Final Group Project X (Set 5)

Final Project Report Submission

Karamveer Singh Ghai, Karan Dhanoa, Harshpreet Bajwa, Lovish Khanna, Yuvraj, Pradiumn Prashar, Dhairya Ashara

University of the Fraser valley,

Comp 371-ON1

Dr. Youry Khmelevsky

December 5,2023

Final Report: Optimizing XGBoost Training Efficiency

Executive Summary:

Our project, focused on optimizing the training efficiency of the XGBoost machine learning system, has been undertaken by our dedicated team under the adept leadership of our Scrum Master. We have navigated through a comprehensive project life cycle, characterized by a comprehensive approach to refinement and innovation.

With a firm understanding of the project's purpose established, we undertook a rigorous elaboration phase, refining requirements with precision. The resultant groundwork provides a pattern for the subsequent design and implementation phases, ensuring alignment with overarching goals.

The resulting phase is one of the most important milestones of our approach, involving the creation of a robust architectural framework. This, in turn, facilitated communication among team members and stakeholders. UML diagrams serve as visual aids for design principles, comprehending a coherent and scalable system. Throughout this report, we will explore various phases of our project.

Then there is the transition to the implementation phase, which comprises a variety of roles—Scrum Master, Head Developer, Code Developers, and Designers—combining expertise to bring conceptual designs to life. Throughout the project, we will explain various aspects, exploring the layers of our approach and showcasing tangible outcomes of our collaborative efforts. From conceptualization to execution, each phase of this initiative is underpinned by a commitment to excellence in the realm of machine learning.

Objective

The primary objective of the project is to improve and optimize the training efficiency of the XGBoost machine learning system. In simpler terms, the project aims to make the process of training XGBoost— a popular machine learning algorithm—faster, more effective, and scalable. This optimization is essential for providing data scientists with enhanced tools and insights, enabling them to use XGBoost more efficiently in real-world applications. The ultimate goal is to deliver a final program that significantly improves the overall performance and usability of XGBoost in the context of machine learning tasks.

Team Structure

The team is well structured with its specific roles to ensure efficient project execution. Karamveer serves as a Scrum master, facilitating and coaching the team while ensuring adherence to the scrum framework. Our head developer Harshpreet, leads the development team, working closely with developers to guide and support their efforts. Yuvraj and lovish are code developers and are responsible developing and delivering high-quality code. The team also includes Designers—Dhairya, Karan, and Pradiumn—responsible for designing diagrams and coordinating with the project's main head. This structured team setup covers

leadership, development, coding, and design, ensuring a comprehensive approach to achieving the project's objectives.

Project phases:

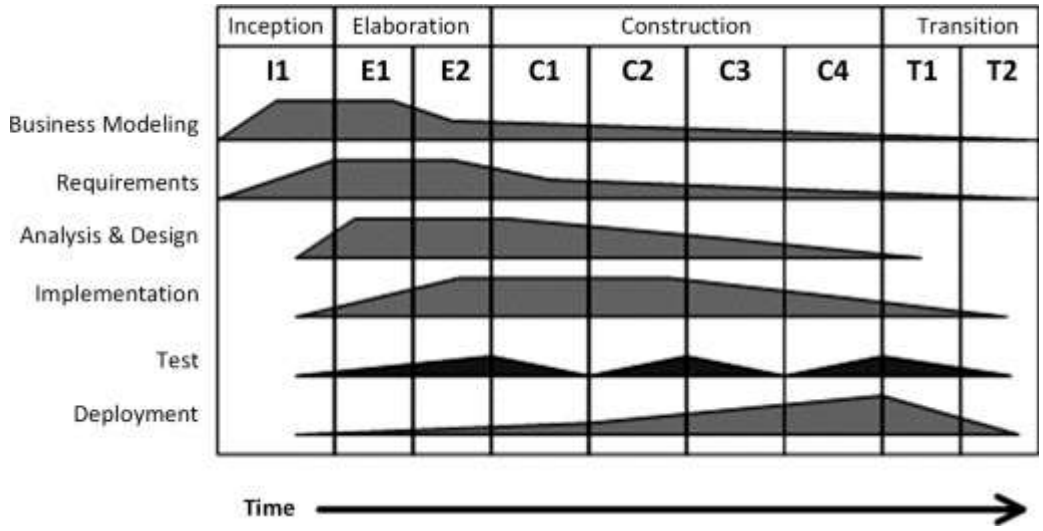

Figure 1: All UP phases stages

Inception phase:

The inception phases starts the project by establishing the fundamental vision , scope, and objectives tailored specifically for optimizing the XGBoost machine learning system. The primary goal is to articulate a clear purpose for enhancing the training efficiency of XGBoost in real-world applications. At this stage we created GANTT Chart, also developing the domain model and little bit start with the code development so atleast we install the XGboost machine learning system.

Elaboration:

During the Elaboration Phase, the project transitions into a more detailed planning stage. The primary focus was on the refinement of the project, specifying datasets and laying out the ground work for subsequent design and implementation phase. We conducted a lot of meetings during this time and then we made activity and use case diagram and we then presented our first milestone report. During this time we didn’t listed any design patterns, we only focused on higher level diagrams.

Designing phase:

Phase1:

During this phase we created multiple diagrams and also constructed a major chunk of our project. But during the presentation of our milestone report 2, We were careless and didn’t implement design patterns in our code as well as in diagrams, also we made use of jupyter

notebook, which was then pointed out by the professor during the milestone report2 submission.

Phase2:

We quickly took professor advice and then started conducted extensive and regular meetings with the professor. We started identifying design pattern and with the guidance of the professor, we figured it out updated use case diagram and activity diagram as well as incorporating design patterns such as creator, information expert, pure fabrication, and MVC pattern. We also then refined our most of our models and then moved on to the make changes and refine our main code that is the implementation part.

Implementation phase:

This is the phase where there is a transition from planning to action. It also involves the actual development of XGBoost training process. We made an interface and try to develop a MVC Pattern implementation in our code also incorporating our 9 grasp patterns that is being explained in the course textbook. We then increased our number of meetings and then and start working on extensive implementation our code and also getting feedback from the instructor on side. We also completely avoided the use of jupyter notebook and make sure that we focus on quality code.

Drawbacks:

Throught our all phases our team has given us excellent performance but we seemed to have a time problem to schedule our meetings, this was then lead to the communication gap among team members. Also we faced problems with the hardware as we don’t have high end computers to accelerate out XGBoost.

Diagrams:

Use Case Diagram:

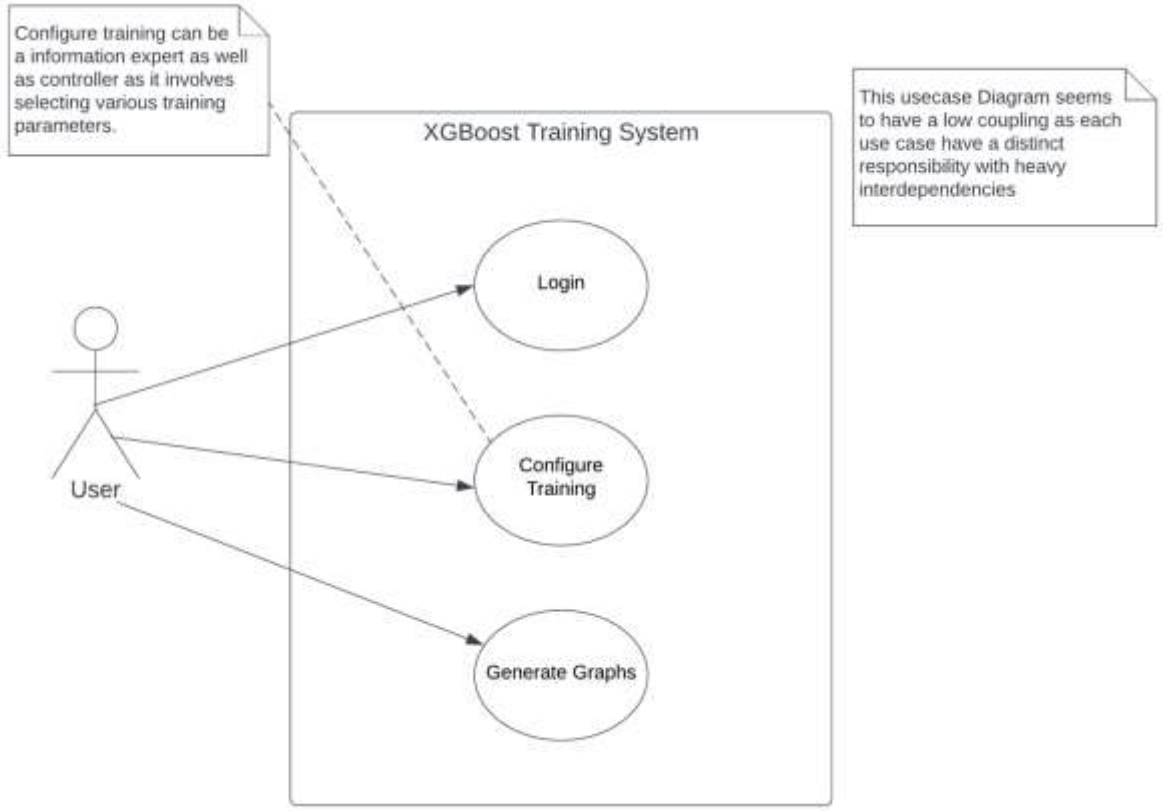

Figure2: Use Case Diagram

We identified 3 use cases in which there was a login use case, configure Training use case and Generate Graph use cases. Here we also identified some grasp patterns. That includes:

In this design, we used “configure training” task that does two jobs. First, it acts like a controller that manages the settings needed to train the model. Secondly, it acts like a creator by making it examples for training. Following a simple rule low coupling and high cohesion, which makes each job is separate, and they only talk to each other when needed. This helps keep things organized and working well. Another important thing is the 'Configure Training' task knows a lot about what's needed for training and the best ways to make the model. We made it this way to follow a rule called Information Expert. So, this task is like a helpful leader in our system, taking care of settings, creating examples, and knowing a lot about how to train the model effectively.

Domain Model:

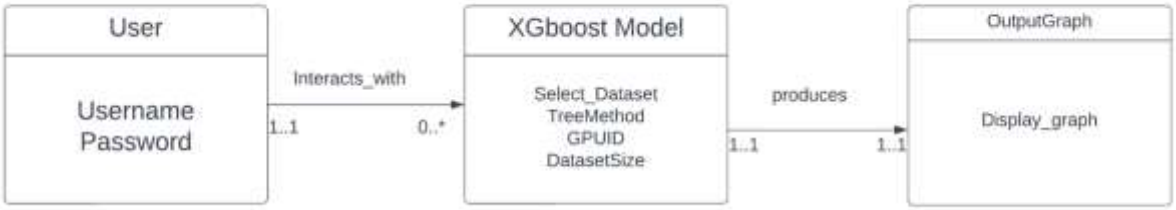

Figure3: Domain model for XGBoost System

In figure 3, a user interacts with an XGBoost model. The user provides the necessary information for the model, such as the dataset and tree method, before training the model. Once trained, the model generates an output graph, which is then displayed to the user. This scenario outlines a series of steps for a person using the XGBoost model. First, they log in by entering their username, password, and other important details. Then, they interact with the model by choosing a dataset and deciding how the model should make decisions (tree method). After that, they start the model's training process by specifying things like the GPU to use and the size of the dataset. Once the model learns from the data, it creates a picture (output graph). Finally, the person looks at this graph to understand how well the model learned from the data. This diagram illustrates each step clearly, making it easy for the person to follow how they interact with the XGBoost model and see the final results.

Class Diagram

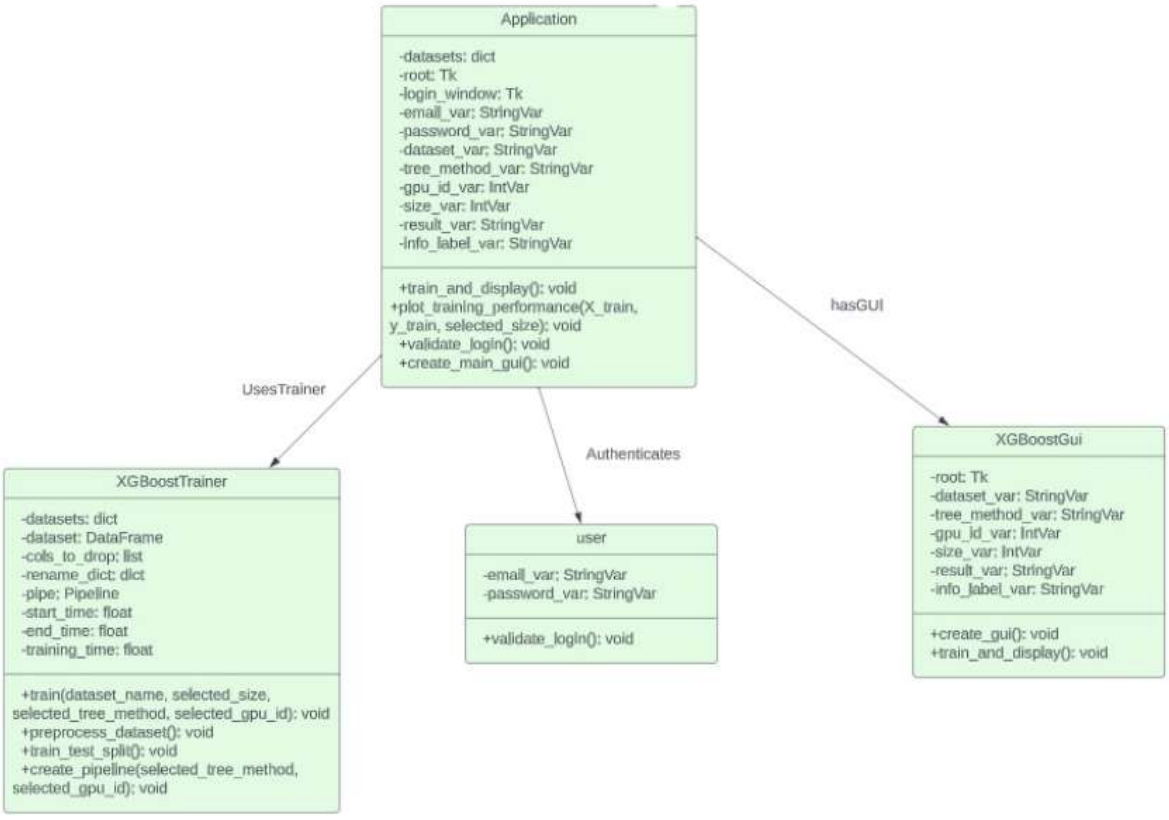

Figure4: Class Diagram

This Diagram shows how different part of the system work and connect together, imagine it as building with blocks. There are two important ways these blocks connect. One way is like blocks helping or using each other—this is called Association. For example, the "XGBoostTrainer" block helps the "Application" block. The other way is Realization, which is like a big block teaching a small block how to do things. In our picture, the "XGBoostGui" block learns from the "Application" block. By looking at these connections, we can easily understand how all the blocks in the computer program fit together.

MVC Pattern Design

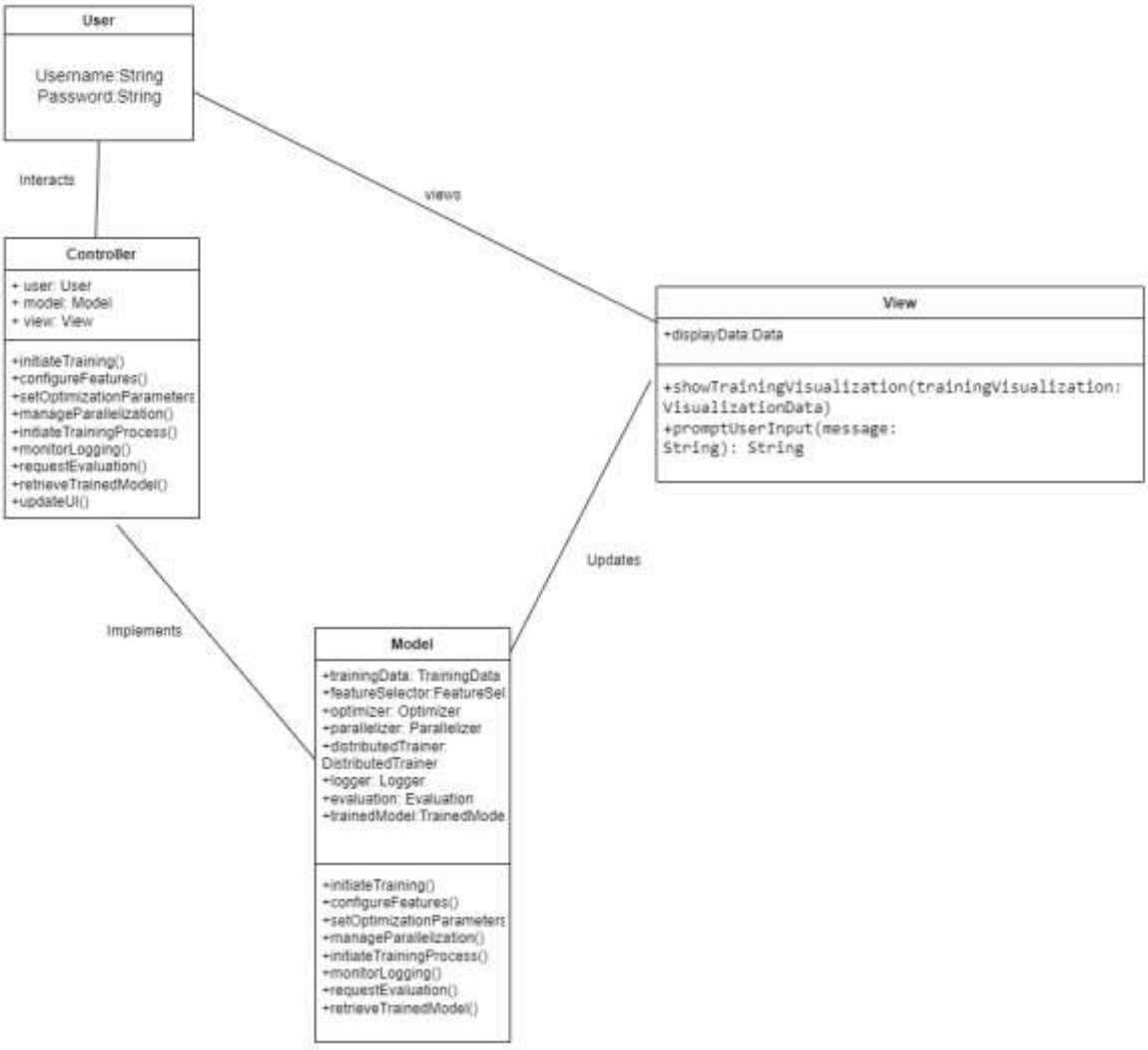

Figure 5: MVC Design pattern

We have explicitly shown the architectural pattern by providing the clear separation of its classes. Here you can see that the user is interacting with the controller for its inputs. Then

the controller would acts as an intermediatory by gathering the information from the model that is here the information expert and then the model has its view which is for the user.

Sequence Diagram

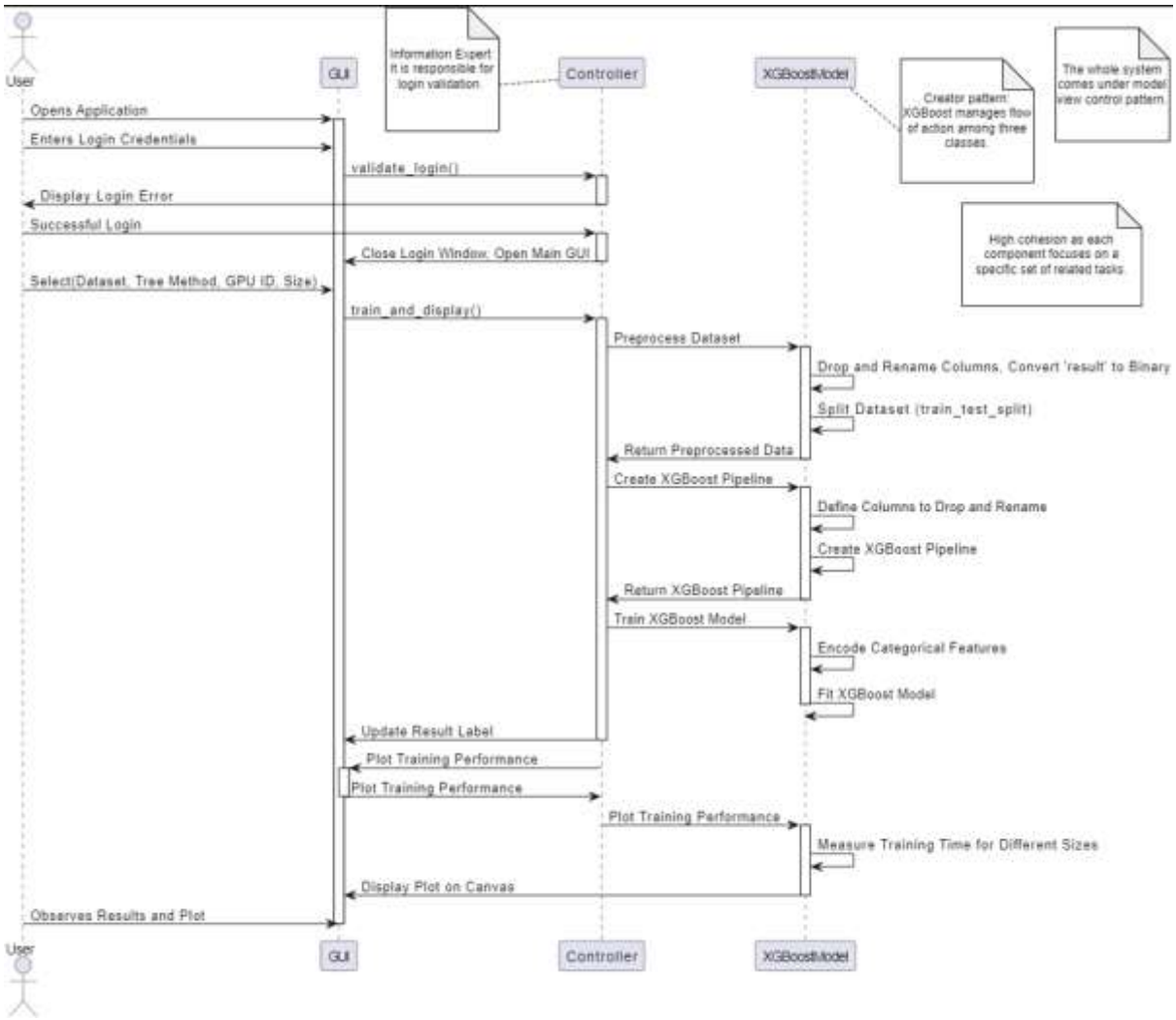

Figure6: Sequence Diagram

We have discussed 9 grasp patterns as follows:

- Information Expert:
  - Observation: The Controller holds the responsibility of managing interactions, making it the central decision-maker.
  - Reasoning: The Controller possesses the most information about the overall system behavior.
- Creator:
  - It is the XGBoost model class that manages the interaction between multiple classes.

- Low Coupling:
  - Observation: Components interact through the Controller with minimal direct dependencies.
  - Reasoning: This reduces coupling, making components more independent and modular.
- High Cohesion:
  - Observation: Responsibilities like loading data and training are appropriately distributed between the Model and Controller.
  - Reasoning: Each component focuses on specific, closely related tasks, ensuring high cohesion.
- Pure Fabrication:
  - Observation: The sequence diagram doesn't explicitly feature the creation of pure fabrication objects.
  - Reasoning: The context doesn't necessitate the introduction of fabricated objects.
- Indirection:
  - Observation: The Controller acts as an intermediary, directing interactions between components.
  - Reasoning: Indirection is applied as the Controller that has responsibilities and messages.
- Polymorphism:
  - Observation: Polymorphism is not evident in the sequence diagram.
  - Reasoning: The diagram doesn't involve class hierarchies or method overriding.
- Protected Variations:
  - Observation: The protection of variations isn't explicitly highlighted in the sequence diagram.
  - Reasoning: While encapsulation is practiced, safeguarding variations is not the primary focus in this scenario.

Activity Diagram:

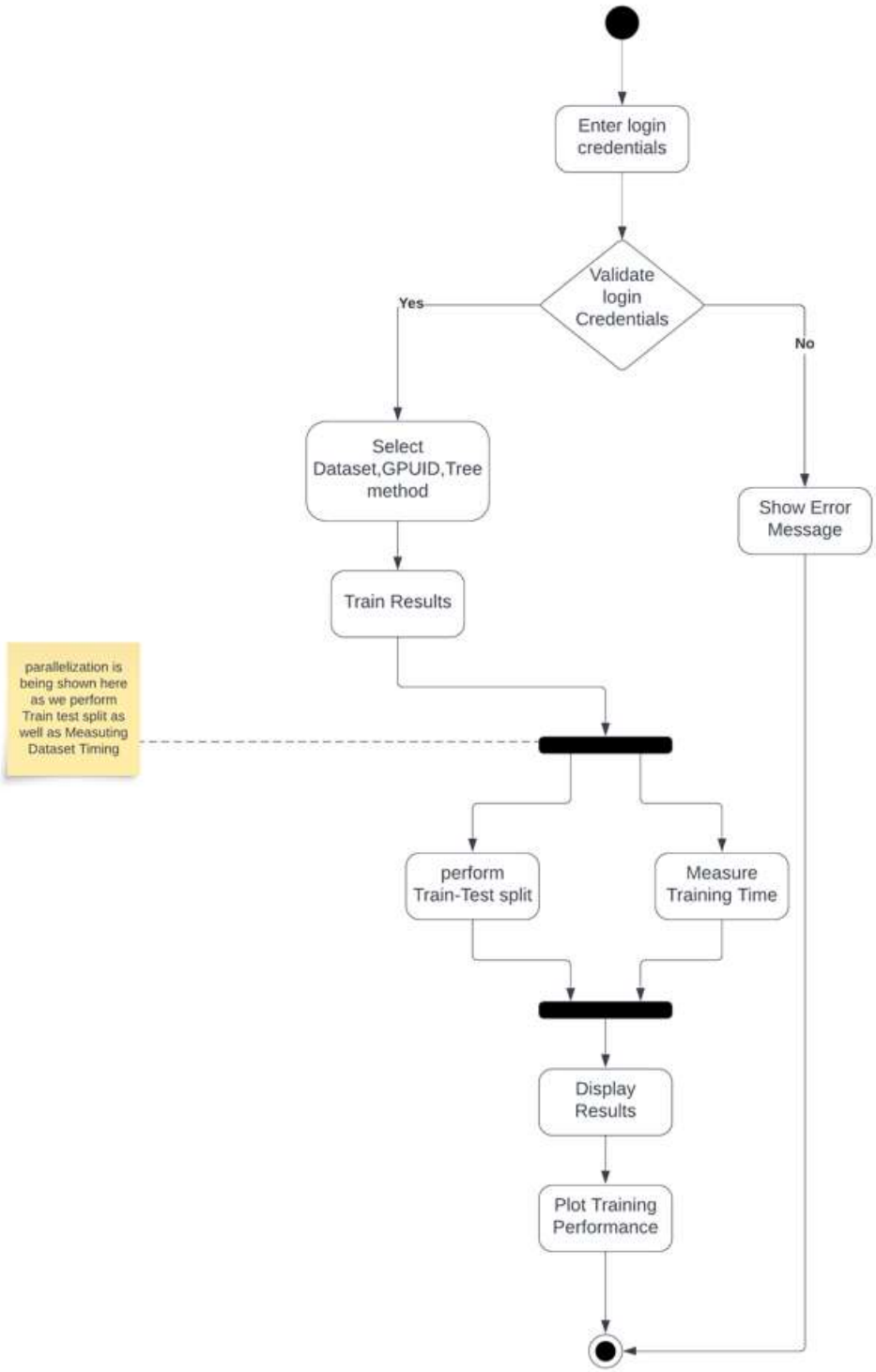

Figure 7: Activity Diagram

This diagram has shown a sense of parallelisation where it goes to performs Train-Test split and measure the training time.

Code:

We have provided the code along with the submission. But we have provided the output for the code.

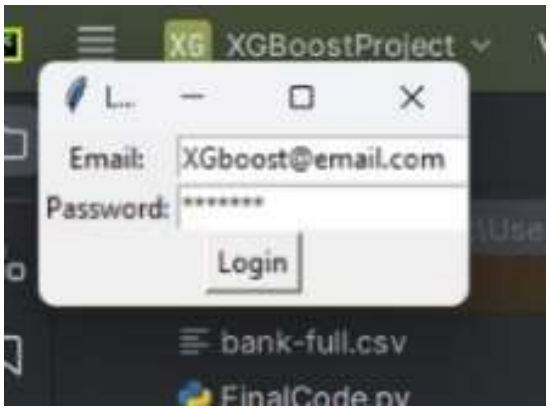

Figure8: Output of login Interface

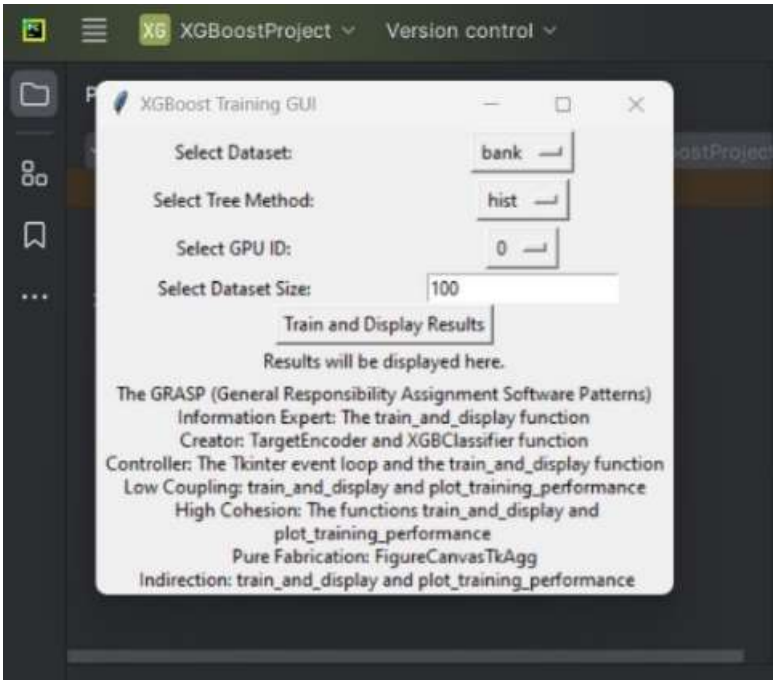

Figure9: Output of intermediatory interface

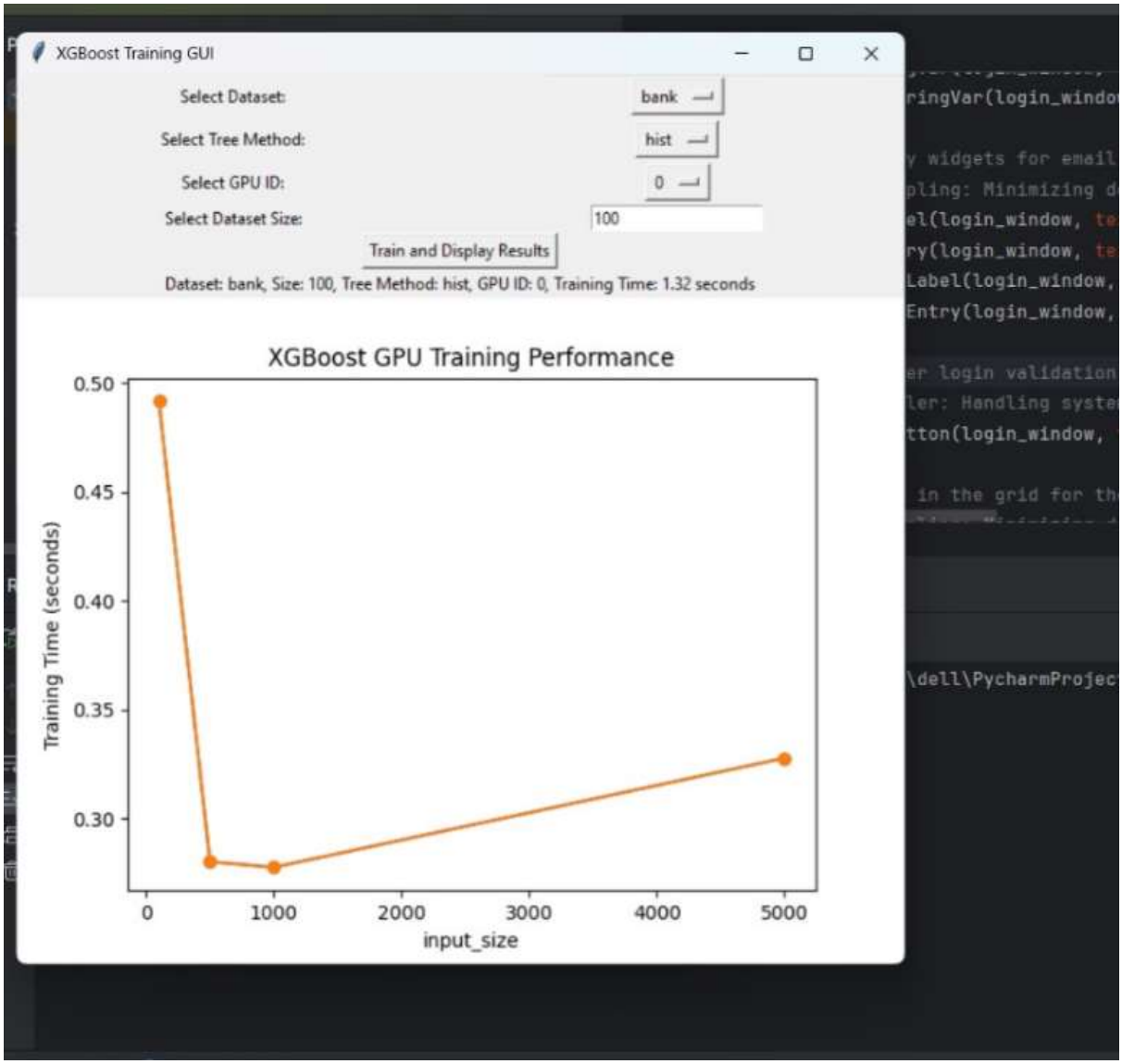

Figure 10: Ouput of Graph interface.

Conclusions

We successfully implemented the GUI application for training the XGBoost classifier on different datasets and displaying the training performance. We also provided the selection of datasets and users can choose one for training an XGBoost classifier. Also we used XGBoost classifier along with a pipeline that includes target encoding. The training process is triggered by a button press. After that it generates a graph showing training time for different input sizes. Before accessing the main GUI, users need to log in with a specific email and password. We also incorporated various design patterns that is displayed in the information label. In summary we have used interface for training an XGBoost classifier, visualizing training performance, and incorporating software design patterns to enhance maintainability and flexibility.

References:

- Larman, C. (2004) Applying UML and patterns: An introduction to object-oriented analysis and design and iterative development. Addison Wesley Professional.
- Metheny, M. (2017). Comparison of federal and international security certification standards. In Federal Cloud Computing (Second Edition): The Definitive Guide for Cloud Service Providers (pp. 211-237).
- Brownlee, J. (2016). XGBoost With Python: Gradient Boosted Trees with XGBoost and scikitlearn. Wade, C., & Glynn, K. (2020). Hands-On Gradient Boosting with XGBoost and scikit-learn:Perform accessible ... Packt Publishing.
- Ramraj, S., Nishant Uzir, S., Sunil, R., & Shatadeep Banerjee. (2016). Experimenting XGBoost Algorithm for Prediction and Classification of Different Datasets. International Journal of Control Theory and Applications, 9(40), ISSN: 0974–5572.