is the basic workflow of feature store which includes, data sources, feature store and use in machine learning model development.



**Figure - 4** Dataflow with feature store [29]

Figure 4 shows the basic dataflow diagram of feature store. The first component, Raw/Structured Data, shows the data sources from where data scientists can get the raw data and pass to the feature store for the feature engineering. The raw data can be extracted from any data source like snowflake, databricks, mongo DB etc. In the second stage, Feature store than do feature engineering on the data which includes, data ingestion, compute features and stores it to the feature store. After that at last, these transformed features can be used for model development by using any ML platform or development techniques or libraries.

The feature store can be set up with two databases, one of which serves as an offline feature store for experimentation with normal latency and the other as an online store for production predictions that require low latency features [1]. An offline feature repository is to train and test data for models during model development and batch inference. This type of feature store typically makes use of a SQL-backed database that is designed to handle massive amounts of feature data. For real-time inference, an online feature repository is necessary. With the help of the feature definition saved in the registry made when populating the offline store, this low-latency database will be used to enhance real-time datasets [30].



**Figure - 5** Feature store with real-time and batch data[29]

Figure 5 shows the workflow of feature store with real-time and batch data. For real-time data, feature store will receive real-time data and will make it possible for the machine learning model to process it as soon as possible. Feature store will update the features in real-time as it will gets the new data. In contrast, for the batch data, feature store handles large amount of the data, and it will handle the data in batches.

In MLOps, a feature store's primary advantages include ensuring data consistency, enabling reuse of features, and scaling across thousands of models. These advantages contribute to a machine learning model's increased effectiveness and precision [31]. According to [29], instead of having each data scientist recreate the same features repeatedly, a feature store allows a data scientist to make this transformation once. Since everyone uses the exact same transformation as part of their models, consistency is ensured. Additionally, a feature store allows a business to modify a complex feature once and propagate it across all models that depend on it. Otherwise, all models using that functionality would need to be manually edited. Furthermore, feature stores shorten the duration of development and let developers start a new project more rapidly [32]. Feature stores produce output that is independent of execution. The program or model will receive data in a consistent format regardless of the algorithm or framework we use.

Feast [33], Tecton [34], Hopsworks [35], and Amazon SageMaker Feature Store [31] are a few of the well-known feature store systems.

# Design an Enterprise Machine Learning System

The task has been assigned to create an object detection system that utilizes a scalable model server with GPU capabilities. The system should host a pre-existing TensorFlow 2 Object detection model from the model zoo (<https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md>) and provide a user-friendly web interface to upload images and receive results. There are several considerations to accomplish this task which includes requirement specification of the problem, system design, infrastructure test plan, and containerize/non-containerize solution. There are two different approaches to develop and deploy machine learning models, manual machine learning and MLOps. As discussed before there are several advantages of using MLOps over manual machine learning so MLOps approach or principles will be used for an object detection system.

## Requirement Specifications

The requirement specifications of an object detection system will include functional and non-functional requirements. The entire workflow of an object detection system requires several functional and non-functional requirements from data preparation to model monitoring. When using MLOps, functional and non-functional requirements are significant factors at every step of the development lifecycle of an object detection system. In MLOps, the three stages of the development lifecycle are usually design, model development and operations. In design phase data scientists usually do requirements engineering, use case prioritization and data availability check of ML problem. Here our focus is on model development and operations phase to specify functional and non-functional requirements for the object detection system. These two stages may include substages like data engineering, model engineering, model testing and validation, model deployment, CI/CD pipelines, monitoring, and triggering. This system's functional requirements will be the desired outcome of the specific tasks performed in substages of model development and operations stages. The non-functional requirements of the system will include collaboration, scalability, reliability, efficiency, performance, and maintainability of the system.

**Functional Requirements**

An object detection system is a complex machine learning system that needs a lot of various components and steps to work properly. The functional requirements for an object detection system can be divided into the following phases of the MLOps pipeline:

* **Data Engineering:** The object detection system must be capable of ingesting and pre-processing data from many sources, such as Snowflake, Databricks, Mongo DB, etc. The system must manage massive amount of data and guarantee data quality and consistency. Moreover, the system must support data augmentation strategies to enhance model performance. Additionally, the capacity to manage data versioning and updates.
* **Model Engineering:** To implement machine learning models, the appropriate object identification techniques and models must be selected. The object detection system must be capable of fine-tuning models for certain use cases and datasets. In addition, the system must allow transfer learning techniques to reduce the quantity of required training data. Additionally, the capacity to manage model versioning and management.
* **Model Testing and Validation:** The object detection system must assess model performance using metrics including precision, recall, and F1-score. The system must provide interpretability methodologies for models in order to comprehend model behaviour and identify biases. In addition, the capacity to manage model validation using approaches such as cross-validation. In addition, the capacity to manage testing and validation with huge datasets at scale.
* **Model Deployment:** The object detection system must facilitate deployment of models, including on-premises, cloud, and edge devices. The system must be able to accommodate model serving and scalability needs. In addition, support for model deployment management and automation.
* **CI/CD Pipelines:** The object detection system must be capable of automating the system's deployment processes. In addition, the system must allow continuous deployment and continuous integration. Additionally, version control and change management capabilities.
* **Monitoring and Triggering:** The object detection system must have the ability to monitor system performance and identify anomalies. In addition, the system must support mechanisms for alerting and notifying system administrators of problems. In addition, the capacity to handle retraining or model updates triggered by data drift, model performance, or other factors.

In general, an effective object detection system should be able to manage large amounts of data, select and implement the appropriate models, validate and test models, and deploy models in a variety of environments. In addition, it should be capable of automating the build and deployment processes and monitoring system performance.

**Non-Functional Requirements**

In addition to functional requirements, an object detection system must meet certain non-functional requirements to ensure its efficiency, reliability, collaboration, scalability, performance, and maintainability. Non-functional criteria fall into the following categories:

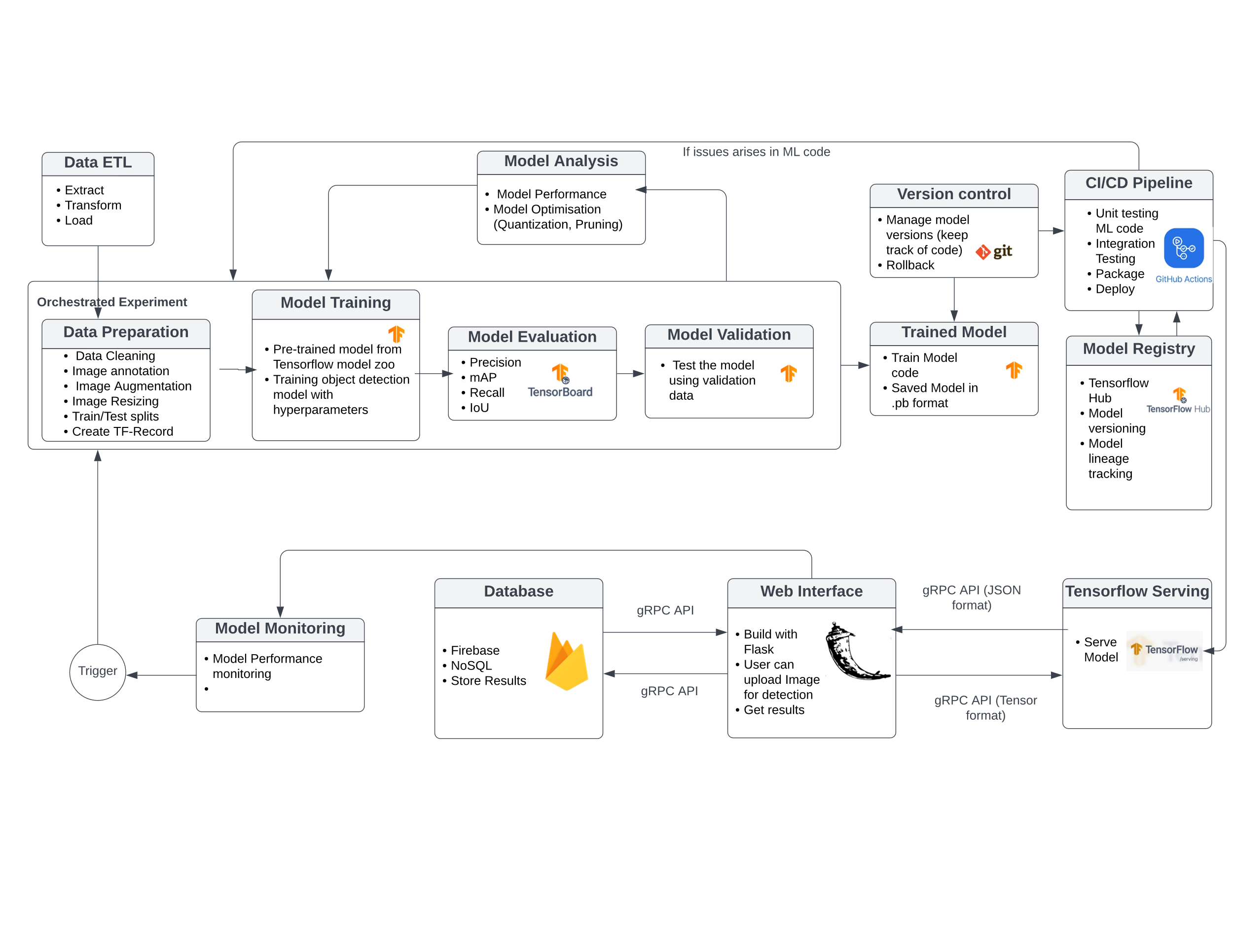
* **Data Engineering:** The system must be able to scale up or down to cater huge amounts of data. In addition, the system must ensure data quality and consistency to prevent training and inference errors.
* **Model Engineering:** The object detection system must ensure that models are accurate and reliable, and that they function well across a variety of data distributions. In addition, the system must strike a balance between model complexity and simplicity to guarantee that models are both fast and effective.
* **Model Testing and Validation:** The object detection system must ensure that all key use cases are thoroughly tested. In addition, the system must run tests and validations properly and efficiently in order to save time and resource needs. In addition, the system must provide techniques for validating model performance.
* **Model Deployment:** The object detection system must be capable of rapidly and efficiently deploying models. In addition, the system must maintain track of multiple model versions. In addition, the system must be able to scale up or down based on the amount of requests and data size.
* **CI/CD Pipelines:** The object detection system must be able to automate the entire pipeline in order to minimise manual intervention and error occurrence. In addition, the system must guarantee that the pipeline is continuous and that changes can be pushed rapidly and effectively. In addition, the system must maintain version control to ensure that all modifications are tracked and tested prior to deployment.
* **Monitoring and Triggering:** The object detection system must be able to continuously monitor performance and identify issues. In addition, the system must be able to respond promptly to issues and implement corrective measures.

Overall, for an object detection system to be scalable, dependable, and effective, it must satisfy both functional and non-functional criteria.

## System design with end-to-end functionality

The task has been assigned by a company to design a object detection system with end-to-end functionality. In this paper, I will use a scalable model server with GPU support to host a pre-trained TensorFlow 2 object detection model from the model zoo and a web interface for uploading images and get the predicted result. The purposed system will include the MLOPs process of hosting, monitoring, and replacing the ML model versions. In this system, will also provide infrastructure test plan, containerize and non-containerize solutions for the object detection system.

The purposed system will have development cycle of machine learning models and then MLOps practices for hosting, monitoring and managing machine learning models. Let’s discuss the end-to-end functionality of the proposed system with system diagram.



**Figure - 6** Object detection system design, non-containerize solution

Figure 6 demonstrates the end-to-end functionality of object detection system from data ETL (Extract, Transform, Load) to ML model monitoring without containerization. The system has two life cycles, development life cycle and ML operations. From the figure 6, system has data ETL phase which is not a part of development and operations cycle of machine learning pipeline. In data ETL phase, data can be collected from different source such as snowflake, databricks, mysql, mongo DB etc. For this object detection system, the bird’s data (in the form of images) has been collected from Liverpool John Moores University to train the machine learning model. This dataset will be the input of ML development life cycle.

The components of development life cycle are data preparation, model training, model evaluation and validation, and model analysis. On the other hand, ML operations has version control, CI/CD pipelines, model registry, model serving and model monitoring.

***Development of Model***

The development life cycle of machine learning model involves several steps from data preparation to model analysis.

**Data preparation:** This is the first step of machine learning model development. Once the data has been collected, it requires data cleaning, data labelling or annotation, data augmentation, splitting the data into train and test splits, and creating TF records (as I will be using Tensorflow for the model training). The purpose of data preparation is to improve the accuracy and efficiency, reduce overfitting and underfitting of the model.

**Model Training:** After data preparation, next step is to pass the data into the model for training. For the proposed object detection system, I will use pre-trained TensorFlow 2 object detection model named ‘Faster R-CNN Resnet101 with 1024x1024’ from the model zoo. This model is pre-trained on COCO 2017 dataset. I will use this model for the object detection of different bird’s species. The hardware resources used by this model will be a single gpu NVIDIA RTX3090. Once the model has been trained, it will saved in the saved folder with .pb extension.

**Model Evaluation and Validation:** Once the model will be trained, it will be evaluated using Tensorflow APIs and by checking the results in TensorBoard. The trained model will be validated on the unseen data to ensure that model has not poor generalization and overfitting. The performance of the model will be evaluated based on following metrics: precision, mAP, recall and IoU.

**Model Analysis:** The purpose of model analysis is to check where model is performing well and where the model performance can be improved by analyzing the predictions of the model. There are several techniques which can be used to enhance the performance of the model in this step. I will be using two techniques for the optimization of the model includes quantization and pruning. Quantization will be used to reduce the weights and activations in the network, which can reduce the memory consumption. Pruning will be used to remove the unnecessary connections between the neurons in the neural network, which can also enhance speed and efficiency without effecting the accuracy of the model.

***ML operations***

ML operations also known as MLOps, has been used in this system design for the productionalization of the ML model so he end-user can interact with the system and gets the predicted results. The ML operations involve several steps from versioning to monitoring of the model.

**Version Control:** To track the changes of the model code, data, and models over a period of time version control plays an important role. I will be using GitHub for the version control of the model. In the GitHub, I will create a repository so I will make the versions of model code including saved model files. I will also make branches on each version to keep track of the different functionalities. Every commit in the GitHub will create a version of the model, by maintaining these versions I can track my model development changes. One of the biggest advantages of version control is rollback. It is the process to revert back to the previous version, for example, if I am working on the new version and model performance of the newer version is poor then I can revert to previous version and can make it live.

**CI/CD pipeline:** CI/CD (Continuous integration or Continuous deployment) has the processes which involves the automation of building code or packaging, testing and deployment of machine learning model. The CI/CD pipeline will be connected to the model development process through version control. For the CI/CD pipeline, I will be using GitHub Actions provided by GitHub. As I will be using GitHub for the version control, connection between GitHub and GitHub Actions will be easy through some configurations. First, by using GitHub Actions runner, I will test my workflow locally before pushing to repository to catch errors. If I will find any errors than can go back to the development process of machine learning model. If there is no error in the workflow the code can be pushed to git repository. The CI/CD pipeline will be configured to pull the latest change in the code repository from GitHub to deploy new changes of the model. This can trigger the workflow to automatically deploy the model for serving (Tensorflow serving). I will also register the model by using GitHub Actions to model registry (Tensorflow hub) so it can be used later or reproduce.

**Model registry:** Model registry is basically a centralized repository where we can store, manage, reproduce, and share machine learning models. I will use Tensorflow Hub for the centralized repository for a trained ML model of an object detection system. The registration process of the trained model can be done by using Tensorflow Hub API from GitHub Actions. Additionally, I will also fetch the registered model from the registry to deploy the trained model by using the TensorFlow Hub API. The whole process can be automated by using GitHub Actions.

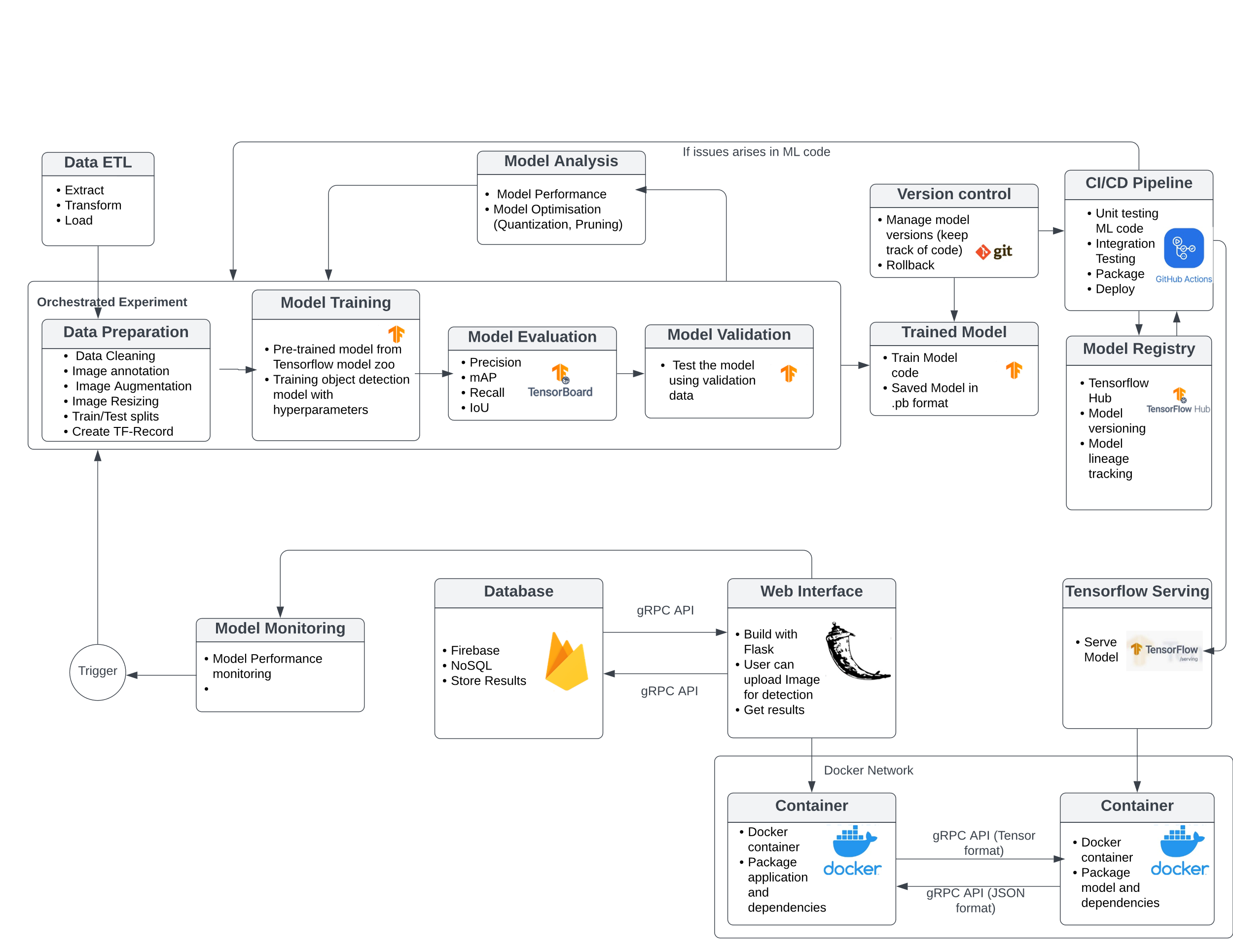
**TensorFlow Serving:** TensorFlow serving is an open-source software for serving the machine learning models into the production. For object detection system, Github Actions will be used to deploy the trained model to TensorFlow serving. TensorFlow serving supports various APIs like REST or gRPC to handle large number of requests. I will use gRPC API (7 times faster than REST API) for the communication between the webapp and model server on tensorflow serving. For the hardware resources, TensorFlow is designed to work with CPU, GPU and TPUs. For the object detection system, I will be using GPU because of the size and complexity of trained model.

**Web Interface:** The web interface will be developed for the user interaction so the end-user can upload the image to the model server for the prediction. The web interface will be built on flask, framework of python. In the web page there will be a web form having the option to upload the image. I will use gRPC API in python to send the image to model server. Before sending the image to model server for the processing (Tensorflow serving) it will be converted into tensor format because Tensorflow serving accepts this format for the processing. When the Tensorflow serving process the image by using respective model than it will send the results back to the flask app in the JSON format. Once the predicted results will be received from the Tensorflow it can be shown to the end-user through web page. I will also save the predicted results in the database (firebase) for the track of model performance. The results can be store and retrieve from the firebase by using gRPC APIs. The flask webapp will be hosted on Heroku.

**Database:** There are many types of databases like relational database (mysql), NoSQL database (firebase or mongo DB). I will be using firebase as it is a real-time database, means any change made to the database can be synced to clients. The firebase will be used to store the user information and predicted results of the model. The communication between firebase and flask app will be done through gRPC APIs.

**Model monitoring and Trigger:** For the model monitoring, I will develop another web page using the flask python framework. I will get the model predicted results and evaluation metrics from the model server to display on the webpage. The model performance will be monitored from the webpage, if there will be any issue occurs of model performance we can go back to development stage of machine learning model to resolve the issue.

***Containerize solution:***

******

**Figure - 7** Object detection system design, containerize solution

Figure 7 demonstrates the end-to-end functionality of object detection system from data ETL (Extract, Transform, Load) to ML model monitoring with containerization. The whole end-to-end functionality will be the same as discussed before in non-containerization solution, only difference will be the containerization of flask app and model server of TensorFlow serving using docker container. The purpose of using docker, it can be easily deployed and scale the model server in various environments, such as on-premises or in the cloud. As, I will create two docker containers, one for flask webapp and other for model server, so for the communication between two docker containers I will need to create a docker network. This will enable the flask webapp container and TensorFlow serving container to talk to each other using their container names as hostnames. Both the container will communicate with each other using gRPC port.

# Conclusion

In conclusion, this paper discusses the challenges of using machine learning models in production and provides the solution of adopting Machine Learning Operations (MLOps) practices. The advantages of using MLOps over manual machine learning has also been discussed in this study. It also discusses the requirement specifications and system design for an object detection system with end-to-end functionality. This paper suggests that the system can be scalable, dependable, and effective if it satisfies both functional and non-functional specifications. Overall, this study gives a detailed overview of MLOps practices and their application in developing and managing machine learning models in production environments.

# Future work

In the future work, the object detection system could incorporate several MLOps components such as feature store, AutoML and version control for flask based webapp. A feature store would consolidate the model's input features for training of the machine learning model. This will ease feature engineering, experimentation, model versioning, and reproducibility. AutoML will automate many of the laborious model development and optimization tasks. AutoML automatically generated and evaluated model variants and found the optimum model architecture and hyperparameters. This will improve performance even more than manual tuning and save time. Finally, adding version control to the Flask-based web application will do code management and tracking.

##### References

[1] D. Kreuzberger, N. Kühl, and S. Hirschl, “Machine Learning Operations (MLOps): Overview, Definition, and Architecture.”

[2] G. Symeonidis, E. Nerantzis, A. Kazakis, and G. A. Papakostas, “MLOps - Definitions, Tools and Challenges,” in *2022 IEEE 12th Annual Computing and Communication Workshop and Conference, CCWC 2022*, 2022, pp. 453–460. doi: 10.1109/CCWC54503.2022.9720902.

[3] M. Testi *et al.*, “MLOps: A Taxonomy and a Methodology,” *IEEE Access*, vol. 10. Institute of Electrical and Electronics Engineers Inc., pp. 63606–63618, 2022. doi: 10.1109/ACCESS.2022.3181730.

[4] L. Gomes Silva, “A Review on How Machine Learning Operations (MLOps) are Changing the Landscape of Machine Learning Development for Production.”

[5] Google Cloud Architecture Center, “MLOps: Continuous delivery and automation pipelines in machine learning”, Accessed: Feb. 19, 2023. [Online]. Available: https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

[6] Azure Machine Learning, “MLOps\_ Machine learning model management - Azure Machine Learning \_ Microsoft Learn”, Accessed: Feb. 20, 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/machine-learning/concept-model-management-and-deployment

[7] Hudgeon & Nichol, “Why Should You Use MLOps\_ - Amazon SageMaker”, Accessed: Feb. 20, 2023. [Online]. Available: https://docs.aws.amazon.com/sagemaker/latest/dg/sagemaker-projects-why.html

[8] G. Fursin, “The Collective Knowledge project: making ML models more portable and reproducible with open APIs, reusable best practices and MLOps,” Jun. 2020, [Online]. Available: http://arxiv.org/abs/2006.07161

[9] J. Bradley, “The Big Book of MLOps eBook A data-centric approach to establish and scale machine learning Introduction 3.”

[10] run.ai, “Machine Learning in the Cloud\_ Complete Guide [2022]”, Accessed: Feb. 28, 2023. [Online]. Available: https://www.run.ai/guides/machine-learning-in-the-cloud#:~:text=A%20cloud%20ML%20platform%20provides,to%20build%20ML%20algorithms%20faster.

[11] Opeyemi, “Data Science & Machine Learning in Containers - neptune.ai”, Accessed: Feb. 28, 2023. [Online]. Available: https://neptune.ai/blog/data-science-machine-learning-in-containers

[12] M. N. Chowdary, B. Sankeerth, C. K. Chowdary, and M. Gupta, “Accelerating the Machine Learning Model Deployment using MLOps,” in *Journal of Physics: Conference Series*, 2022, vol. 2327, no. 1. doi: 10.1088/1742-6596/2327/1/012027.

[13] Hashesh, “Version Control for ML Models\_ Why You Need It, What It Is, How To Implement It - neptune.ai”, Accessed: Feb. 28, 2023. [Online]. Available: https://neptune.ai/blog/version-control-for-ml-models

[14] A. Géron, “Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems,” 2017. [Online]. Available: http://oreilly.com/safari

[15] Ellingwood, “An Introduction to Continuous Integration, Delivery, and Deployment \_ DigitalOcean”, Accessed: Feb. 21, 2023. [Online]. Available: https://www.digitalocean.com/community/tutorials/an-introduction-to-continuous-integration-delivery-and-deployment

[16] S. Garg, P. Pundir, G. Rathee, P. K. Gupta, S. Garg, and S. Ahlawat, “On Continuous Integration / Continuous Delivery for Automated Deployment of Machine Learning Models using MLOps,” in *Proceedings - 2021 IEEE 4th International Conference on Artificial Intelligence and Knowledge Engineering, AIKE 2021*, 2021, pp. 25–28. doi: 10.1109/AIKE52691.2021.00010.

[17] Saha, “MLOps\_ Model Monitoring 101 \_ Towards Data Science”, Accessed: Feb. 28, 2023. [Online]. Available: https://towardsdatascience.com/mlops-model-monitoring-101-46de6a578e03

[18] ml-ops.org, “MLOps Principles”, Accessed: Feb. 21, 2023. [Online]. Available: https://ml-ops.org/content/mlops-principles

[19] A. Carqueja, J. Paulo Fernandes Co-Supervisor, and B. Cabral, “FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO On the Automation of Machine Learning Pipelines,” 2022.

[20] Awan, “17 Top MLOps Tools You Need to Know \_ DataCamp”, Accessed: Feb. 23, 2023. [Online]. Available: https://www.datacamp.com/blog/top-mlops-tools

[21] Valohai, “MLOps Platforms Compared Valohai”, Accessed: Feb. 23, 2023. [Online]. Available: https://valohai.com/mlops-platforms-compared/

[22] Menzli, “Best End-to-End MLOps Platforms\_ Leading Machine Learning Platforms That Every Data Scientist Need to Know - neptune.ai”, Accessed: Feb. 23, 2023. [Online]. Available: https://neptune.ai/blog/end-to-end-mlops-platforms

[23] Hill, “AWS Marketplace\_ DataRobot AI Cloud Platform for AWS”, Accessed: Feb. 25, 2023. [Online]. Available: https://aws.amazon.com/marketplace/pp/prodview-qmlu2m6haaf4i

[24] K. Salama and J. Kazmierczak, “Practitioners guide to MLOps: A framework for continuous delivery and automation of machine learning,” 2021.

[25] Morgunov, “The Life Cycle of a Machine Learning Project\_ What Are the Stages\_ - neptune.ai”, Accessed: Mar. 03, 2023. [Online]. Available: https://neptune.ai/blog/life-cycle-of-a-machine-learning-project

[26] Kubeflow, “Documentation \_ Kubeflow”, Accessed: Feb. 28, 2023. [Online]. Available: https://www.kubeflow.org/docs/

[27] MLflow, “MLflow Documentation — MLflow 2.1.1 documentation”, Accessed: Feb. 28, 2023. [Online]. Available: https://mlflow.org/docs/latest/index.html

[28] TensorFlow, “The TFX User Guide  \_  TensorFlow”, Accessed: Feb. 28, 2023. [Online]. Available: https://www.tensorflow.org/tfx/guide

[29] Saha, “Feature Stores\_ Components of a Data Science Factory [Guide] - neptune.ai”, Accessed: Feb. 28, 2023. [Online]. Available: https://neptune.ai/blog/feature-stores-components-of-a-data-science-factory-guide

[30] Tompkins, “Feature Stores and Why You Need Them — Advancing Analytics”, Accessed: Mar. 01, 2023. [Online]. Available: https://www.advancinganalytics.co.uk/blog/2022/8/1/feature-stores-and-why-you-need-them#:~:text=Feature%20Stores%2C%20as%20their%20name,but%20instead%20retrieve%20when%20needed

[31] Hudgeon & Nichol, “Amazon SageMaker Feature Store for machine learning (ML) – Amazon Web Services”, Accessed: Mar. 01, 2023. [Online]. Available: https://aws.amazon.com/sagemaker/feature-store/?sagemaker-data-wrangler-whats-new.sort-by=item.additionalFields.postDateTime&sagemaker-data-wrangler-whats-new.sort-order=desc

[32] Lawton, “Why and when to consider a feature store in machine learning \_ TechTarget”, Accessed: Mar. 01, 2023. [Online]. Available: https://www.techtarget.com/searchenterpriseai/feature/Why-and-when-to-consider-a-feature-store-in-machine-learning#:~:text=A%20feature%20store%20enables%20a,as%20part%20of%20their%20models.

[33] Feast, “Feast\_ Feature Store for Machine Learning”, Accessed: Mar. 01, 2023. [Online]. Available: https://feast.dev/

[34] Tecton, “Tecton \_ Feature Platform for Machine Learning”, Accessed: Mar. 01, 2023. [Online]. Available: https://www.tecton.ai/

[35] Hopsworks, “Hopsworks - The Python-Centric Feature Store”, Accessed: Mar. 01, 2023. [Online]. Available: https://www.hopsworks.ai/