Project Proposal

Project Title:

Data Analysis and Prediction Models for AI Benchmarking Data

Submitted By:

Team 6

Parth Praful Parakhiya | 110179090

Tirth Rameshbhai Patel | 110157043

Sanket Harshadbhai Kothiya | 110156177

Internal Supervisor:

Dr. Olena Syrotkina

School of Computer Science, University of Windsor

Industry Supervisor:

Nishatha Nagarajan, Jaguar Land Rover (JLR)

University of Windsor

Summer 2025

Contents

1 Abstract

The rapid evolution of AI accelerators such as GPUs, TPUs, and custom ASICs has created a critical demand for standardized and interpretable benchmarking tools. While existing suites like MLPerf provide structured performance data, comparing across diverse hardware remains complex due to architectural variability. This project aims to bridge that gap by developing a centralized AI benchmarking database and predictive modeling tool capable of estimating key performance indicators (KPIs) such as throughput, latency, and energy efficiency. The approach involves curating publicly available benchmark datasets, engineering architecture-aware features, and training regression models with bias normalization to ensure cross-platform interpretability. The final deliverable will be a validated prototype that supports hardware-model compatibility analysis, streamlines benchmarking workflows, and guides efficient deployment planning for AI systems.

Keywords: AI Benchmarking, KPI Prediction, Hardware Architecture, Performance Modeling, MLPerf, SHAP, Regression

2 Introduction

The performance of AI models is highly sensitive to the characteristics of the hardware on which they are executed. Factors such as memory bandwidth, compute density, and architectural support for operations like matrix multiplication directly impact key metrics such as latency, throughput, and energy consumption. As AI continues to be deployed across increasingly diverse environments—from data centers to edge devices—hardware selection has become a critical design choice.

Despite the availability of benchmarking suites like MLPerf, which provide standardized evaluation protocols for image classification, language modeling, and more, the decision-making process remains opaque. These benchmarks typically focus on narrow workloads and lack mechanisms to account for hardware-specific architectural advantages or limitations. Additionally, they do not provide predictive insights for new or unseen hardware-model pairings.

This project proposes a modular framework that integrates benchmarking data from trusted sources, normalizes it across architectural dimensions, and enables predictive modeling using regression techniques. The resulting tool will support developers, researchers, and system architects in making informed, data-driven decisions about hardware selection—reducing cost, accelerating deployment, and improving reproducibility in AI system design.

3 Problem Statement

Evaluating the performance of AI models across diverse hardware platforms remains a fragmented and unreliable process. Current approaches rely heavily on:

- Parsing disparate benchmarking reports with inconsistent formats and incomplete metadata.
- Trusting vendor-reported or synthetic performance metrics that lack transparency or reproducibility.
- Repeating computationally expensive benchmark runs on physical hardware for every new configuration.

These challenges are driven by three fundamental gaps:

1. **Fragmented Data**: Benchmarking results are scattered across academic papers, vendor whitepapers, and platforms like MLPerf, with no unified structure for comparison.
2. **Architectural Variability**: Hardware architectures differ in compute layout, memory access patterns, and software stack optimizations—leading to biased or incomparable performance results.
3. **Lack of Predictive Insight**: There is no lightweight, architecture-aware tool that can predict AI performance across unseen hardware without requiring actual execution.

Without a standardized and interpretable mechanism to bridge these gaps, stakeholders face longer development cycles, increased infrastructure costs, and poor visibility into optimal hardware-model pairings.

4 Background Study

While several benchmarking initiatives exist, most focus on measuring raw performance without offering predictive insights or architectural normalization. Notable efforts include:

- **MLPerf** provides standardized benchmarks across AI tasks such as image classification and NLP. However, results are segmented by task and hardware class, with limited comparability and no interpretability mechanisms [1].
- **Vendor Benchmarks** (e.g., NVIDIA A100 datasheets) highlight best-case performance but often omit architectural constraints like underutilized accelerators or thermal throttling [2].
- **Google's TPUv4 Studies** introduced architecture-aware modeling for TPUs but lack generalization across third-party hardware or diverse workloads [6].
- **NAS-Bench-201** and other Neural Architecture Search tools evaluate model efficiency under resource constraints but require exhaustive sampling, making them computationally intensive for hardware comparisons [5].

In contrast, our project uniquely integrates:

- Database engineering for structured benchmark consolidation
- Regression-based modeling to enable prediction across architectures
- Bias-correction features to normalize architectural variability
- A lightweight interface for usability in both research and industry settings

This combination provides early estimations of performance trends using minimal inputs while preserving interpretability—offering a novel solution to a long-standing benchmarking bottleneck.

5 Goals and Objectives

The primary goal of this project is to design a lightweight, reliable, and interpretable system that can estimate the performance of AI models on diverse hardware platforms without requiring full-scale benchmarking. This will support developers, researchers, and organizations in making informed hardware-model deployment decisions efficiently and accurately.

To achieve this goal, the following objectives are defined:

- **Centralize AI Benchmark Data:** Aggregate benchmark results from sources like MLPerf, academic publications, and vendor datasheets into a structured and searchable database.
- **Engineer Meaningful Features:** Extract and encode critical attributes of AI models and hardware architectures (e.g., parameter count, compute type, FLOPs, memory bandwidth) to serve as model inputs.
- **Build Predictive Models:** Train interpretable regression models (e.g., linear, random forest, boosting) to forecast key performance indicators (KPIs) like throughput, latency, and efficiency.
- **Incorporate Architecture-Aware Bias Correction:** Normalize prediction outputs to account for vendor-specific accelerators and structural variations across platforms.
- **Validate with Real-World Benchmarks:** Evaluate model accuracy using unseen benchmark data to ensure predictive reliability and generalization.
- **Develop a Functional Interface:** Create a simple yet effective CLI or Streamlit-based tool to accept user inputs and provide performance predictions with comparisons.

6 Methodology

The project is structured into seven systematic phases, each building toward a robust, predictive benchmarking system.

Phase 1: Data Aggregation — Collect AI benchmarking data from MLPerf, vendor whitepapers, academic publications, and hardware specification sheets. Store the cleaned and standardized data in CSV or relational formats for downstream use.

Phase 2: Schema and KPI Normalization — Design a structured schema to capture models, tasks, datasets, and hardware attributes. Normalize raw metrics into unified KPIs such as latency (ms), throughput (samples/sec), and energy efficiency (inferences/Watt).

Phase 3: Feature Engineering — Extract both low-level and derived features. These include FLOPs, parameter count, memory bandwidth, and precision type. Create composite features like FLOPs/core and latency per GB/s. Ensure consistency in target KPI scaling.

Phase 4: KPI Prediction Modeling — Train baseline and non-linear models (Linear Regression, Random Forest, Gradient Boosting) to predict KPIs. Evaluate performance using Mean Absolute Error (MAE), R^2, and cross-validation.

Phase 5: Architecture Bias Handling — Introduce bias flags for architectural distinctions (e.g., tensor cores, matrix units). Adjust model predictions using correction weights. Apply SHAP or permutation feature importance to ensure explainability.

Phase 6: Prototype Tool Development — Build a lightweight UI using Streamlit or CLI to accept model/hardware configurations and return KPI predictions. Include comparison tables and download functionality for reports.

Phase 7: Validation and Reporting — Test model accuracy on unseen hardware-model combinations. Document the pipeline, tool interface, error margins, and results. Prepare a GitHub repository and final presentation materials.

System Architecture Diagram

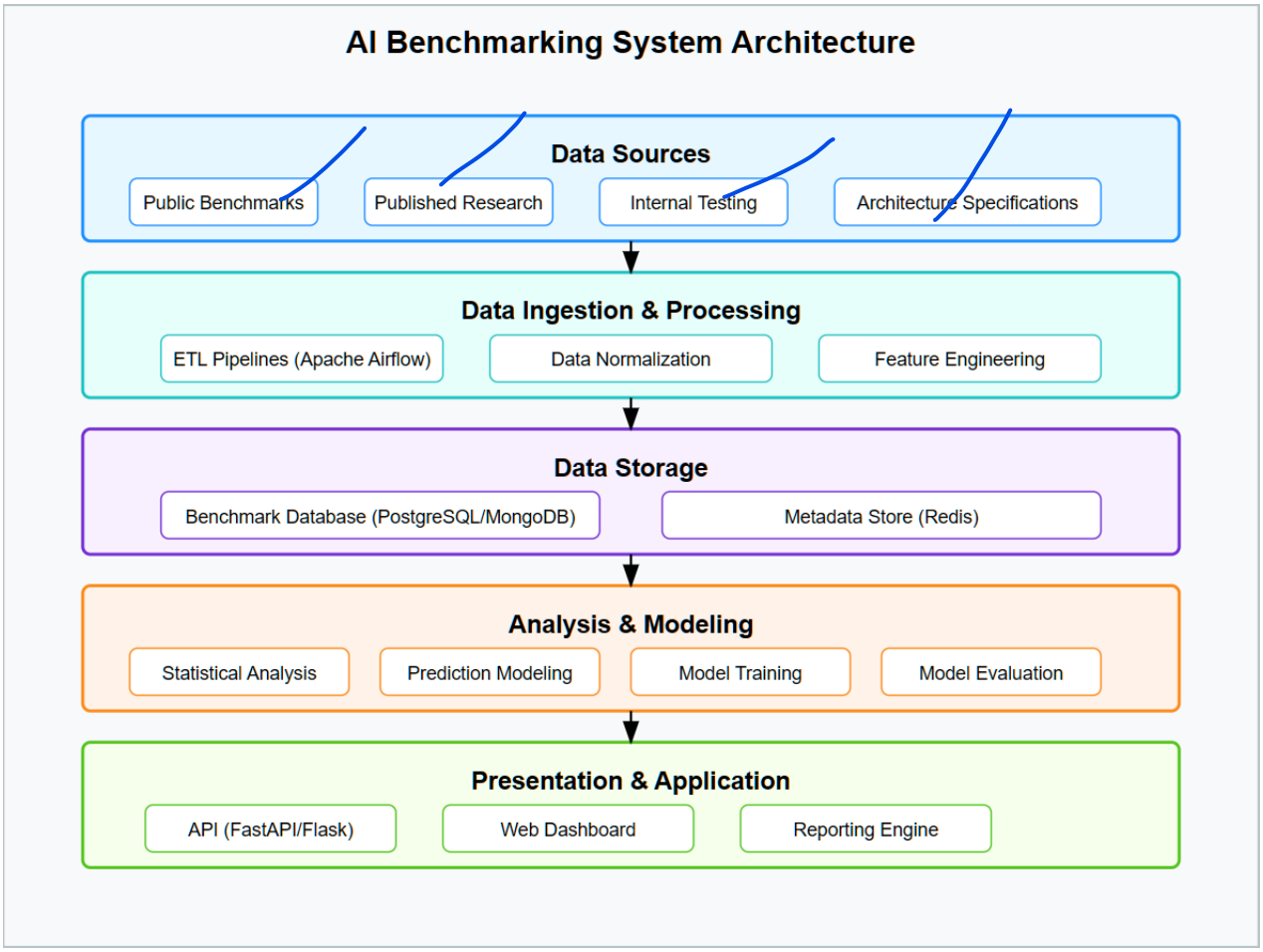

Figure 1: This diagram outlines the modular data pipeline supporting benchmark ingestion, modeling, and result presentation

7 Timeline Overview

The project is scheduled over a 10-week period from June 1 to August 1, with each week targeting specific deliverables aligned to core development phases.

Week	Milestone Description

Week 1	**Benchmark Source Identification** — Finalize trusted sources such as MLPerf, vendor documentation, and academic repositories. Establish task ownership and initialize Git repository.
Week 2	**Data Aggregation and Schema Design** — Scrape and collect initial benchmark data. Define relational schema. Normalize data units (latency, throughput, power) and store structured records in CSV or SQL.
Week 3	**Feature Extraction and Categorization** — Extract hardware and model attributes (e.g., FLOPs, core type, parameter count). Classify models (e.g., CNNs, Transformers). Clean incomplete records.
Week 4	**Exploratory Data Analysis (EDA)** — Visualize feature distributions, detect outliers, and uncover performance trends by hardware type. Finalize features for modeling.
Week 5	**Baseline Model Training** — Train linear regression and random forest models to estimate KPIs. Evaluate with MAE and R^2 scores. Document baseline results.
Week 6	**Bias Correction and Feature Refinement** — Integrate architectural bias flags. Improve prediction accuracy with tuning. Interpret features using SHAP or importance metrics.
Week 7	**Tool Interface Development** — Build and test the prototype CLI or Streamlit interface. Implement prediction display, configuration input, and export functionality.
Week 8	**Validation Against Known Benchmarks** — Run tool on known model-hardware combinations. Compare predictions with actual values and calculate error margins.
Week 9	**Documentation and Final Polish** — Complete technical report covering methodology, evaluation, and system architecture. Finalize code structure and internal usage guide.
Week 10	**Final Testing and Presentation Prep** — Conduct supervisor review. Validate tool reproducibility. Submit all deliverables and prepare for final presentation on August 1.

8 Tools and Technology Stack

The toolchain selected for this project emphasizes reproducibility, accessibility, and industry relevance. Each tool serves a specific role in the data processing, modeling, deployment, or documentation pipeline.

Category	Tools and Justification

Data Collection	**Python** — Primary language for scripting and data handling. **Requests, BeautifulSoup** — For web scraping and structured extraction from sources like MLPerf leaderboards and vendor sites.
Data Storage & Processing	**Pandas** — For data wrangling, transformation, and filtering of benchmark and hardware data. **PostgreSQL / SQLite** — Optional relational databases for persistent storage of normalized schema records.
Modeling & Prediction	**scikit-learn** — Core library for regression modeling (Linear, Random Forest, Gradient Boosting) and evaluation metrics. **NumPy** — Efficient numerical operations and matrix manipulation for feature engineering.
Explainability & Analysis	**SHAP (SHapley Additive Explanations)** — For interpreting feature importance and understanding prediction drivers. **Matplotlib, Seaborn** — For visualizing trends, correlations, model accuracy, and bias effects.
Tool Interface	**Streamlit** — Rapid deployment of a user-friendly web interface for model input and result visualization. **Command Line Interface (CLI)** — Lightweight terminal tool for users preferring script-based usage.
Documentation & Reporting	**Jupyter Notebooks** — Interactive development and presentation of data exploration and model outputs. **Overleaf (LaTeX)** — For preparing structured technical documentation and final reports. **Markdown + GitHub Wiki** — For hosting usage guides, model documentation, and update logs.
Version Control	**Git + GitHub** — For team collaboration, codebase tracking, and public/private repository management. Project assets include: • Source code for data processing, model training, and CLI/Streamlit interface • Cleaned datasets and metadata • Instructions and usage guides

9 Expected Deliverables

The project will produce the following key deliverables, each contributing to the overarching goal of enabling interpretable performance prediction for AI workloads:

1. **Centralized AI Benchmark Database** A structured dataset combining results from MLPerf, vendor datasheets, and academic benchmarks. Includes model specs, hardware attributes, and normalized KPIs (e.g., latency, throughput, efficiency).

2. **Feature-Engineered Modeling Dataset** A processed dataset ready for regression modeling, with derived features such as FLOPs/core, latency per GB/s, and architectural flags.

3. **Trained Regression Models** Models capable of estimating key KPIs across hardware platforms. Includes linear, random forest, and boosting variants with validation metrics (MAE, R^2).

4. **Prediction Interface (Prototype)** A functional CLI or Streamlit app that accepts hardware-model configurations, predicts KPIs, and displays comparative performance.

5. **Technical Report and Documentation** Comprehensive LaTeX document detailing methodology, model development, evaluation results, interface design, and assumptions.

6. **Presentation and Demo Assets** Slide deck and live demo walkthrough for final review. Visualizations of prediction accuracy, data pipeline, and feature importances.

7. **GitHub Repository** Public/private version-controlled repository containing:

   - All source code and configuration files
   - Final datasets and schema documentation
   - README, usage guide, and evaluation notebooks

10 Conclusion

This proposal presents a structured and technically feasible plan to develop a data-driven system for AI benchmarking and performance prediction. By consolidating fragmented benchmarking datasets, applying architecture-aware normalization, and building interpretable regression models, the project addresses a growing industry challenge: enabling fair, efficient, and scalable evaluation of AI workloads across diverse hardware platforms.

The deliverables—including a centralized benchmark database, prediction models, and a user-facing tool—aim to empower researchers, developers, and hardware vendors to make informed deployment decisions without extensive manual benchmarking. The inclusion of architectural bias correction and explainability ensures the predictions are not only accurate but also trustworthy.

The project is designed with a minimal learning curve and clear milestones, making it realistic within the 10-week timeframe. Future extensions could explore:

- Integration with live benchmark APIs for real-time updates
- Support for hyperparameter-level comparisons (e.g., batch size, quantization)

- Expansion to a cloud-hosted dashboard for continuous performance tracking

By bridging the gap between raw performance data and intelligent infrastructure decisions, this work contributes to a more transparent and efficient AI hardware ecosystem.

References

[1] V. J. Reddi *et al.*, "MLPerf Inference Benchmark," *IEEE Micro*, vol. 41, no. 2, pp. 45–52, Mar.-Apr. 2021. [Online]. Available: https://mlcommons.org/en/inference-datacenter-40/

[2] NVIDIA Corporation, "NVIDIA A100 Tensor Core GPU Architecture," NVIDIA, Tech. Rep., 2020. [Online]. Available: https://resources.nvidia.com/en-us/data-center/a100-datasheet

[3] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011. [Online]. Available: https://scikit-learn.org/stable/

[4] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," in *Proc. NeurIPS*, 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html

[5] X. Dong and Y. Yang, "NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search," in *Proc. ICLR*, 2020. [Online]. Available: https://arxiv.org/abs/2001.00326

[6] N. P. Jouppi *et al.*, "Ten Lessons From Three Generations Shaped Google's TPUv4i Architecture," *arXiv preprint*, arXiv:2204.05839, 2022. [Online]. Available: https://arxiv.org/abs/2204.05839

[7] V. Sze *et al.*, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017. [Online]. Available: https://ieeexplore.ieee.org/document/9174923

[8] Alibaba DAMO Academy, "Efficient Benchmarking and Prediction of AI Models," *arXiv preprint*, arXiv:2301.12937, Jan. 2023. [Online]. Available: https://arxiv.org/abs/2301.12937