Qual-Bench AI

Feb 10, 2025

Author:
Kabire Akbari
Jacob Autus
Ji Guo
Lucas Gomes

Special Thanks to:
Haotao Wang

Table of Contents

[Qual-Bench AI]

Version 1.0

February 10, 2025

1. Introduction

Qual-Bench AI aims to be an artificial intelligence benchmarking website using the Qualcomm AI Hub[1]. The website seeks to evaluate and compare all state-of-the-art (SOTA) object detection models while also providing a comparison of the subsequent models. The platform aims to visualize relevant performance metrics of each object detection AI model available on AI Hub using Qualcomm hardware and multiple runtimes (ONNX, TFLite, Qualcomm AI Engine Direct).

The objective of this project is to assist developers of applications and AI researchers in understanding the strengths and weaknesses of different object detection models, allowing them to make informed decisions when selecting models for their applications. By integrating Qualcomm AI Hub's APIs, this platform will streamline the benchmarking process and provide a user-friendly interface to visualize and compare models and choose the best model for their application and device use case.

The scope of the project will be to benchmark all object detection models on Qualcomm AI Hub trained on the COCO 2017 validation dataset [2] of 5,000 images and provide the following metrics: latency, inference time (ms/image), compute units, estimated peak memory usage and mAP (mean average precision). The benchmark will include all valid device and runtime configurations for each model.

2. Software Requirements

2.1 User Interface Requirements

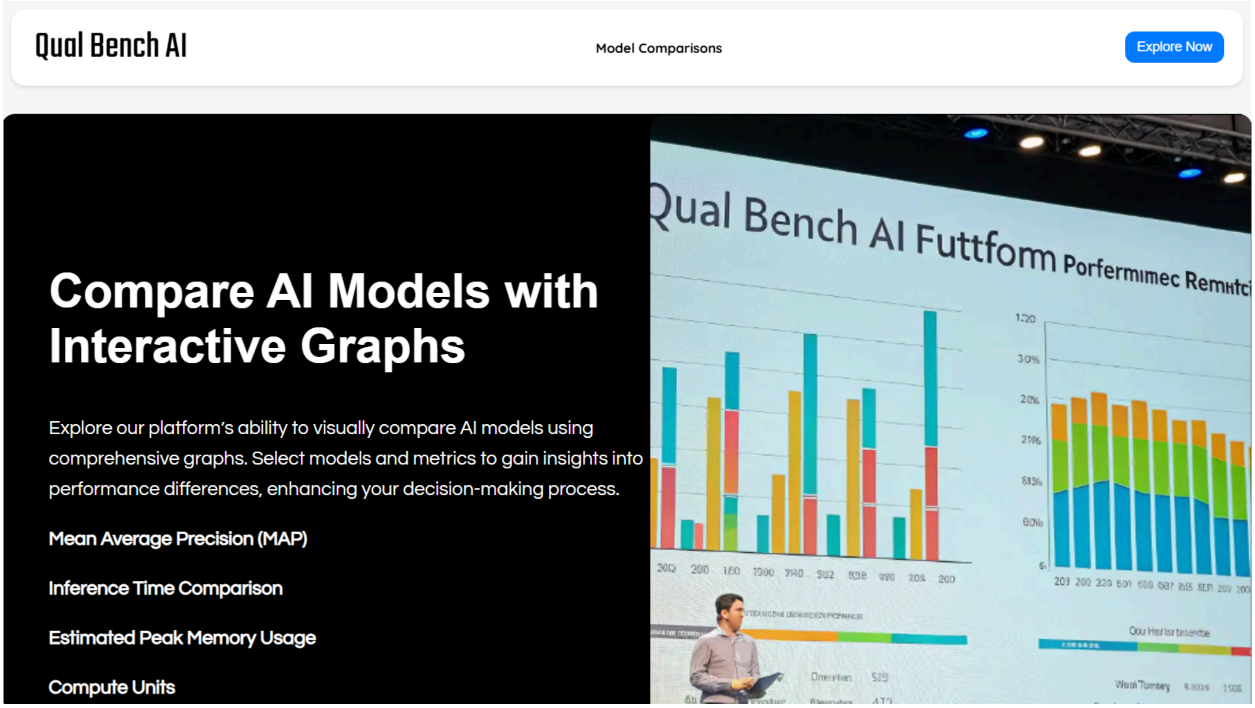
2.11 Home Interface:

The "Home Page" function provides users with an intuitive and modern interface for navigating the system.

The home page will display the navigation menu on the top of the screen. Meanwhile demonstrating example images of using our website and explanations of our website down below.

Source of Input OR Destination of Output	homePage.jsx
Required Screen Formats/Organization	Web Application
Report Layouts	The interface will dynamically adjust based on user input, directing them to relevant pages.
Menu Structures	- **Top Navigation Bar:** Provides access to key sections: **"Qual Bench AI"** (Home) **"Model Comparisons"** **"Explore Now"**

Error/Other Messages	- "Page not found. Please check the URL." - "Navigation error. Please try again."
Function Keys	- "Explore Now" button - "Model Comparisons" button - "Qual Bench AI"(Home) button



Compare AI Models with Interactive Graphs

Explore our platform's ability to visually compare AI models using comprehensive graphs. Select models and metrics to gain insights into performance differences, enhancing your decision-making process.

Mean Average Precision (MAP)

Inference Time Comparison

Estimated Peak Memory Usage

Compute Units

Figure 2.11: Main page

2.12 Group Comparison Interface:

The "Group Comparison" function provides users with a clear and interactive interface for comparing multiple models based on inference time or other selected metrics.

The page is designed to emphasize data visualization allowing users to interact with graphs through dropdown selections or directly touch on it.

Source of Input OR Destination of Output	groupComparison.jsx
Required Screen Formats/Organization	Web Application
Report Layouts	The interface dynamically updates based on user input, changing the chart visualization based on selected comparison metrics. Each bar on the graph can display its detailed information when the user interacts with it.
Menu Structures	- **Top Navigation Bar:** Provides access to key sections:

	"Qual Bench AI" (Home) **"Model Comparisons"** **"Explore Now"** - **Main Content Area:** Displays the comparison graph and drop down box for input controls. **"Drop down metrics selection"** **"Detailed Comparison"** **"Bars Graph"** **"Bars"**
Error/Other Messages	- "Invalid Selection." – If an unsupported metric is chosen. - "Data Unavailable." – If no data exists for the selected model.
Function Keys	- "Explore Now" button - "Model Comparisons" button - "Qual Bench AI"(Home) button - Metrics Dropdown - "Detailed Comparison" Button

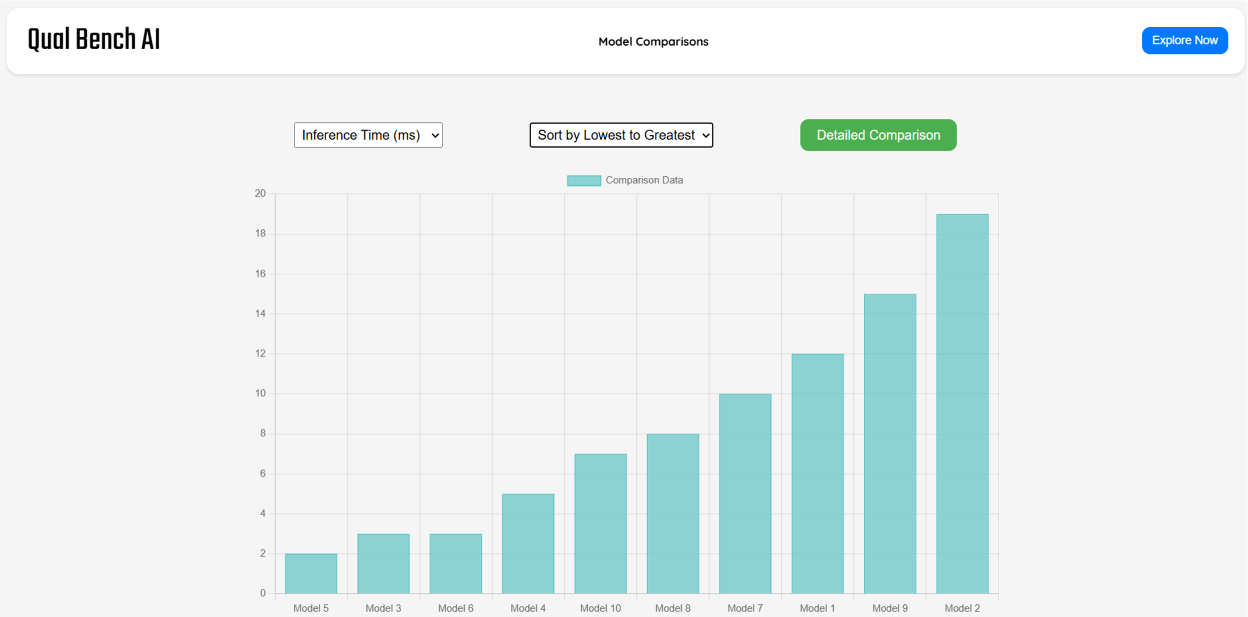


Figure 2.12: Model comparison page - group comparison

2.13 Detailed Comparison Interface:

The "Detailed Comparison" function allows users to compare two AI models across multiple performance metrics using visually distinct bar charts.

The interface is designed for clarity and ease of comparison, helping users quickly assess differences between selected models.

Source of Input OR	groupComparison.jsx

Destination of Output	
Required Screen Formats/Organization	Web Application
Report Layouts	The interface dynamically updates based on user input, changing the chart visualization based on selected comparison metrics. Each bar on the graph can display its detailed information when the user interacts with it.
Menu Structures	- **Top Navigation Bar:** Provides access to key sections: **"Qual Bench AI"** (Home) **"Model Comparisons"** **"Explore Now"** - **Main Content Area:** Displays the comparison graph and drop down box for input controls. **"Drop down metrics selection"** **"Detailed Comparison"** **"Bars Graph"** **"Bars"** **"Comparison data section"**
Error/Other Messages	- "Data Unavailable." – If no data exists for the selected model.
Function Keys	- "Explore Now" button - "Model Comparisons" button - "Qual Bench AI"(Home) button - Model Dropdown

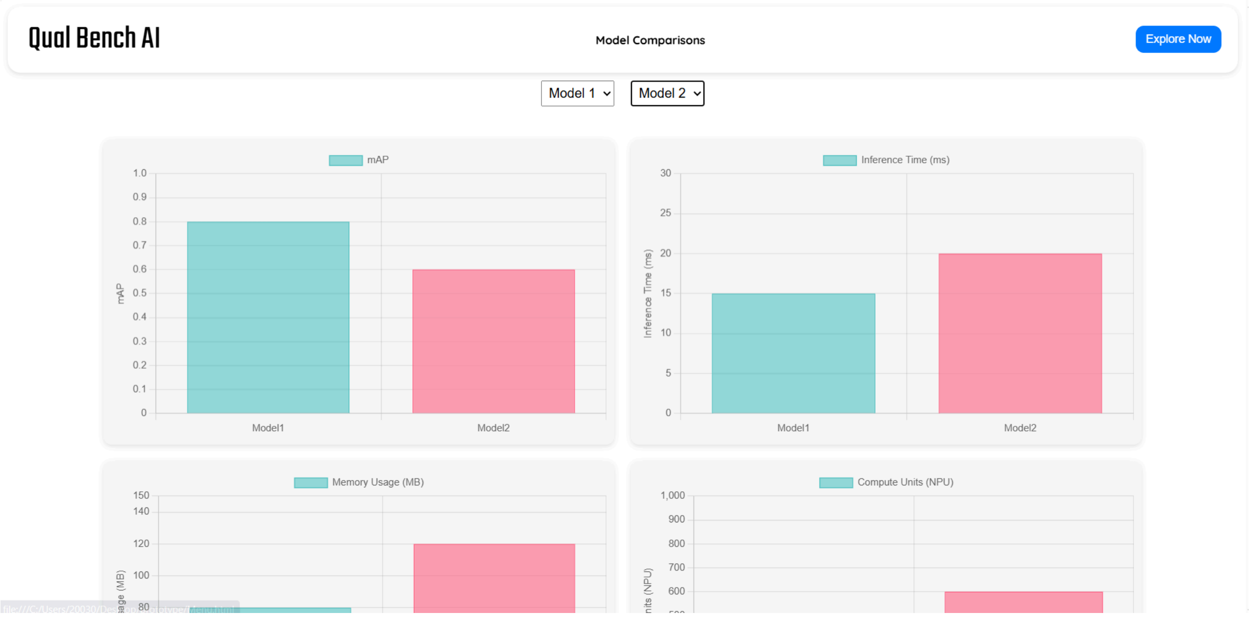
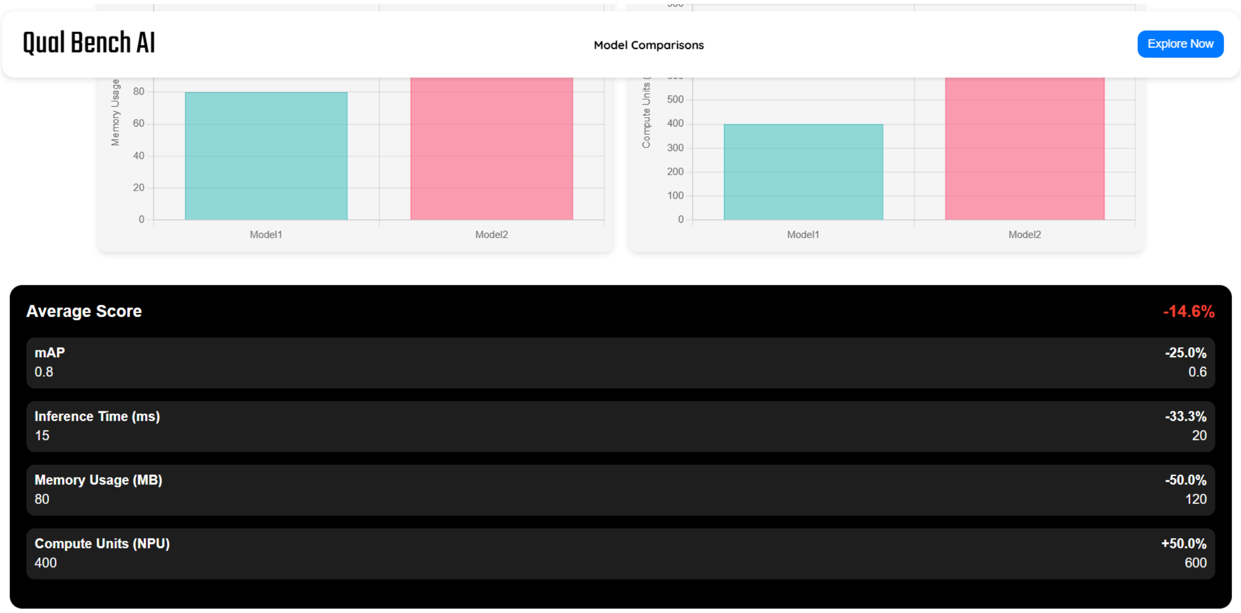


Figure 2.13a: Model comparison page - detailed comparison



Average Score -14.6%

mAP
0.8
-25.0%
0.6

Inference Time (ms)
15
-33.3%
20

Memory Usage (MB)
80
-50.0%
120

Compute Units (NPU)
400
+50.0%
600

Figure 2.13b: Model comparison page - detailed comparison

2.2 Functional Requirements

The following is a table describing our functional requirements

Req#	Requirement	Priority	Date Rvwd	Reviewed / Approved
QB_FER_01	The system shall display a chart of all object detection models ranked by a default metric of accuracy	1	02/10/2025	Kabire Akbari Ji Guo Jacob Autus Lucas Gomes
QB_FER_02	The system shall allow the user to re-sort the chart by another metric of the users choice	2	02/10/2025	Kabire Akbari Ji Guo Jacob Autus Lucas Gomes
QB_FER_03	The system shall allow user to filter chart by runtime or target device hardware	2	02/10/2025	Kabire Akbari Ji Guo Jacob Autus Lucas Gomes
QB_FER_04	The system shall allow the user to perform a one-on-one comparison by selecting two object detection models and displaying a side-by-side comparison of their performance metrics.	1	02/10/2025	Ji Guo Jacob Autus Lucas Gomes
QB_FER_05	The system shall enable users to click on any object detection model on the leaderboard to view detailed benchmark information, including latency, inference time, compute units, estimated peak memory usage, and mAP.	2	02/10/2025	Ji Guo Jacob Autus Lucas Gomes

Req#	Requirement	Priority	Date Rvwd	Reviewed / Approved
QB_DB_01	The system shall have a secure database that stores all the results.	1	02/10/2025	Kabire Akbari Ji Guo Jacob Autus Lucas Gomes
QB_DB_02	The system shall validate benchmark test results and store them in the database prior to making them accessible to users.	1	02/10/2025	Kabire Akbari Ji Guo Jacob Autus Lucas Gomes

2.3 Database Security and Integrity

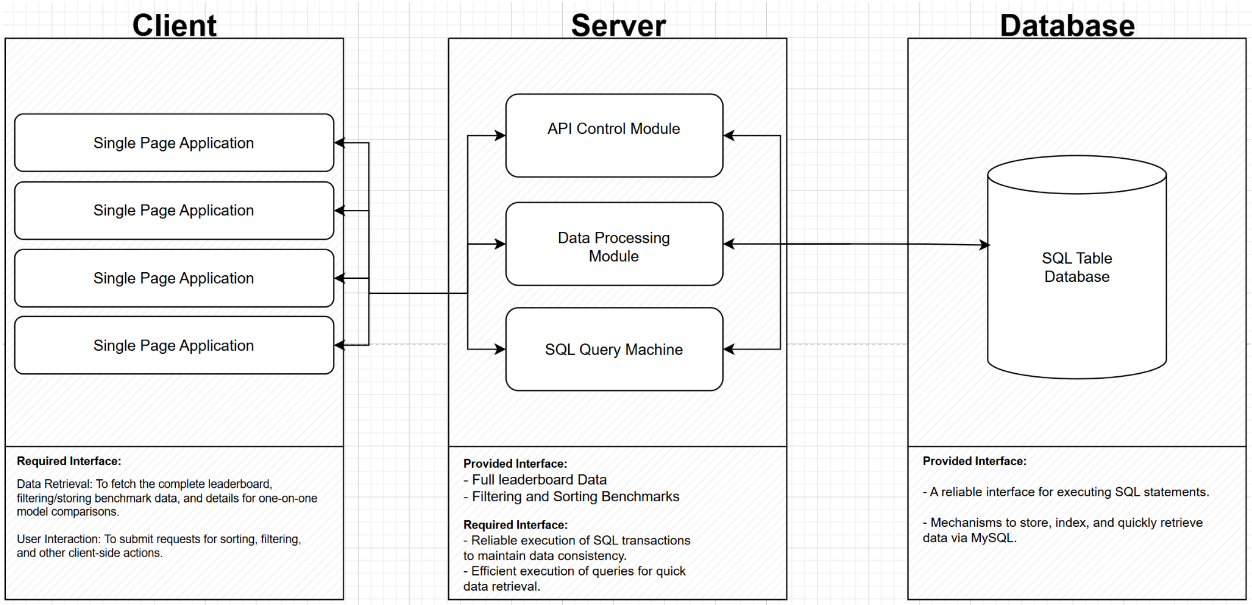
The system shall not allow any outside party to send a query to the database while also not allowing any outside party to modify, delete, add or otherwise alter the database.

2.4 Compatibility

The website shall be compatible with major web browsers (e.g., Chrome, Firefox, Safari, Edge) and support responsive design across various devices (desktop, tablet, mobile).

3. Architectural Design

Below, in figure 3, is a figure of our design, followed by a breakdown of each module and its respective required and provided interfaces. The whole architecture is divided into three modules: the User Client, Server, and Database.



Required Interface:

Data Retrieval: To fetch the complete leaderboard, filtering/storing benchmark data, and details for one-on-one model comparisons.

User Interaction: To submit requests for sorting, filtering, and other client-side actions.

Provided Interface:
- Full leaderboard Data
- Filtering and Sorting Benchmarks

Required Interface:
- Reliable execution of SQL transactions to maintain data consistency.
- Efficient execution of queries for quick data retrieval.

Provided Interface:

- A reliable interface for executing SQL statements.

- Mechanisms to store, index, and quickly retrieve data via MySQL.

Figure 3: Architectural Design Diagram

3.1 User Client Design

Purpose.

The **User Client** serves as the interface through which end users interact with the benchmark leaderboard. This module is designed with a focus on usability and interactivity, providing essential features to enhance the user's experience and facilitate data-driven insights.

One of the core functionalities of the **User Client** is **Displaying Charts**, specifically showcasing object detection models ranked by a chosen metric, such as mean Average Precision (mAP) or inference time. This visual representation allows users to quickly assess model performance and identify leading options in the dataset.

Additionally, the **User Client** offers **Sorting and Filtering Capabilities**, enabling users to refine the displayed data based on various performance metrics and device characteristics. This flexibility allows for tailored data views that match specific analysis needs, contributing to a more personalized and efficient user experience.

Another key feature is the **One-on-One Comparison Tool**, which allows users to select specific models and analyze their performance side by side. This detailed comparison feature aids in making informed decisions by highlighting the strengths and weaknesses of each model relative to the selected metrics.

High-level Design.

In this context, the proposed application architecture consists of three primary modules: the **UI Components Module**, the **Visualization Module**, the **Interaction Module**, and the **Data Formatting Module**. Each of these modules serves a distinct purpose and contributes to the overall functionality and usability of the application.

The **UI Components Module** is responsible for managing the presentation and layout aspects of the application. This module is divided into two main categories: **Presentation Components** and **Layout and Navigation**.

Presentation Components include a collection of reusable visual elements such as buttons, cards, modals, and other interface components. These components ensure consistency in design and improve development efficiency by allowing reuse across different parts of the application. Additionally, this module incorporates dashboard views, including leaderboards, interactive charts, and detailed comparison interfaces, providing users with a comprehensive and visually appealing display of data.

On the other hand, the **Layout and Navigation** components manage the routing and navigation within the single-page application (SPA). These components include menu controllers that facilitate seamless transitions between different views, such as the general overview and detailed model views. This structured approach to layout and navigation enhances the user experience by providing intuitive and accessible navigation throughout the application.

The **Visualization Module** is dedicated to presenting data in a meaningful and interactive manner. It primarily focuses on **Charting Components**, which are crucial for rendering various types of charts, including leaderboards and one-on-one comparison charts. By offering interactive and dynamic visualizations, this module enables users to gain insights from data effectively. The ability to visualize data through well-designed charts not only improves data comprehension but also adds a layer of engagement to the user interface.

The **Interaction Module** plays a vital role in bridging the gap between the user interface and the backend services. It consists of three key elements: **Event Handling**, **API Communication**, and **Data Formatting Modules**.

The **Event Handling** component is designed to capture and respond to user interactions, such as clicks, filter selections, and sorting actions. By efficiently managing these events, the application can deliver a responsive and dynamic user experience.

The **API Communication** component is responsible for managing AJAX/HTTP requests to the server’s API. This includes not only sending and receiving data but also handling errors and implementing retry mechanisms to maintain application stability in the event of network issues.

Finally, the **Data Formatting Modules** provide utility functions for converting raw data into a format suitable for visualization and display. These functions ensure that the data presented to users is both accurate and easy to understand, contributing to the overall clarity of the application’s output.

Required Interface.

The **Required Interface** outlines the critical interactions between the application and external data sources, focusing on **Data Retrieval** and **User Interaction** components.

Data Retrieval involves establishing reliable methods to fetch various types of data required by the application. This includes retrieving the complete leaderboard data, accessing filtered and sorted benchmark information, and obtaining detailed data for one-on-one model comparisons. By ensuring efficient and accurate data retrieval, the application can provide up-to-date and relevant information to users, enhancing the decision-making process and the overall user experience.

User Interaction components facilitate the submission of user-generated requests. These requests may include actions such as re-sorting data, applying filters, or executing additional client-side interactions. By offering a responsive interface that captures and processes these inputs, the application can adapt to user needs dynamically, maintaining a high level of interactivity and functionality.

3.2 Server Design

Purpose

The **Server** functions as the core backend component of the Qual-Bench AI system, serving multiple critical roles to ensure seamless operation and data flow. As the **Central Hub for API Requests**, the server manages all incoming requests from **User Clients**, acting as the primary communication bridge between the frontend interface and backend resources.

In addition to handling API requests, the **Server** is responsible for **Interfacing Securely with the Database Server**. This involves managing secure connections to the database where benchmark results are stored, ensuring data integrity and confidentiality.

High-Level Design

The **Server's High-Level Design** comprises two primary modules: the **API Controller Module** and the **SQL [3] Query Module**.

The **API Controller Module** is at the forefront of request management. It includes:

- **Routing**
- **Request Parsing**
- **Data Processing**

Routing maps incoming HTTP requests to their respective API endpoints, managing URL routing for data retrieval. It ensures that requests are directed to the appropriate service efficiently. Then for **request parsing**, once a request is received, the server interprets and validates the incoming data. It then forwards validated requests to the business logic layer, ensuring only legitimate and properly formatted data is processed. Finally for **data processing** the core functionality of the server resides in this component, which handles tasks such as data **Filtering**, **Sorting**, and **Metric Calculations**. This processing is crucial for delivering accurate and relevant data to the **User Client**.

The **SQL Query Module** is the gateway to the **Database Server**. It is responsible for executing SQL queries to retrieve and manipulate data. This module ensures that data displayed to users is current and consistent, providing quick and reliable access to the benchmark results stored in the database.

Required Interface

The **Required Interface**, as detailed in section 3.3 of the system documentation, needs to ensure the **Reliable Execution of SQL Transactions**, maintaining data consistency across all operations. Additionally, this interface must support the **Efficient Execution of Queries**, enabling quick retrieval of data and enhancing the system's overall performance.

Provided Interface

The **Provided Interface** focuses on **Fetching Full Leaderboard Data**, **Filtering and Sorting Benchmarks**, and **Retrieving Detailed Comparison Data**. Through these capabilities, the **Server** plays a pivotal role in delivering dynamic and responsive data services to the **User Client**.

3.3 DataBase Server Design

Purpose

The **Database Server** plays a critical role in the Qual-Bench AI system by securely storing and managing all benchmark data and associated system information. The **Database Server** holds essential data points, including latency, inference time, compute units, memory usage, and mean Average Precision (mAP) for. This centralized storage ensures data consistency and reliability, providing a solid foundation for analytics and reporting within the application.

High-Level Design

The **Database Server** is organized into several specialized modules, each contributing to efficient data management and retrieval. The primary components include the **Benchmark Results Table**, **Model Information Table**, and **Index Management**.

The Benchmark Results Table is designed to store comprehensive test outcomes for object detection models. Key metrics recorded include **Memory Usage**, **mAP**, **Inference Time**, and other performance-related data. The structured storage of benchmark results facilitates quick access and detailed analysis through the application’s front-end.

In addition to benchmark metrics, the **Database Server** maintains a **Model Information Table**, which houses metadata about each object detection model. This includes details such as the **Model Name**, **Version**, **Description**, and **Runtime**. By maintaining detailed model information, the server supports advanced filtering and model-specific queries from the **User Client**.

To enhance performance, the **Database Server** employs **Index Management** techniques. Indexes are created on frequently queried columns, including **Model IDs**, **Runtime**, and **Device Types**. This optimization reduces query execution time and improves the responsiveness of the system when handling complex or large-scale data retrieval operations.

Provided Interface

The **Database Server** focuses on delivering a **Reliable Interface for SQL Operations**. This interface supports essential database actions such as **Creating**, **Reading**, **Updating**, and **Deleting (CRUD)** data within the repository. The **Database Server** also incorporates robust **Indexing and Data Retrieval Mechanisms**, ensuring that data is stored efficiently and can be accessed rapidly in response to **Server** requests.

4. Implementation Plan

4.1 Implementation Platform and Frameworks

The frontend will be developed using React.js [4], a robust JavaScript framework known for its efficiency in building dynamic and responsive user interfaces. To enhance data representation, the implementation will integrate Chart.js [5] or D3.js [6], enabling intuitive and visually appealing data visualizations for performance metrics.

Moving onto the backend infrastructure, the system will utilize Flask [7], a lightweight, flexible web framework written in Python that helps build web applications and APIs. Flask provides a scalable and modular approach to backend development, making it an ideal choice for managing API integration and application logic. We will be using a relational database in MySQL [8] to store critical data, including model details, datasets, and processed results. MySQL's structured data storage and query capabilities will ensure efficient data retrieval and management.

During the initial development phase, the application will be hosted locally for testing and demonstration purposes. For full-scale deployment, the system will be migrated to Google Cloud [10], leveraging its reliability, scalability, and security features. To facilitate seamless collaboration and maintain a structured development workflow, GitHub [11] will be used for version control. This ensures efficient tracking of changes, bug fixes, and feature updates while enabling collaborative development among team members.

4.2 Implementation Tasks

Req#	**Tasks**
TSK_DT_01	Run two models and record the correct result, determine output format and ensure standardization for results gathering.
TSK_DB_01	Design and implement the MySQL database schema.
TSK_DB_02	Store results from TK_DT_01 into table. And test the queries.
TSK_DB_03	Implement security measures.
TSK_DB_04	Record result for all models into the database.
TSK_FR_01	Implement core UI components in React.js, including leaderboard displays.
TSK_FR_02	Implement interactive charts, integrate Chart.js/D3.js for data visualization.
TSK_BK_01	Host local server to test with database.
TSK_BK_02	Integrate frontend and backend components with each other.
TSK_TST_01	Conduct unit tests and integration tests for API endpoints and UI functionality.
TSK_TST_02	Conduct security tests.
TSK_TST_03	Conduct tests on interactive charts.
TSK_FN_DP_01	Deploy the system on Google Cloud, including both the Flask server and MySQL database.

Req#	Tasks
TSK_FN_UI_01	Finalize and polish the UI, ensuring responsiveness and accessibility.

4.3 Implementation Schedule

The following is a table describing the schedule we plan to follow for implementation.

Category	Development Activities	Outcomes	Timeline	Personnel
Data Calculation	TSK_DT_01: Run two models and record the correct result, determine output format and ensure standardization for results gathering.	Standardized results format established for benchmarking.	2/19/2025 – 2/26/2025	Kabire Akbari
Database	TSK_DB_01: Design and implement the MySQL database schema.	Database schema designed and ready for storing benchmark results.	2/26/2025 – 3/03/2025	Ji Guo Lucas Gomes Kabire Akbari Jacob Autus
Front-End	TSK_FR_01: Implement core UI components in React.js, including leaderboard displays.	Functional leaderboard displaying AI model rankings.	2/12/2025 – 3/05/2025	Ji Guo Jacob Autus
Database	TSK_DB_02: Store results from TSK_DT_01 into table. And test the queries.	Results stored successfully; queries validated for correctness.	3/03/2025 – 3/05/2025	Lucas Gomes Jacob Autus Kabire Akbari
Back-End	TSK_BK_01: Host local server to test with the database.	Local development environment set up for testing.	3/05/2025 – 3/17/2025	Ji Guo
Back-End	TSK_BK_02: Integrate frontend and backend components with each other.	Fully functional system with UI connected to API and database.	3/17/2025 – 3/18/2025	Ji Guo
Testing	TSK_TST_01: Conduct unit tests and integration tests for API endpoints and UI functionality.	API and UI tested; ensured stability and correctness.	3/18/2025 – 3/19/2025	Ji Guo Jacob Autus
Prototype	Prototype ready	Prototype good for demonstrate	3/19/2025	Kabire Akbari Ji Guo Jacob Autus Lucas Gomes
Database	TSK_DB_03: Implement security measures.	Database secured with parameterized queries and access controls.	3/19/2025- 4/09/2025	Ji Guo
Front-End	TSK_FR_02: Implement interactive charts, integrate Chart.js/D3.js for data visualization.	Charts integrated for visualizing benchmarking results.	3/19/2025- 4/09/2025	Ji Guo

Category	Development Activities	Outcomes	Timeline	Personnel
Testing	TSK_TST_02: Conduct security tests.	Verified system security and mitigated vulnerabilities.	4/09/2025- 4/14/2025	Kabire Akbari
Testing	TSK_TST_03: Conduct tests on interactive charts.	Interactive UI tested; ensured stability	4/09/2025- 4/14/2025	Ji Guo Jacob Autus
Front-End	TSK_FN_UI_01: Finalize and polish the UI, ensuring responsiveness and accessibility.	Optimized UI with full responsiveness and accessibility compliance.	4/14/2025-5 5/05/2025	Ji Guo
Database	TSK_DB_04: Record results for all models into the database.	Complete dataset populated in the database for benchmarking.	4/21/2025- 4/23/2025	Kabire Akbari
Collect Advice	Collect advices from the milestone presentation	Using advice on TSK_FN_UI_01 and other improvements.	4/23/2025	Kabire Akbari Ji Guo Jacob Autus Lucas Gomes
Deployment	TSK_FN_DP_01: Deploy the system on Google Cloud, including both the Flask server and MySQL database.	Fully deployed system accessible on the cloud.	4/23/2025- 5/05/2025	Ji Guo
Final	Final Project Showcase	Final stage of product is ready	5/05/2025- 5/07/2025	Kabire Akbari Ji Guo Jacob Autus Lucas Gomes

References

1. “Qualcomm Ai Hub.” *Qualcomm AI Hub*, aihub.qualcomm.com/models?domain=Computer%2BVision&useCase=Object%2BDetection. Accessed 24 Feb. 2025.
2. “COCO.” *COCO - Common Objects in Context*, https://cocodataset.org/#download. Accessed 13 Feb. 2025.
3. “Sqlalchemy.” *SQLAlchemy*, www.sqlalchemy.org/. Accessed 24 Feb. 2025.
4. “React.js” *React.js*, https://react.dev/. Accessed 24 Feb. 2025
5. “Chart.js” *Chart.js*, https://www.chartjs.org/. Accessed 24 Feb 2025
6. “D3.js” *D3.js*, https://d3js.org/. Accessed 24 Feb 2025
7. “Flask.js” *Flask.js*, https://flask.palletsprojects.com/en/stable/. Accessed 24 Feb 2025
8. “MySQL” *MySQL*, https://www.mysql.com/. Accessed 24 Feb 2025
9. “W3schools.Com.” *W3Schools Online Web Tutorials*, www.w3schools.com/ai/ai_chartjs.asp. Accessed 24 Feb. 2025.
10. “Google Cloud” *Google Cloud*, https://cloud.google.com/. Accessed 24 Feb 2025
11. “Github” *Github*, https://github.com/. Accessed 24 Feb 2025