Auto-ML Pipeline Builder

Project: Software Engineering (DLMCSPSE01)
Yash Manohar Chaudhari
Student ID: 4252695

Project Overview

This project involves developing a web-based application that provides end users with an organized interface for configuring and managing automated data processing and machine learning pipelines. By dividing down pipeline creation into different steps and components, the approach is intended to minimize technical complexity.

The goal of software engineering in this project is to create a system architecture that is both modular and maintainable. The program divides data administration, User interface and backend processing into discrete parts making it simple to expand and maintain the system.

The projects overall goal is to produce an adaptable and expandable software system that follows software engineering techniques and offers a strong basis for additional development in subsequent stages.

Focus and Scope

- Conceptual planning provides way to active program development in phase 2.
- This phases primary goal was to provide strong technical foundation by establishing the development environmenet, refining requirements and putting the intended archotecture into practice.
- Software engineering techniques were prioritirized over focusing more on machine learning.

Development Environment Setup

- Configuration of the backend development environment.
- Initialization of the frontend project structure.
- Establishment of version control with GitHub.
- Structured the project repository for incremental development.

Frameworks and Tools with Raionale

- React.Js – I selected React.Js because it provides a component based, adaptable user interface which is perfecr for data driven interfaces. It is future proof, scalable and maintainable.

- FastAPI – Selected based on its great performance, built in type checking, automatic documentation and support for a clean API structure. Perfect for backend engineering.

- During development PyCaret and Scikit-Learn are used to make machine learning jobs easier. They make it easier to focus on software architecture while lowering implementation efforts.

- SQLite – SQLite meets server overhead requirements for local development. If deployment happens later the architecture is still flexible.

- Docker – Docker guarantees consistency and gets the project ready for eventual cloud deployment, even while deployment is not currently planned.

- GitHub – Offers appropriate versioning, release control and traceability which are essential components for software engineering.

Refined Funtional Requirements

- Before accepting the dataset, the system must verify the file’s size and format.
- Users will be able to setup preprocessing settings.
- The list of available tasks variables for the machine learning tasks will be displayed on the system.
- The pipeline will manage feature scaling, categorical encoding and missing values.
- The Auto-ML module will evaluate models and choose the top performing model.
- Both the resulting Python script and the trained model will be available to download for the users.
- For reproducibility the system will record processing stages.
- The user interface will show the training and preprocessing progress in real-time.

Non-Functional Requirements

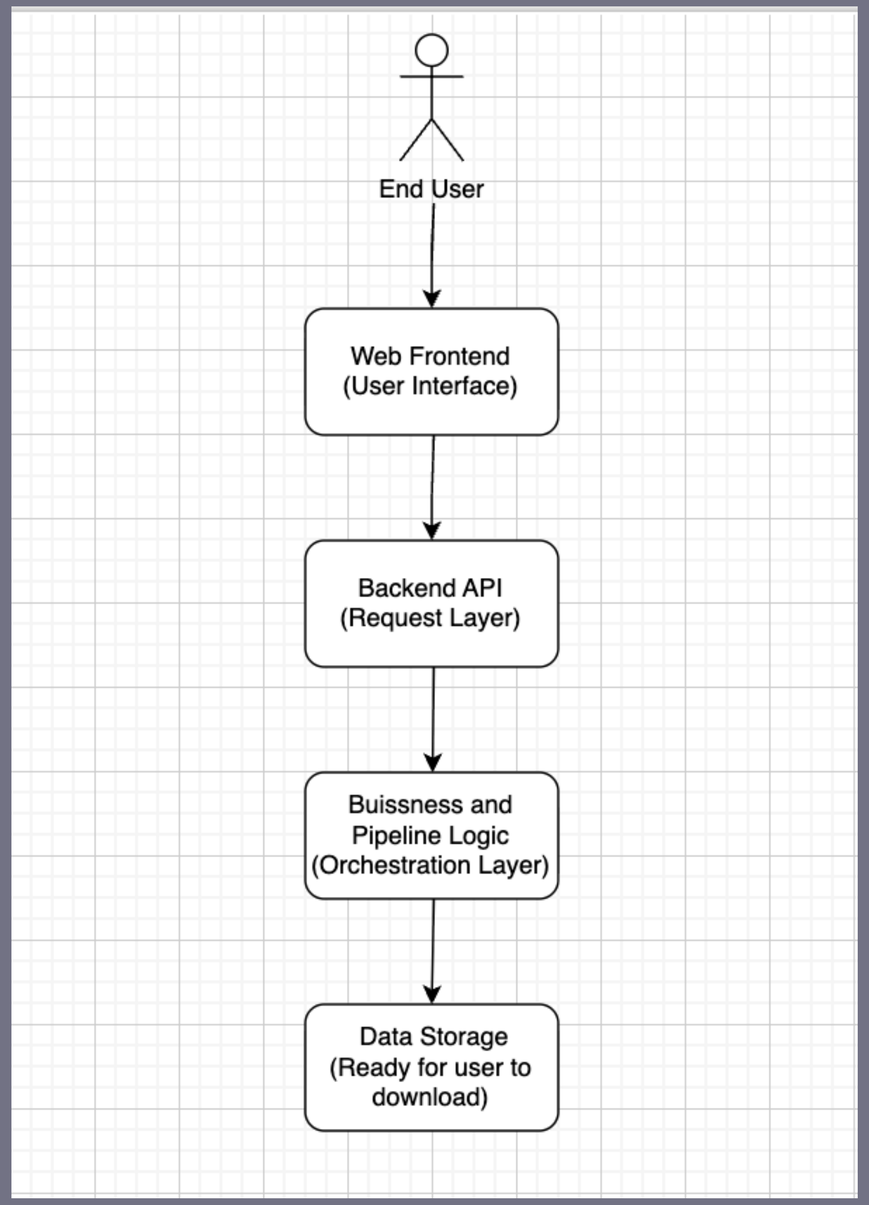
- For non-ML requests the backend api should reply within 300 miliseconds.
- For future deployment readiness, all essential functions must function consistently in Docker.
- A consiste error handling method must be followed by system logs.
- At least 30% of the backend functionality must be covered by backend unit tests.
- Medium-sized datasets must be handled by the system without going over its memory capacity.

Architecture Overview

The System architecture should remain modular and consist -

- React Frontend : User interface for uploading datasets and visualizing outcomes.
- FastAPI Backend : Manages the ML pipeline creation, data processing and result delivery.
- ML Engine : Scikit-Learn and PyCaret for automatic model comparison.
- Storage Layer : SQLite for structured metadata and a local filesystem.
- Docker is the development environment for reproducibility (Can be used for deployment in future.)

System Design



The architecture of the system is multilayered. The end users commnuicate with the system by using the web-based frontend. RESTful API is used to connect the frontend and the backend. Request processing, building buisness logic and pipeline orchestration is managed by the backend server. A storage layer is then used to store the end-to-end pipeline created for the users project which can be later downloaded by the users.

Design and Implementation

Based on the conceptual framework based on the Phase 1, I began implementing the Auto-ML Pipeline Builder. Taking your feedback for the Phase 1 into consideration, I started to focus more on professional software engineering practices rather than focusing on machine learning. I created and utilized the development environment, which included Python based Machine Learning libraries, Docker for environment consistency, Fast API for Backend, React.js for Frontend and Github for version and release control.

I refined the initial thoughts from phase 1 and turned them into tangible program elements. I worked on the first iterations of the backend modules, which included dataset handling, preprocessing, evaluation logic and exporting artifacts. In order to simplify simple user interaction for dataset upload and configuration, I also started to work on the user interface. For future implementation, I improved functional and non-functional requirements during this phase.

(Cont…) Design and Implementation

As I proceed with the system implementation, I am going to document important design choices, including using the PyCaret to speed up the pipeline generation and Fast API for typed and modular backend development. To make the source code easier to understand and maintain, I will further organize it in a modular fashion. In Phase 3 testing will begin after all essential features have been fully implemented and stabalized.

Challenges and Learning

- Careful division of duties among components was necessary to design a modular architecture while maintaining the system's adaptability for future additions.
- It was difficult to incorporate machine learning features without detracting from professional software engineering techniques.
- It was necessary to assess compatibility and steer clear of tight coupling in order to integrate third-party libraries in a manner consistent with the system architecture.
- Architectural and design descriptions have to be updated frequently in order to maintain clear and consistent documentation while implementation was changing.
- Careful organization and standardized commenting techniques were necessary to ensure code readability and maintainability during early development.

Conclusion

- The project effectively moved from conceptual planning to active software development in Phase 2.
- Based on improved criteria, the development environment and fundamental architectural framework were created.
- Important design choices were made and recorded to promote extension, maintainability, and modularity.
- The anticipated software architecture is in line with the initial implementation of key system components.
- Professional software engineering techniques were routinely used, including version control, documentation, and code readability.
- The project is under development and will be ready for thorough testing, functionality completeness, and Phase 3's final assessment.