ML Insights Hub: Real-Time Metrics, Model Storage, and Collaboration

Affan Mehmood
Zifan Zhang
Wenyu Fang
Farjad Akbar
Banin Sensha Shrestha(8447196)

October 10, 2024

Abstract

This project aims to provide a cloud based platform to streamline the development process of managing and tracking Machine learning experiments. This platform will provide developers and researchers with robust tools for storing deep learning model files, tracking training results, and logging key metrics. Currently ML engineers and researchers do the model experiment tracking and model storage manually and often on their local device, which is hard to share and explain with their teams, hence decreasing productivity. This platform will also offer advanced data visualization, customizable user interfaces, and resource management tools, including integrated GPU/CPU usage tracking. Collaboration features will allow teams to securely share results and insights.

1 Introduction

In recent years, the growth of deep learning and machine learning (ML) has led to increasingly complex models and experiments. As ML models thrived, managing and monitoring these experiments becomes critical for reproducibility, collaboration, and performance optimization. Researchers and developers need efficient ways to track model versions, hyperparameters, training results, and resource usage. Tools that enable transparency in the ML training process are essential to ensuring that models can be reliably deployed and continuously improved. Although several tools exist to assist with model tracking and experiment management, they often fall short in offering a comprehensive, flexible solution. The ability to efficiently store models, compare training runs, and visualize metrics is crucial, but existing platforms often lack advanced features like customizable dashboards, real-time dynamic metric logging, integrated explainable AI (XAI) tools, and resource management capabilities. These gaps create inefficiencies in model development pipelines, especially in large-scale projects involving teams.

Tools such as Weights and Biases and MLFlow provide essential services like model tracking, metric logging, and experiment visualization. However, they tend to be rigid in their design, lacking customizable interfaces and the ability to log new metrics dynamically during or after training. Additionally, these platforms generally do not offer native explainability features to help interpret models through techniques like SHAP or LIME, and their resource management capabilities are limited or nonexistent.

We introduce a tool that is user-friendly and easy to use, allowing ML engineers to efficiently track and manage their experiments without the need for manual logging. This tool provides seamless integration with their existing workflows, enabling the storage of models, tracking of hyperparameters, and visualization of key performance metrics in real-time. With a focus on transparency, collaboration, and resource optimization, the platform empowers engineers to streamline their development processes, improve reproducibility, and accelerate the deployment of high-performing models, all while maintaining secure, private accounts for each user.

1.1 Background

A number of basic understandings that are required to comprehend machine learning tracking tools will be covered in this part.

1.1.1 The Machine Learning Lifecycle

Different writers have covered the various stages of effectively utilising a machine learning model, each with their own nomenclature. This process is sometimes referred to as the "machine learning lifecycle." The machine learning lifecycle, according to the author [?], involves three primary stages as shown in **Figure ??**.

The pipeline's development is the initial stage. Tasks including data pretreatment, exploration, and visualisation are part of this iterative step. Model designs are chosen at this step, and models are trained using various configurations and hyperparameters. The author highlights that the pipeline, which can be used to create models from different datasets, is the phase's primary output rather than the model itself.

The pipeline that was previously created is used to train and validate the models that will be used for inference in the second step, which is called training. The last stage is known as inference. At this point, the model and data preparation are included in the prediction service, which uses user input to provide predictions. It is also possible to use these forecasts to enhance upcoming training procedures. The authors also point out that distinct teams usually handle these phases [?].

MLOps.drawio.png

Figure 1: Machine Learning Lifecycle

1.1.2 Experiment Tracking

According to the author [?], machine learning is an experimental discipline, and he draws comparisons between the empirical methods used in physics and chemistry and the process of finding an appropriate model. This perspective aligns with the results of interviews done in 2016 by the author [?] with practitioners of machine learning, where each of the seven participants stated that they relied on "basic trial and error." According to the author

[**?**], an experiment entails carrying out several runs or iterations in order to examine the impact of altering one or more independent variables on dependent variables. Furthermore, the author [**?**] highlight that data scientists frequently create hundreds of models, each of which represents the dependent variable in a run, before arriving at a result that is adequate.

It's interesting to note that, as the author [**?**] point out, experiment tracking tools can also be used throughout pipeline construction. Instead of creating a model in these situations, the objective is to create a training pipeline, which serves as the dependent variable. Thus, a run in this work is defined as a particular segment of an experiment that produces a training pipeline or a model. To determine the set of independent variables that result in the best dependent outcomes, each experiment consists of a compilation of these runs. Recognising that finding the optimal combination of independent variables is frequently unfeasible or prohibitively expensive in actual settings is crucial [**?**].

There are multiple methods for assessing a machine learning model's quality. One popular technique is to use a dataset that wasn't used for training to generate a statistic, like accuracy. However, nonfunctional quality metrics, such inference time, training time, or predictability, might also be important. It is quite useful to keep track of the experiments and their runs, considering the possibly large number of experiment runs involved.

In order to facilitate analysis and evaluation in the future, experiment tracking is the act of documenting all pertinent information about an experiment and its runs. While ”tracking” is generally linked with experiments, some tools employ terminology like ”log” or ”logger,” which are interchangeable in this sense. Tracking can be carried out manually or with the aid of specialised equipment.

Experiment tracking has many advantages. It streamlines the process of determining the optimal variables and examining which combinations have been tried and tested or need more research. This is especially helpful when tasks are completed in groups or when duties are transferred. Tracked trials may be easily compared with the correct tool, which can be very helpful when a model is put into production. Furthermore, the ability to replicate discoveries is a crucial advantage in research contexts. The use of an experiment tracking system in an organisation or project facilitates an organised method of handling the data produced by experimentation, guaranteeing accessibility irrespective of the experimenters.

2 Aims

The primary goal of this project is to provide machine learning (ML) and artificial intelligence (AI) developers with a comprehensive tool that centralizes the tracking, storing, and visualization of experiments, ensuring greater transparency, ease of use, and productivity. By addressing the challenges inherent in managing and reproducing experiments, the platform will streamline the development process. The specific aims of this project are:

- **Enhance Experiment Transparency** Machine learning experiments typically involve a variety of complex variables, including hyperparameters, model architectures, datasets, and external factors like hardware configurations. Without a dedicated tool, tracking these variables manually can lead to errors and inconsistency, especially in larger teams where multiple members may be running different experiments simultaneously. This project aims to centralize the storage and management of all these variables, ensuring that every experiment is meticulously documented. By providing a transparent log of all the factors involved in each experiment, it becomes easier to trace the exact conditions under which certain results were achieved. This, in turn, allows for improved reproducibility of experiments, a critical factor in ML and AI development, where being able to replicate successful runs is vital for deploying models into production.

- **Improve Team Productivity** One of the key pain points in ML experimentation is the difficulty of sharing results in real time. With numerous experiments being run concurrently, teams often struggle to maintain clarity over which models performed best or which configurations were most effective. This platform addresses this by offering real-time dashboards and visualizations of key metrics like accuracy, F1 scores, ROC curves, and resource usage. By making these results instantly visible and accessible to all team members, the platform encourages faster decision-making and collaboration. Additionally, the ability to track ongoing experiments and monitor performance trends over time helps avoid redundant work and keeps the team aligned toward shared goals, leading to higher overall productivity.
- **Simplify Experiment Comparisons** In machine learning development, comparing results from different experiments can be challenging without a structured system in place. This project aims to offer intuitive tools for side-by-side comparisons of different experiments, highlighting variations in performance metrics, resource usage, and configurations. By allowing developers to easily compare results from different hyperparameter settings, models, or hardware configurations, this feature helps identify patterns or anomalies that may have been missed otherwise. This improves the overall experimentation process by reducing trial and error and helping developers focus on the most promising approaches early on. The platform will also include advanced filters, allowing for in-depth analysis of different experiment runs, making it easier to evaluate the impact of individual variables.
- **Facilitate Resource Optimization** ML experiments are often computationally intensive, requiring significant resources in terms of GPU, CPU, and memory usage. These resource demands can vary significantly between experiments, and tracking them manually is time-consuming and inefficient. This platform will provide detailed logs and visualizations of resource usage, helping developers monitor how much computing power each experiment consumes. This information is vital for optimizing experiments—by identifying underutilized or overburdened hardware, developers can adjust resource allocation more efficiently. Additionally, by monitoring usage trends across multiple experiments, teams can better plan and forecast resource requirements, ensuring they don't encounter unexpected bottlenecks or overspend on computing infrastructure.
- **Streamline Collaboration and Sharing** In large or distributed teams, sharing ML experiment results, models, and insights can be cumbersome, particularly when collaborating across departments or with external partners. This platform addresses these challenges by enabling seamless and secure sharing of experiment data. The platform will include user permissions and access controls, ensuring that sensitive data is protected while still being easy to share within the appropriate team. The ability to package and share detailed reports—including model performance graphs, metrics, and experiment configurations—means that results can be shared not just internally, but also with stakeholders outside the development team. This streamlines the feedback process, fosters more collaboration, and helps external partners understand the model's performance, contributing to a more integrated development environment.

This project is positioned to become an essential tool for ML and AI development teams, offering features that not only streamline experiment tracking but also improve decision-making, collaboration, and resource efficiency. This will ultimately enhance the effectiveness of experimentation, reducing the time and effort required to deploy high-performing models.

3 Related Work:

Several platforms in market currently offer similar features to the ones we are building for our web platform, that stores and tracks deep learning models. So here we will discuss about some key competitors, their services and what innovation we will be adding into our project.:

3.1 Amazon SageMaker (AWS)

- **Cloud-Based Storage**: Stores models and data in Amazon S3.
- **Version Control for Models**: Keeps track of different versions of models.
- **Training Metrics Dashboard**: Has tools to monitor metrics like accuracy and loss during training.
- **Collaboration & Sharing**: Allows team members to work together and securely share models and data.
- **Resource Management**: Monitors hardware use and helps efficiently use GPUs and TPUs. [**?**]

Azure Machine Learning (Microsoft)

- **Cloud-Based Storage**: Uses Azure Blob Storage for storing models, data, and results.
- **Version Control**: Tracks and manages model versions through MLOps.
- **Training Metrics & Visualization**: Provides real-time monitoring and performance tracking with detailed visuals.
- **Collaboration**: Allows secure sharing and teamwork within the platform.
- **Resource Management**: Monitors and optimizes hardware usage to ensure smooth training. [**?**]

Google Cloud Vertex AI

- **Cloud-Based Storage**: Uses Google Cloud Storage for easy access to model files and training data.
- **Version Control for Models**: Tracks different versions of models and training runs.
- **Training Metrics Dashboard**: Monitors training progress and allows custom metrics.
- **Collaboration & Sharing**: Lets teams collaborate securely on shared resources.
- **Resource Management**: Manages hardware usage like GPUs and TPUs to save costs. [**?**]

Weights & Biases (WandB)

- **Cloud-Based Storage**: Provides cloud storage for models and datasets.
- **Version Control**: Automatically tracks different model versions and experiments.
- **Training Metrics Dashboard**: Monitors system metrics and training progress with real-time visuals.
- **Collaboration & Sharing**: Focuses on team collaboration with shared projects and real-time dashboards.
- **Resource Management**: Monitors hardware usage and integrates with cloud platforms for better tracking.

  [?]

Comet ML

- **Cloud-Based Storage**: Tracks and stores models, data, and training results in the cloud.
- **Version Control**: Automatically saves different model versions and experiment details.
- **Training Metrics Dashboard**: Provides detailed visuals for real-time tracking and comparison of experiments.
- **Collaboration**: Supports teamwork with commenting, sharing, and Slack integration.
- **Resource Management**: Manages hardware resources to improve training efficiency.

These platforms offer various tools for managing deep learning models, tracking training progress, and collaboration, which can be useful as we develop our project.
[?]

3.2 Similarities

Cloud-Based Storage

Like AWS SageMaker, Azure ML, Google Vertex AI, and WandB, our platform provides cloud-based storage for deep learning model files, checkpoints, and training data, which allows easy access and sharing.

Version Control for Models

Most platforms, including Azure ML, Comet ML, and WandB, offer model version control, allowing users to track and revert to previous versions of their machine learning models, just as we plan to implement in our project.

Training Metrics Dashboard

Platforms such as SageMaker, Comet ML, and WandB offer real-time dashboards to monitor key metrics like accuracy, loss, and other training results, much like our project.

Collaboration and Sharing

Competitors like WandB, Comet ML, and Azure ML emphasize collaboration with features to securely share models and experiment results within teams or externally, similar to the collaboration and sharing tools we're planning.

Resource Management

Monitoring hardware usage and optimizing GPU/CPU resources is a common feature in competitors like SageMaker, Azure, and Run, as well as in this project.

3.3 Differences and Extra Features in our Project:

Advanced Data Visualization

Our platform's emphasis on interactive and custom visualizations for training results and comparisons between models, experiments, and hyperparameters may offer deeper insights compared to more basic tools provided by some competitors like MLflow. Platforms like Comet ML and WandB also focus on visualization, but this system could stand out if it offers more customization or better comparison tools for models.

User Experience

Although competitors like WandB and Comet ML offer rich visualizations and interactive dashboards, we can focus on a more user-friendly and intuitive interface tailored for easy comparison between model versions and experiment runs. If we offer easier integration for beginners or more accessible collaboration features, that would be a distinctive advantage.

Custom Metrics

While many platforms provide predefined metrics tracking, our project's ability to support user-defined custom metrics for deep analysis may give it an edge over solutions like Azure ML or SageMaker, which may require more advanced configuration for custom metrics.

3.4 Conclusion

Our project is very competitive, offering many of the key features found in the leading platforms, but it can stand out with features like cost tracking, enhanced custom visualization tools, and a user-friendly experience tailored for deep learning models. These additional features, especially if well-implemented, could differentiate our platform from others like WandB, SageMaker, and Comet ML.

4 Methodology

4.1 Technical Implementation Plan

4.1.1 System Architecture Design

The system architecture of this project will be composed of a React.js frontend, FastAPI backend, and PostgreSQL database, all deployed on AWS cloud. The architecture will provide a structured and scalable framework that supports efficient ML experiment tracking and visualization.

- **Frontend:** React.js will be used to create a dynamic and responsive user interface. Material UI will be used for all components, including buttons, forms, and charts. This ensures a modern, clean design that is both functional and visually appealing. After logging in, users will be taken to their personalized dashboard, where they can view and manage their ML experiments, visualize data, and track metrics.
- **Backend:** FastAPI will act as the server, processing requests from the frontend and communicating with the PostgreSQL database. FastAPI is an ideal choice because it is lightweight, fast, and highly efficient for handling asynchronous requests. It also provides built-in support for data validation and documentation through OpenAPI, making it a suitable framework for a high-performance ML experiment management system.
- **Database:** PostgreSQL will be used as the database because ML experiments often involve structured data, such as model parameters, hyperparameters, and performance metrics, which fit well into relational table schemas. PostgreSQL offers robustness and reliability in managing such structured datasets and allows for complex queries.

4.1.2 API Development and Integration

The platform's core functionality will rely on a set of REST APIs that will allow users to upload and interact with their machine learning experiments in real time. Developers will write their training code in frameworks like TensorFlow or PyTorch, and integrate the platform's API calls directly into their code to store and retrieve models, metrics, and related data.

Once a user has trained their model (e.g., using model.fit() in TensorFlow), they will interact with the platform's REST API to upload the trained model and relevant metrics. The API will include endpoints for uploading models, logging metrics, and resource usage.

- **Upload the Model:** After completing the model training, the user will call the REST API to upload the trained model file along with necessary metadata (such as the experiment name, hyperparameters used, and the training dataset). The model file will be stored in a cloud location (e.g., S3 bucket), and the metadata will be recorded in the PostgreSQL database.
- **Log Metrics:** As training progresses, or after completing the experiment, the user can send performance metrics (such as accuracy, loss, F1 score) to the API for logging. The metrics will be linked to the specific experiment and stored in the database for easy retrieval and visualization later.

- **Track Resource Usage:** In parallel, resource usage (such as GPU, CPU, and memory utilization) will be tracked either by the user or automatically by the platform. These metrics will also be logged through the API to monitor and optimize resource usage across different experiments.
- **Retrieve and Visualize Results:** Once the data is uploaded, users can retrieve and visualize their models and experiment results through the platform's dashboard. For example, they may query the API for a list of previous experiments, compare models, or visualize training metrics via interactive charts.

4.1.3 Data Management and Storage

PostgreSQL will store all the critical data related to ML experiments, such as model files, hyperparameters, results, and system resource usage. The structured nature of the data makes PostgreSQL an ideal choice for organizing this information.

4.1.4 User Authentication and Data Security

User authentication will be handled through secure login systems using password hashing and tokens. Each user will have access only to their own experiments and results, ensuring privacy and security. FastAPI's OAuth2 or JWT token-based authentication will be used to manage user sessions securely. This ensures that only authenticated users can access their dashboards, experiment data, and models.

4.1.5 Front-End Development

The React.js frontend will offer an intuitive user interface. Material UI will serve as the design framework, providing pre-built components for forms, tables, and interactive elements. MUI Charts will be used to display visualizations of training metrics like accuracy, F1 scores, and system resource usage. Users will log into the app and be presented with a dashboard that allows them to manage their experiments, view metrics, and compare results across experiments. The interface will be highly interactive, with customizable features that allow users to filter and compare experiments based on different variables.

4.1.6 Performance and Resource Monitoring

The platform will track and log the CPU, GPU, and memory usage for each experiment, providing insights into how resources are being utilized. PostgreSQL will store logs of resource usage, while FastAPI will retrieve this data and send it to the frontend for visualization. This will allow users to monitor the efficiency of their experiments and adjust their usage accordingly.

4.1.7 Collaboration Features and Data Sharing

The platform will include collaboration features that allow users to share their experiment data with team members securely. Users will be able to generate reports, compare models, and share insights without exposing sensitive data.

- **Data Sharing:** Permissions and access control will be handled by the platform, ensuring that users can securely share their results without giving full access to other users' experiments.

4.1.8 Deployment Plan

The platform will be deployed on AWS using modern cloud practices, ensuring that it is scalable, secure, and available.

4.2 Project Challenges

4.2.1 Scalability of the system

Our project will need to handle a large number of network data requests from multiple users, especially in large-scale machine learning experiments, where model training and data storage can balloon rapidly. Traditional self-built single-server architectures cannot cope with this load, so a highly scalable architecture design is required. We can use a service like Auto Scaling offered by AWS to automatically adjust the number of servers based on changes in traffic and workloads. Elastic Load Balancing (ELB) can be configured to dynamically distribute traffic, ensuring that the platform remains stable during peak periods. In order to better ensure the stable operation of the system, we split the system into multiple microservices (such as front end, back end, database, model storage, etc.), each service runs independently and can be independently extended. Each service can be individually scaled based on its resource requirements. You can also use caching technologies such as Redis or Memcached to reduce database query pressure and improve system response speed.

4.2.2 Real-time performance tracking

For machine learning, real-time tracking of the training progress and performance of machine learning models is crucial for model optimization. However, if data and charts are updated frequently, it can put additional strain on the server and increase latency. FastAPI supports asynchronous processing of HTTP requests, allowing us to process model training in the background while the front end can quickly respond to other user requests. Users can view training progress in real time without blocking the server. After the startup, the real-time detection of the time does not lose the signal and reduces the waste of resources for the server. It can maintain a long connection with the front-end through WebSocket. This avoids the waste of resources caused by HTTP polling and enables efficient real-time data updates. Of course, real-time data can be fragmented, or only updated when the part has changed. It's like having to upload and update over a long period of time instead of sending all the data every second.

4.2.3 Data security and privacy

On cloud platforms, protecting users' experimental data, models, and sensitive information is a top priority, especially in collaborative environments where concurrent access and data sharing by multiple users increases security risks. All model files and data stored in the cloud should be encrypted using AES-256 encryption to ensure security in data transmission and storage. AWS S3 provides a default server-side encryption option that you can enable to encrypt stored model files.

4.2.4 Integration with existing workflows

Most machine learning developers already use tools (e.g. TensorFlow, PyTorch) to manage their training process, so platforms need to integrate seamlessly into these existing workflows to avoid interrupting the development process that users are accustomed to. Develop REST apis for users to call in their code, providing features such as model uploading, metric logging, resource usage monitoring, and more. By calling these apis in training scripts, developers can automatically send data to the platform without manually recording and uploading it. The API was designed with popular machine learning frameworks such as TensorFlow's model.fit() or PyTorch's train() in mind, providing a simplified interface that reduces developer effort. Or develop a plugin or SDK that integrates directly into a framework like TensorFlow or PyTorch. This allows developers to quickly connect to the platform without modifying existing code.

4.2.5 Resource optimization

Large deep learning models often require large amounts of GPU, CPU, and memory resources. If not monitored, resources may be wasted or training may fail due to insufficient resources. This requires resource monitoring, using nvidia's NVIdia-SMI tool to monitor GPU usage in real time, and using Linux's own system monitoring tools (such as top and htop) to monitor CPU and memory usage. This data is uploaded to the platform via apis for users to view in real time. You can even provide suggestions for resource optimization based on historical data, such as reducing GPU idle time, dynamically adjusting GPU/CPU allocation, and even predicting possible resource bottlenecks. While helping users to monitor and adjust in real time, it also provides some reliable resources for the algorithm of Kanban to give optimization suggestions to ensure the accuracy of the optimization suggestions given

4.2.6 Customizable user interface

Each user has different needs, just as some users may be more concerned about the accuracy of the model, while others may be more concerned about resource consumption. Therefore, it is necessary to provide a flexible user interface that allows users to customize their workspace according to their personal preferences. With React.js and Material UI, users can freely drag and drop, add or remove components (such as training curves, metrics charts, etc.) to create personalized dashboards. The customized Settings of the user can be saved in the back-end database, and the user's Settings can be restored at the next login. Provides users with a variety of visual configuration options, such as changing the chart type (line chart, bar chart, pie chart, etc.), selecting different metrics to display, and allowing users to customize how often the chart is updated. Multi-language support is achieved through react internationalization plug-ins such as React-i18Next to meet the needs of users in different regions and languages.

4.2.7 Teamwork and data sharing

In large-scale machine learning projects, teamwork is very important. How to effectively share experimental data and models among team members, while keeping the data safe, is a key challenge. Introduce a detailed rights management system into the platform, allowing refined access controls when experimental data, models, and results are shared within the team. Each project or experiment can be set up to be public, private, or accessible only to certain members. Users can generate full reports, including model performance charts, experiment configurations, and metrics, which can be shared with team members or external partners as PDFS or online links, simplifying the collaboration process. To avoid data conflicts between team members, the platform can provide version control for experimental models and data. Each time an experiment or model is modified, a new version is generated, and team members can look back to see the historical version.

5 Project Team

Our team consists of four individuals with diverse backgrounds and skills:

1. **Affan Mehmood:** Experienced developer with a strong background in machine learning and programming. Responsible for implementing core ML experiment tracking features and data analysis tools.
2. **Zifan Zhang:** Developer with skills in frontend programming. In charge of React.js implementation and user interface design.
3. **Wenyu Fang:** Team leader without a strong technical background, but eager to learn and contribute to the project management and documentation aspects.
4. **Farjad Akbar:** Data scientist with experience in data analysis and visualization. Responsible for designing and implementing data visualization features and assisting with backend development.
5. **Banin Sensha Shrestha:** Experienced full stack developer with strong background in system integration and Machine Learning. Responsible for implementing the features in React as well as implementing API end points. Also, responsible for integration between Frontend and Backend.

6 Plan and Timeline

The tasks are scheduled based on each team member's strengths:

- Weeks 1-2: Zifan will set up the basic frontend structure, while Affan begins work on the core ML features.
- Weeks 3-4: Affan and Banin continues with ML implementation, Farjad starts on data visualization features.
- Weeks 5-6: Farjad and Banin focuses on backend development integrating with UI with assistance from Affan.
- Weeks 7-8: All team members collaborate on integration and testing, with Wenyu taking the lead on documentation.
- Weeks 9-10: Final refinements, Wenyu compiles the final documentation, and the team prepares for launch.

6.1 Work Breakdown Structure (WBS)

1. Develop Cloud-Based Platform for ML Experiment Management
   - (a) Set up development environment
     - i. Install necessary tools and frameworks
     - ii. Set up version control system
   - (b) Design system architecture
     - i. Create high-level system design
     - ii. Define component interactions
   - (c) Develop frontend (React.js)
     - i. Create UI conponents
     - ii. Implement basic components
     - iii. Develop user dashboard
   - (d) Develop backend (FastAPI)
     - i. Set up FastAPI server
     - ii. Implement core API endpoints
     - iii. Integrate with database
2. Implement Robust ML Model Storage and Retrieval System
   - (a) Set up cloud storage AWS S3
     - i. Configure S3 buckets
     - ii. Implement access controls
   - (b) Design database schema for metadata
     - i. Define tables and relationships
     - ii. Optimize for query performance
   - (c) Develop model upload functionality
     - i. Create file upload interface
     - ii. Implement server-side storage logic
   - (d) Implement model retrieval system
     - i. Develop search functionality
     - ii. Create download mechanism
3. Create Real-Time Metrics Tracking and Visualization Tools
   - (a) Design metrics logging system
     - i. Define key metrics to track
     - ii. Create data structures for metrics
   - (b) Implement real-time data streaming
     - i. Set up WebSocket connections
     - ii. Develop server-side streaming logic
   - (c) Create visualization components
     - i. Implement charts and graphs
     - ii. Develop interactive dashboards
   - (d) Integrate metrics tracking with ML workflows

        i. Create hooks for popular ML frameworks
        ii. Implement automatic metric logging

4. Implement Resource Usage Monitoring and Optimization
    (a) Develop resource tracking system
        i. Implement GPU/CPU usage monitoring
        ii. Create memory usage tracking
    (b) Design resource visualization tools
        i. Create real-time usage graphs
        ii. Implement historical usage views
    (c) Develop resource optimization suggestions
        i. Implement usage analysis algorithms
        ii. Create optimization recommendation system
5. Develop Collaboration and Sharing Features
    (a) Implement user management system
        i. Create user roles and permissions
        ii. Develop user authentication
    (b) Design sharing mechanisms
        i. Implement project sharing functionality
        ii. Create team collaboration tools
    (c) Develop notification system
        i. Implement real-time notifications
        ii. Create email notification system
6. Ensure Platform Security and Data Privacy
    (a) Implement data encryption
        i. Encrypt data at rest
        ii. Implement secure data transmission
    (b) Develop access control system
        i. Implement role-based access control
        ii. Create audit logging system
    (c) Conduct security testing
        i. Perform penetration testing
        ii. Conduct security code review
7. Create User-Friendly Interface and Documentation
    (a) Refine user interface
        i. Conduct usability testing
        ii. Implement UI improvements based on feedback
    (b) Develop user documentation
        i. Write user manual
        ii. Create video tutorials
    (c) Implement in-app guidance
        i. Develop tooltips and help text
        ii. Create onboarding tour for new users

7 Outcomes

The expected outcomes of this project include:

- A fully functional platform for managing ML experiments: A professional-grade React.js [?] web application paired with a FastApi [?] server and PostgreSQL [?] database, offering robust experiment tracking, model storage, and performance visualization capabilities for developers.
- Efficient deployment on AWS Cloud: The platform will be securely deployed on AWS [?] cloud infrastructure, ensuring scalability, reliability, and high availability for users managing a high volume of ML experiments.
- REST API integration for seamless interaction with ML code: A fully documented REST API will allow users to easily integrate the platform into their existing ML workflows, enabling the upload of models, metrics, and other critical data directly from their codebase, similar to popular platforms like Weights and Biases [?] and [?].
- User-specific data privacy and security: Each user will have a dedicated, secure account that ensures only they have access to their experiments, models, and results. This enhances data protection and provides peace of mind when working with proprietary or sensitive data.
- Enhanced productivity and collaboration: The platform will streamline experiment tracking and result sharing, significantly improving team productivity by making it easier to compare, replicate, and share ML experiments across a team or organization.
- Increased resource optimization: The ability to monitor GPU/CPU and memory usage across experiments will allow users to optimize hardware resources effectively, reducing costs and improving overall computational efficiency.
- Valuable skills and portfolio enhancements for team members: For developers working on this project, the experience will offer insights into full-stack development, cloud deployment, and machine learning experiment management. This will result in a valuable portfolio piece demonstrating expertise in cutting-edge technologies such as FastApi, React.js, PostgreSQL, and AWS.