INDUSTRY-11 – Prompt Optimization
Final Report
AI 7993 – Section W01 – Spring 2026
April 29, 2026

Faith Ogbefho
Researcher/Presenter

Destiny Raburnel
Documentation/Developer

Jamia Jackson
Documentation/Developer

Team Members:

Name	Role	Cell Phone/Alternate Email
Faith Ogbefho	Researcher/Presenter	4784429497 Faithogbefho@gmail.com
Destiny Raburnel	Documentation/Developer	678.622.9815 Destiny.Raburnel@gmail.com
Jamia Jackson	Documentation/ Developer	478.258.9794 Jamia.Jackson30@gmail.com
Arthur Choi	Advisor	770.867.5309 achoi13@kennesaw.edu
Sai Bharath	Mentor	bharathsrv1@gmail.com

Website: https://destinyj621.github.io/prompt_optimization/
GitHub: https://github.com/destinyj621/prompt_optimization

1. INTRODUCTION

1.1 Overview

This report documents the design, development, and evaluation of the Prompt Optimization Workbench for AI Benchmarking. The purpose of this system is to evaluate and compare different prompt strategies across tasks and models by collecting performance metrics such as token usage, latency, cost estimation, and output quality. Results are stored and visualized to support side-by-side comparisons, so users can see which prompt strategies work best for their needs.

1.2 Project Goals

The goals of the Prompt Optimization Benchmarking Platform are:

1. Design a repeatable framework for evaluating prompt optimization strategies.
2. Quantify trade-offs between cost, latency, sustainability, and output quality.
3. Support multiple benchmark task types to evaluate prompt strategies across a range of real-world AI applications.
4. Deliver a demo-ready UI that clearly communicates results to non-ML stakeholders.
5. Collect and store structured experiment results to support reproducible and statistically reliable benchmarking.
6. Provide transparent metric reporting that allows users to interpret and compare results with full context.

1.3 Background and Motivation

As organizations increasingly adopt large language models across industries, challenges related to cost, latency, output quality, and sustainability become more significant. Prompt design is one of the simplest ways to improve performance, but it rarely receives the same attention as model selection or infrastructure decisions.

The way a prompt is constructed directly influences how a model interprets a task, how many tokens it consumes, how long it takes to respond, and the quality of the output it produces. Small changes in prompt structure, the inclusion or exclusion of examples, and the level of instruction detail can produce measurably different results across the same model and task. Despite this, prompt design decisions are often made informally and without systematic evaluation. This workbench addresses that gap by providing a structured environment for testing, comparing, and analyzing prompt strategies in a reproducible and objective way.

1.4 Assumptions

It is assumed that users have basic familiarity with generative AI concepts, including prompts and model execution. Users are expected to provide appropriate datasets or text inputs. The system is assumed to be used in an academic or controlled evaluation context where interpreted results are used to support decisions rather than guarantees.

2. REQUIREMENTS

2.1 Design Constraints

2.1.1 Environment The system operates within a controlled environment using a local model. The system functions within the computational and network limitations of an academic setting.

2.1.2 User Characteristics The system is designed for users with familiarity with generative AI concepts and prompt engineering. The user interface prioritizes clarity, guided workflows, and interpretability to support non-technical stakeholders in understanding results.

2.1.3 System Constraints The system is limited to prompt evaluation and observability and does not support model training or fine-tuning. Sustainability metrics are presented as approximations rather than exact measurements. The system focuses on evaluation, comparison, and reporting rather than deployment or real-time use.

2.2 Functional Requirements

2.2.1 Prompt Input

- The system shall allow users to create, edit, and store multiple prompt variants per task.
- The system shall support structured prompt inputs, including system prompts, user instruction prompts, and optional examples or data.
- The system shall associate metadata with each prompt variant, including strategy type and description.

2.2.2 Experiment Configuration

- The system shall allow users to define experiments by selecting task and strategy type.
- The system shall support local execution.
- The system shall allow users to configure the number of runs per prompt variant.

2.2.3 Results Comparison and Visualization

- The system shall compute and display metrics across multiple runs.
- The system shall show side-by-side comparisons of prompt strategies using tables and charts.

2.2.4 Reporting and Exporting

- The system shall allow users to export results in CSV format.

2.3 Non-Functional Requirements

2.3.1 Security

- The system shall restrict access to stored prompts, datasets, and results to authorized users.
- The system shall avoid permanent storage of sensitive or harmful prompt data unless explicitly configured.
- The system shall support anonymization or redaction of sensitive content where applicable.

2.3.2 Capacity

- The system shall store and display results for at least fifty experiments without functionality degradation.
- The system shall maintain acceptable response times for experiment execution and result visualization.

2.3.3 Usability

- The system shall provide an intuitive user interface suitable for non-ML stakeholders.
- The system shall clearly label metrics, assumptions, and limitations to avoid misinterpretation.

2.4 External Interface Requirements

2.4.1 User Interface Requirements The system shall be a web-based user interface accessible through a modern browser. The interface shall allow users to define experiments through the selection of a task type, entering a prompt, and choosing a prompt strategy. The interface will present results on a separate page displaying metrics, side-by-side comparisons, and recent runs. The UI prioritizes clarity, consistency, and ease of interpretation to support non-technical stakeholders.

2.4.2 Hardware Interface Requirements The system shall operate on standard computing hardware without requiring specialized accelerators.

2.4.3 Software Interface Requirements The system shall interface with local or hosted large language model runtimes through APIs or command-line interfaces. The system shall integrate with data storage mechanisms for reading input and results. The system shall support exporting results in standard formats such as CSV.

2.4.4 Communication Interface Requirements The system shall use standard HTTP protocols for any external communication. All internal communication shall occur within the host operating system.

3. ANALYSIS

3.1 Problem Statement

With growing enterprise adoption of generative AI, teams face a fundamental challenge: how should prompt design decisions be made systematically? Most organizations experiment informally, with no structured mechanism for measuring and comparing prompt performance across tasks, models, or execution environments. This leads to suboptimal prompt choices, wasted token usage, inconsistent output quality, and missed opportunities for cost and latency optimization.

3.2 Tasks Selected for Benchmarking

Three task types were selected for their real-world relevance and their ability to produce objectively measurable outputs. Each task remains constant during experimentation while only the prompt strategy changes, ensuring that any measured differences in performance can be attributed to prompt design rather than task variation.

Sentiment Classification assigns a sentiment label to a given input text, with the possible labels being Positive, Negative, or Neutral. This task is widely applicable to real-world use cases such as customer feedback analysis, social media monitoring, and public opinion tracking. It was selected as a benchmark task because it has a well-defined input format, a strict output requirement of exactly one label, and a straightforward automated scoring method. Performance is evaluated using exact label match accuracy, calculated as the number of correct predictions divided by the total number of predictions. For this task, users must provide ground truth labels as part of the experiment configuration to enable accuracy evaluation.

Text Summarization condenses a longer passage into a shorter, coherent summary that retains the most important information. This task is relevant to use cases such as document review, research paper digestion, and report generation. It was selected because it produces open-ended outputs that require more nuanced evaluation than classification, making it a good test of how prompt strategy affects output quality beyond simple label matching. Output quality is evaluated using ROUGE scores, specifically ROUGE-1, ROUGE-2, and ROUGE-L, which measure the overlap between the model-generated summary and a reference summary at the unigram, bigram, and longest common subsequence levels respectively. An overall quality score is also computed using an LLM-as-judge evaluation approach, in which a separate model assessment rates the output based on correctness and completeness.

Question Answering generates a response to a question based on a provided passage or dataset. This task is relevant to conversational AI systems, search applications, and automated support tools. It was selected because it requires the model to ground its response in provided context rather than rely solely on pretrained knowledge, which introduces a different kind of evaluation challenge compared to classification and summarization. Output is evaluated using Exact Match, which checks whether the predicted answer precisely matches the reference answer, F1 Score, which measures token-level overlap between the predicted and reference answers, and an LLM-as-judge quality score that assesses correctness, completeness, and grounding relative to the provided context.

3.3 Prompt Strategies Evaluated

Two prompt strategies were implemented and evaluated across all three task types. These strategies were chosen because they represent the two most fundamental approaches to prompt construction and provide a meaningful baseline comparison.

Zero-Shot Prompting provides the model with only the task instructions and input text, with no prior examples included in the prompt. This strategy establishes a performance baseline by relying entirely on the model's pretrained knowledge to interpret the task and produce an appropriate output. Zero-shot prompting is the simplest and most token-efficient strategy, making it a useful reference point for evaluating whether additional prompt complexity actually improves results.

Few-Shot Prompting includes sample input-output pairs alongside the task instruction before the actual input is provided. These examples are appended to the prompt so the model receives everything in a single request. By seeing examples of the expected input format and output structure, the model can learn the reasoning pattern required for the task from within the prompt itself rather than relying solely on pretrained behavior. Few-shot prompting typically uses more tokens than zero-shot, so one of the key questions this system is designed to answer is whether the quality improvement justifies the additional token cost and latency.

3.4 Metrics Collected

For each experiment run, the system collects:

- **Latency:** End-to-end response time in milliseconds
- **Throughput:** Tokens per second and requests per second
- **Token Usage:** Prompt tokens, completion tokens, and total tokens
- **Energy Consumption:** Estimated kilowatt-hours based on configurable hardware assumptions
- **Energy Cost:** Estimated dollar cost of energy consumed
- **Hardware Cost:** Estimated hourly hardware cost
- **Output Quality:** Task-specific scores (accuracy, ROUGE, LLM-as-judge)

3.5 Challenges and Risk Assessment

Hardware variability poses a challenge because latency and throughput measurements are directly dependent on the executing machine's CPU and GPU resources, limiting cross-machine comparability. This was mitigated by recommending consistent hardware configurations across runs.

Sustainability metric accuracy is limited because energy and hardware cost values are estimated using configurable assumptions (GPU wattage, CPU wattage, energy cost per kWh) rather than directly measured. These values are intended for relative comparison rather than precise real-world reporting.

Dataset format compliance requires that input data conform to expected formats per task type. Improper formats can cause evaluation failures, particularly for sentiment classification tasks that require user-provided ground truth labels.

Concurrency risks were identified for simultaneous experiment runs. This was mitigated by executing runs sequentially to prevent race conditions and ensure consistent metric collection.

4. DESIGN

4.1 Architectural Overview

The system follows a layered architecture consisting of three primary layers: the Execution Layer, the Storage Layer, and the Presentation Layer. This structure divides benchmarking functionality, data management, and user interaction into clearly defined components, ensuring each responsibility is handled independently.

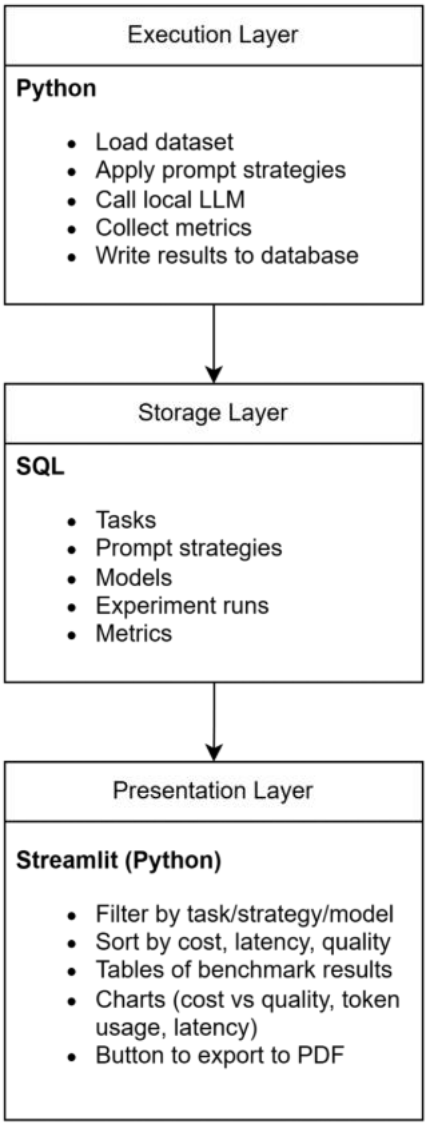


Figure 1. Software Architecture Diagram.

4.2 Architectural Strategies

Layered Architecture was chosen to improve modularity, maintainability, and team collaboration. By separating user interaction, benchmarking logic, and data persistence into distinct layers, the system can evolve more easily over time. Changes made in one layer do not impact others, and each layer can be developed and tested independently. This structure also allowed team members to work on different parts of the system simultaneously with minimal overlap, supporting efficient parallel development. A tightly coupled monolithic design was considered early on but rejected because it would reduce scalability, make testing more difficult, and create unnecessary dependencies between components.

Python Backend was selected as the primary language for the Execution Layer due to its strong ecosystem for machine learning, data processing, and LLM integration. Libraries for token counting, model interaction, and metric computation are readily available and well supported in Python, making it a natural fit for a benchmarking system. Other languages such as Java and Node.js were considered but not selected because they provide less direct support for LLM experimentation workflows and would introduce unnecessary complexity for rapid prototyping and benchmarking.

MySQL Relational Database was selected for the Storage Layer using a star schema design. The star schema centralizes all measurable experiment results in a fact table and links contextual metadata through dimension tables, enabling structured benchmarking analysis and reliable aggregation queries. A cloud-based NoSQL solution such as MongoDB or Firebase was considered but rejected because the project requires structured relationships and referential integrity that a relational database is better suited to provide. A cloud-based solution would also introduce unnecessary external dependencies for a locally focused benchmarking system.

Streamlit Frontend was chosen for the Presentation Layer to enable rapid development of interactive dashboards and data visualizations. Streamlit integrates naturally with Python, which reduces the need for complex frontend-backend separation and allowed the team to focus development effort on benchmarking functionality rather than UI infrastructure. A full React-based frontend was considered but rejected for this project scope. While React offers greater flexibility and scalability, it introduces additional architectural overhead and development time that are not necessary for a benchmarking prototype.

4.3 Execution Layer

The Execution Layer is the Python backend responsible for managing benchmark execution workflows and coordinating the core logic of the system. It handles the full lifecycle of an experiment run, including applying prompt strategies to locally hosted LLMs, collecting performance metrics per run, computing task-specific quality scores, and writing results to the database. This layer ensures that all data collected across runs is consistent and reproducible.

All model execution is handled locally using Ollama as the runtime environment. Ollama allows the system to run open-source large language models directly on the user's machine without requiring external API calls or cloud dependencies, making benchmarking repeatable and cost-free. The application itself is run using Streamlit, which serves as the entry point for the full system. Supported models include llama3.1:8b, mistral, gemma:7b, phi3, and llama3.1:3b, each of which must be pulled and available in Ollama before experiments can be executed. Model inference performance is directly dependent on the host machine's CPU and GPU resources, so hardware configuration plays a role in the latency and throughput metrics collected during each run.

4.4 Storage Layer

The Storage Layer uses MySQL with a star schema design. The central fact table is experiment_runs, which captures the measurable outcomes of each benchmark execution,

including references to the associated task, prompt strategy, model, dataset input, and runtime metrics. Surrounding the fact table are the dimension tables: tasks, models, prompt_strategies, dataset_inputs, and run_times, each connected to experiment_runs through foreign key relationships.

The schema also includes an experiments dimension table that groups related prompt variants together under a single named experiment. Each named experiment can contain multiple prompt variants, and all associated experiment_runs records are linked back to it through a foreign key relationship. This structure enables grouped comparison across variants within the same experiment and supports variance calculation when a prompt is executed multiple times across runs.

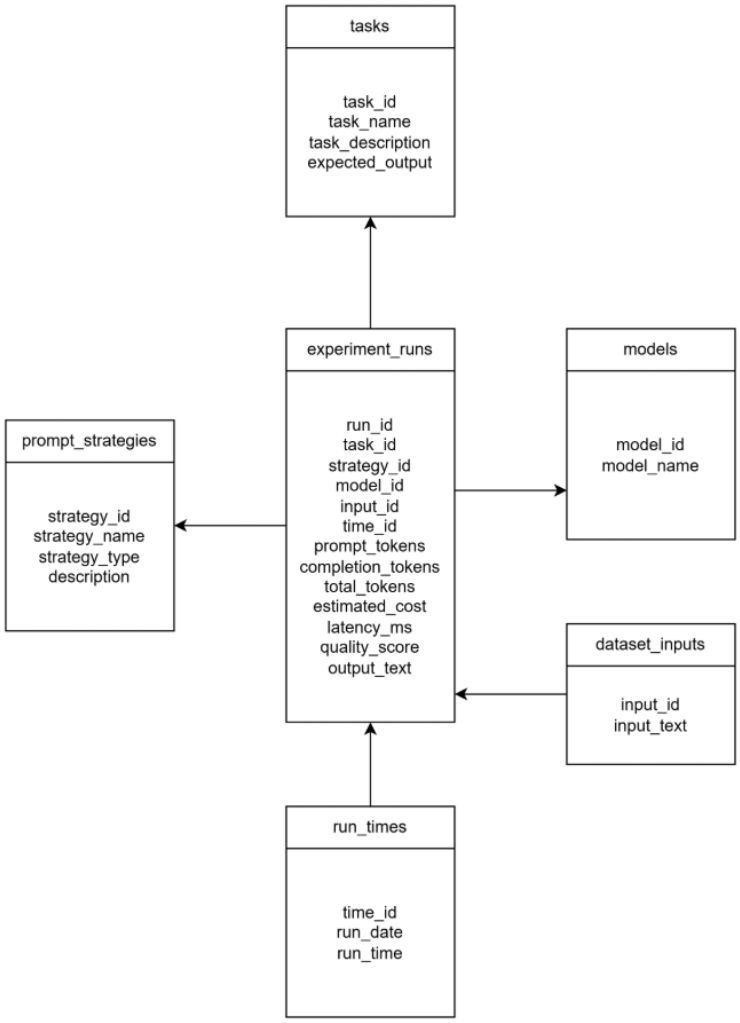
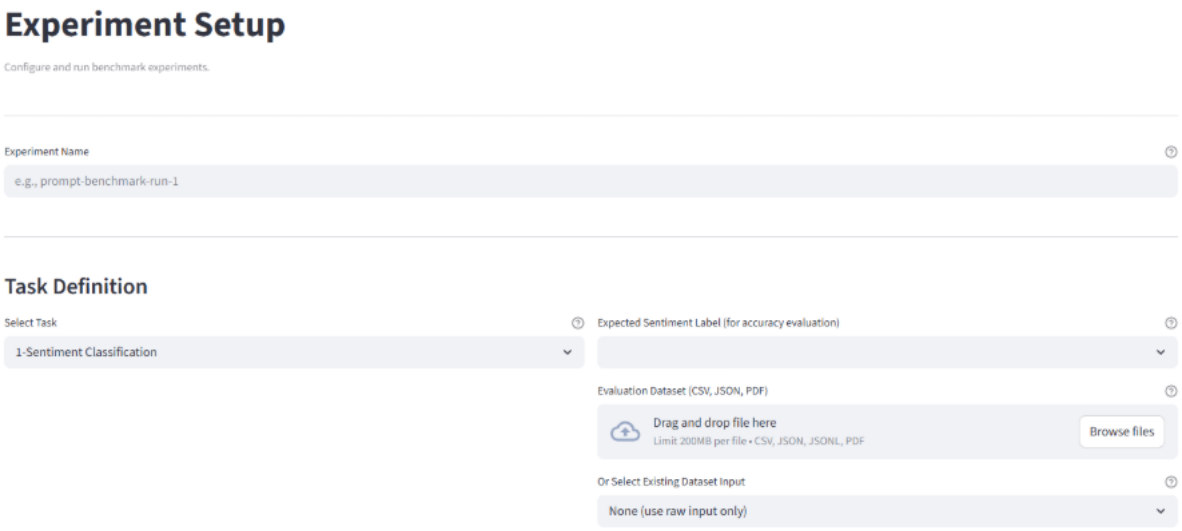


Figure 2. Database Star Schema.

4.5 Presentation Layer

The Presentation Layer is built using Streamlit and consists of three primary pages:

The Experiment Setup Page is where users configure and launch benchmarking experiments. Users begin by assigning a name to the experiment, selecting a task type, choosing a locally hosted Ollama model, and uploading a dataset in CSV, JSON, JSONL, or PDF format. From there, users configure one or more prompt variants for side-by-side comparison. Each variant includes editable fields for the system prompt, instruction prompt, and optional context or few-shot examples. Field labels adapt dynamically based on the selected task type to provide contextually relevant guidance. Tooltips are provided throughout the page to explain each field and configuration option, so users can understand what each field does without any prior experience with the system. Users may add multiple variants within a single experiment using the Add Prompt Variant button, and an existing variant can be duplicated using the Duplicate Variant button to support efficient replication of configurations with minor modifications. The Execution Configuration section at the bottom of the page allows users to set the number of runs per prompt variant, which controls how many times each variant is executed to support statistical comparison across runs.



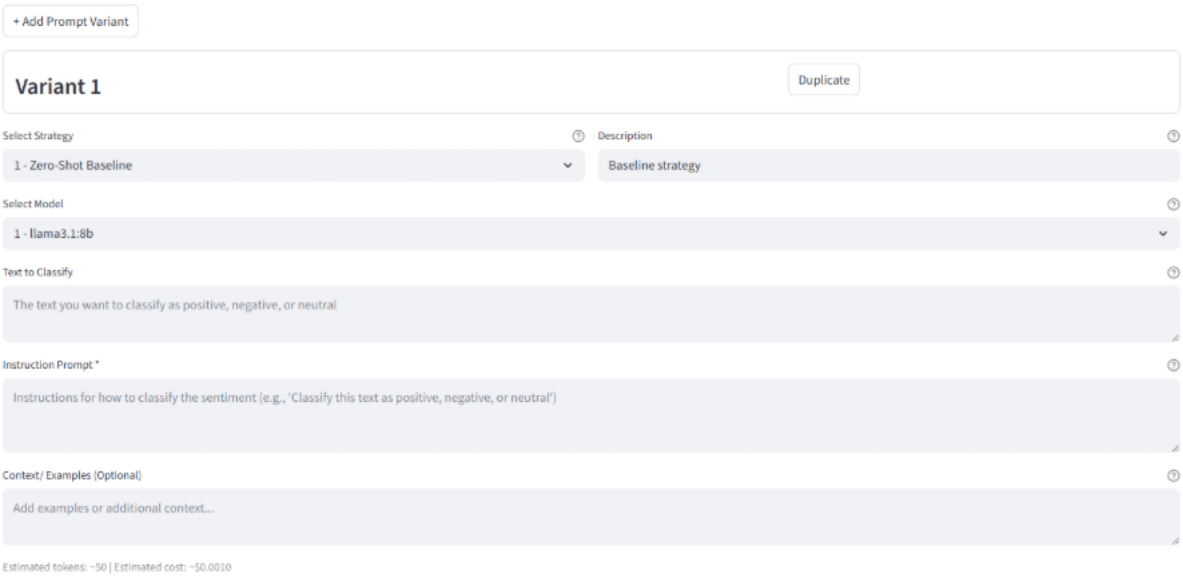
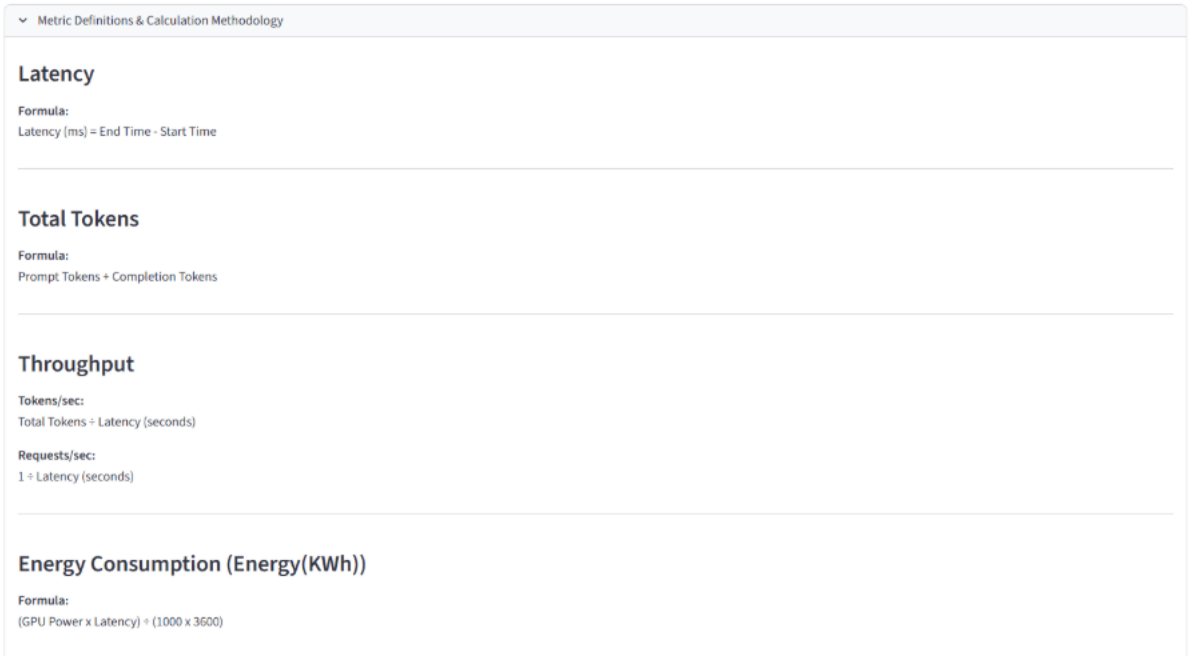


Figure 3. Experiment Setup Page.

The Results and Comparison Page displays benchmarking outcomes for all variants within an experiment. A summary section at the top of the page highlights the winning variant across five categories: Best Quality, Lowest Cost, Fastest, and Best Efficiency. Results are also presented in a tabular format with columns for cost, latency, token usage, quality score, variance, and efficiency, allowing users to compare prompt variants side by side in a structured and readable format. Chart-based visualizations are available as an alternative view for users who prefer a graphical representation of the results. A metrics formula reference section is included on the page to provide transparency into how each evaluation metric is calculated, so users can interpret results with full context. An aggregate efficiency score is also displayed, summarizing the overall cost-quality-latency trade-off across all variants to help users identify the best performing prompt strategy at a glance. Export options allow users to download results as CSV for further analysis outside of the platform.



Results & Comparison

Run #1 — Question Answering • Completed on Apr 27, 2026

Chat Table Charts

Export Report

Summary

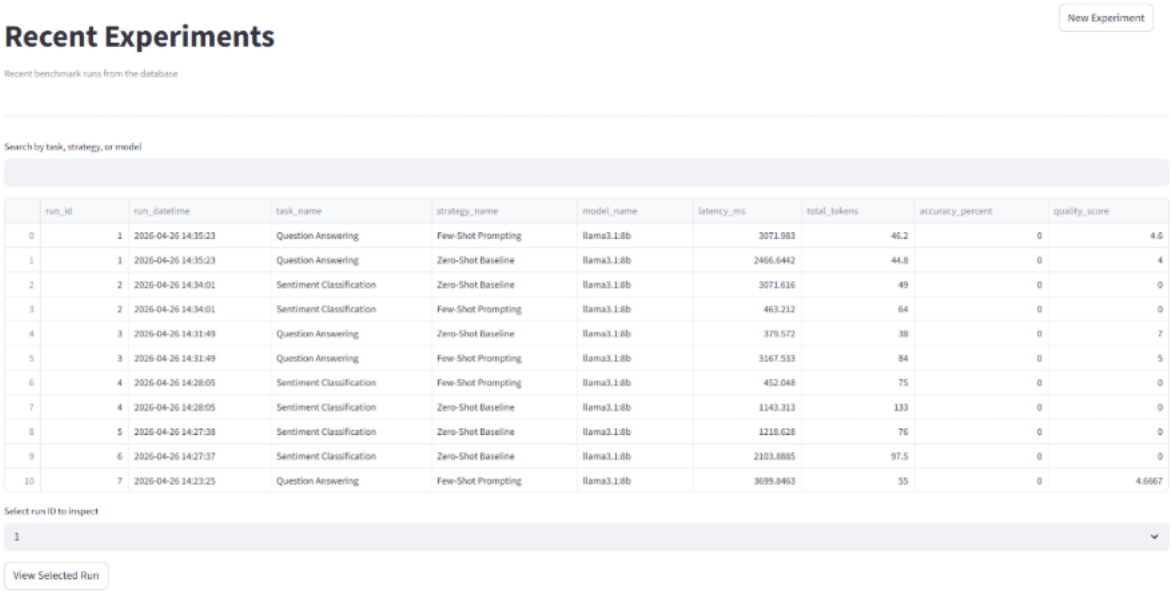
Best Quality	Lowest Energy	Fastest	Best Throughput	Lowest Tokens
4.6%	0.0000 kWh	2467 ms	N/A	44
↑ Few-Shot Prompting	↑ Zero-Shot Baseline	↑ Zero-Shot Baseline	↑ N/A	↑ Zero-Shot Baseline

Detailed Results

	variant	latency	quality	tokens	throughput	energy_cost
0	Few-Shot Prompting	3071.983000	4.600000	46.000000	None	0.000026
1	Zero-Shot Baseline	2466.644200	4.000000	44.000000	None	0.000020

Figure 4. Results and Comparison Page.

The Recent Experiments Page provides a searchable, filterable list of all past experiments. Each row in the table displays the experiment name, task type, variant count, average cost, average latency, date, and status. Users can search and filter historical runs by task, strategy, or model to quickly locate a specific experiment. A New Experiment button in the top right corner provides direct navigation back to the Experiment Setup Page, allowing users to start a new benchmarking run without having to navigate away from their experiment history.



	run_id	run_datetime	task_name	strategy_name	model_name	latency_ms	total_tokens	accuracy_percent	quality_score
0	1	2026-04-26 14:35:23	Question Answering	Few-Shot Prompting	llama3.1:8b	3071.983	46.2	0	4.6
1	1	2026-04-26 14:35:23	Question Answering	Zero-Shot Baseline	llama3.1:8b	2466.6442	44.8	0	4
2	2	2026-04-26 14:34:01	Sentiment Classification	Zero-Shot Baseline	llama3.1:8b	3071.616	49	0	0
3	2	2026-04-26 14:34:01	Sentiment Classification	Few-Shot Prompting	llama3.1:8b	463.212	64	0	0
4	3	2026-04-26 14:31:49	Question Answering	Zero-Shot Baseline	llama3.1:8b	379.572	38	0	7
5	3	2026-04-26 14:31:49	Question Answering	Few-Shot Prompting	llama3.1:8b	3167.533	84	0	5
6	4	2026-04-26 14:28:05	Sentiment Classification	Few-Shot Prompting	llama3.1:8b	452.048	75	0	0
7	4	2026-04-26 14:28:05	Sentiment Classification	Zero-Shot Baseline	llama3.1:8b	1143.313	133	0	0
8	5	2026-04-26 14:27:38	Sentiment Classification	Zero-Shot Baseline	llama3.1:8b	1218.628	76	0	0
9	6	2026-04-26 14:27:37	Sentiment Classification	Zero-Shot Baseline	llama3.1:8b	2103.8885	97.5	0	0
10	7	2026-04-26 14:23:25	Question Answering	Few-Shot Prompting	llama3.1:8b	3699.8463	55	0	4.6667

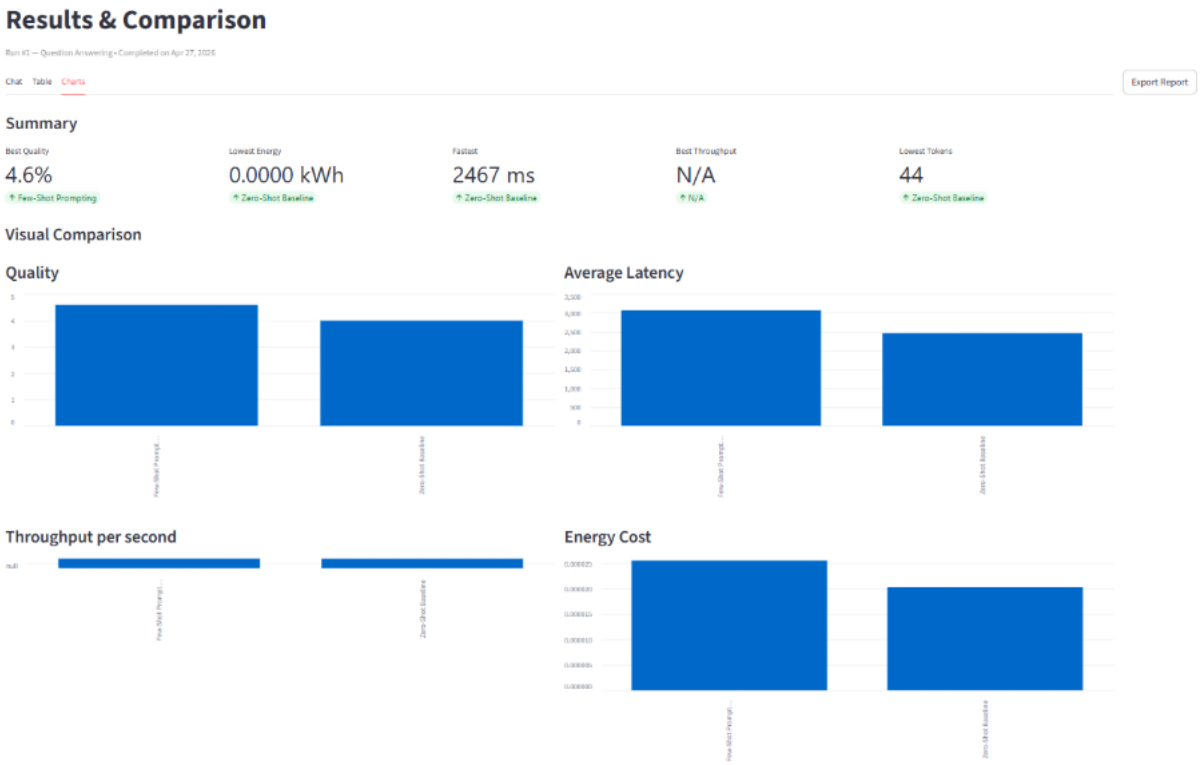


Figure 5. Recent Experiments Page.

5. DEVELOPMENT

5.1 Overview

Development followed an iterative, milestone-based approach broken into three phases. Each milestone built directly on the previous one, with adjustments made based on testing, advisor feedback, and sponsor review.

5.2 Milestone 1: Architecture and Planning

The first milestone focused on requirements gathering, system architecture design, benchmark task definition, and prompt strategy selection. The team produced the Software Requirements Specification (SRS), Software Design Document (SDD), and Figma wireframes for all three UI pages. The three-layer architecture (Execution, Storage, Presentation) was finalized, the star schema was designed, and the three benchmark tasks (sentiment classification, text summarization, question answering) and two initial prompt strategies (zero-shot, few-shot) were defined.

Key deliverables: SRS, SDD, architecture diagram, star schema diagram, Figma wireframes, GitHub repository initialization.

5.3 Milestone 2: Execution Engine and Logging

The second milestone focused on building the Python execution backend, implementing the local execution mode using Ollama, logging performance metrics per run, integrating task-specific scoring logic, and connecting results to the MySQL database.

At this stage, the team focused on a single benchmark task, sentiment classification, to validate the end-to-end execution pipeline. The sentiment task was selected because it has a well-defined input format, strict output format (exactly one label: Positive, Negative, or Neutral), and automated scoring logic using exact match accuracy.

Example inputs validated during Milestone 2:

- Input: "I love this product. It works amazing." Expected: Positive
- Input: "This is the worst service I've ever experienced." Expected: Negative
- Input: "The package arrived yesterday." Expected: Neutral

The execution engine was implemented in Python and connected to the MySQL database schema. Performance metrics (latency, token usage, estimated cost, quality score) were logged per run. The team also identified future work areas including expanded dataset evaluation, visualization improvements, and additional task and strategy support.

Key deliverables: Python execution engine, local execution mode, task-specific scoring module, database integration, initial end-to-end demo.

5.4 Milestone 3: Finalization and UI Improvements

The third and final milestone extended the platform to support all three final task types (sentiment classification, summarization, question answering) and finalized both prompt strategies (zero-shot and few-shot). Significant UI improvements were implemented, including:

- Added a metrics formula reference section on the results page
- Improved result visualization with charts and summary cards
- Improved wording for experiment setup fields
- Added tooltips on the experiment setup page
- Added a duplicate variant button for efficient configuration replication

A comprehensive README was developed alongside the codebase documenting all prerequisites, installation steps, database initialization, Ollama model setup, configuration options, and troubleshooting guidance to ensure reproducibility.

Key deliverables: Full three-task platform, two finalized prompt strategies, UI improvements, README, final demo.

5.5 Prompt Construction

At the core of the execution engine is the prompt builder module, which is responsible for assembling the final prompt that gets sent to the LLM for each experiment run. Regardless of the

strategy selected, the system always constructs one complete prompt per run and sends it to the locally hosted Ollama model in a single request.

For zero-shot prompting, the final prompt consists of the system prompt and instruction prompt as configured by the user, with the input text appended directly. No examples are provided. The model relies entirely on its pretrained knowledge and the instructions given to produce an output.

For few-shot prompting, the process is the same except that the user-provided examples are parsed and appended to the prompt before the input text is sent. When a user enters examples in the Context/Examples field during experiment setup, the system automatically parses them and includes them in the constructed prompt so the model receives both the examples and the new input together in one request. This allows the model to learn the expected output format and reasoning pattern from the examples before generating a response.

5.6 Configuration and Metrics Estimation

Runtime configuration for the system is managed through the settings.json file located in the benchmarking_backend/config/ directory. This file controls database connection settings, default benchmark parameters such as the number of repetitions per run and default batch size, and the hardware assumptions used to estimate energy and cost metrics.

Energy and hardware cost metrics are not measured directly but are estimated using the assumptions defined in settings.json. These include GPU wattage, CPU wattage, energy cost per kilowatt-hour, hardware hourly cost, and carbon emissions per kilowatt-hour. Because these values vary depending on the user's hardware environment, they are intended to support relative comparisons across experiments rather than precise real-world cost reporting. Users can modify these values before running experiments to better reflect their actual hardware setup.

6. TESTING AND VALIDATION

6.1 Overview

Each task type was tested end to end to validate that the execution pipeline, scoring logic, and database logging were all functioning correctly. All three tasks were tested using both zero-shot and few-shot strategies to confirm that the full execution pipeline functioned correctly across all supported combinations.

6.2 Sentiment Classification

Sentiment Classification was the first task validated and served as the primary test case for the execution pipeline during Milestone 2. Because the output requirement is strict, exactly one label of Positive, Negative, or Neutral, it was straightforward to verify whether the system was scoring correctly using exact match accuracy.

Zero-Shot Example:

- Input: "I love this product. It works amazing."
- Strategy: Zero-Shot, instruction prompt only with no examples provided
- Expected Output: Positive
- Actual Output: Positive
- Score: 1 (Correct)

Few-Shot Example:

- Input: "I love this product. It works amazing."
- Strategy: Few-Shot, the following examples were appended to the prompt before the input:
    - Example 1: "This movie was absolutely terrible." → Negative
    - Example 2: "The weather today is okay." → Neutral
- Expected Output: Positive
- Actual Output: Positive
- Score: 1 (Correct)

Validation confirmed that the model returned a single label per run, that the accuracy scoring logic correctly compared the predicted label against the ground truth, and that results were written to the database as expected. Negative testing was also performed to verify that the system handled invalid or unexpected inputs gracefully. This included inputs that did not clearly align with any of the three sentiment labels, confirming that the system behaved consistently under edge case conditions.

6.3 Text Summarization

Text Summarization was validated by providing a passage of text and comparing the model-generated summary against a reference summary. Because summarization produces open-ended output, exact match scoring is not applicable. Validation focused on confirming that ROUGE scores were computed correctly and that the LLM-as-judge quality score was returned and stored per run. Results were reviewed alongside the ROUGE scores to confirm that the scoring reflected the quality of the output.

Zero-Shot Example:

- Input: "NASA's Artemis program aims to return humans to the Moon by the mid-2020s. The program includes a series of missions designed to establish a sustainable human presence on and around the Moon, which will serve as a stepping stone for future missions to Mars."
- Strategy: Zero-Shot, instruction prompt only asking the model to summarize the passage
- Expected Output: A concise summary capturing that NASA's Artemis program plans to return humans to the Moon and use it as a foundation for future Mars exploration
- Evaluation: ROUGE scores measuring overlap with the reference summary and an LLM-as-judge quality score assessing correctness and completeness

Few-Shot Example:

- Input: Same passage as above
- Strategy: Few-Shot, an example summary was appended to the prompt before the input:
    - Example: "The Apollo program successfully landed humans on the Moon six times between 1969 and 1972." → "Apollo landed astronauts on the Moon multiple times in the late 1960s and early 1970s."
- Expected Output: A concise summary in a similar style to the provided example
- Evaluation: ROUGE scores and LLM-as-judge quality score, with few-shot expected to produce output more closely aligned with the reference summary format

6.4 Question Answering

Question Answering was validated by providing a context passage alongside a question and verifying that the model's response was grounded in the provided context. Validation confirmed that the Exact Match and F1 Score logic correctly evaluated the predicted answer against the reference answer, and that the LLM-as-judge quality score assessed grounding and correctness as intended.

Zero-Shot Example:

- Context: "The Eiffel Tower is located in Paris, France. It was constructed between 1887 and 1889 and stands 330 meters tall."
- Question: "Where is the Eiffel Tower located?"
- Strategy: Zero-Shot, instruction prompt asking the model to answer based only on the provided context
- Expected Output: Paris, France
- Actual Output: Paris, France
- Exact Match: 1 (Correct)

Few-Shot Example:

- Context: "The Eiffel Tower is located in Paris, France. It was constructed between 1887 and 1889 and stands 330 meters tall."
- Question: "Where is the Eiffel Tower located?"
- Strategy: Few-Shot, the following example was appended to the prompt before the input:
    - Example Context: "The Colosseum is located in Rome, Italy."
    - Example Question: "Where is the Colosseum located?"
    - Example Answer: "Rome, Italy"
- Expected Output: Paris, France
- Actual Output: The Eiffel Tower is located in Paris, France
- Exact Match: 0 (F1 Score captures partial credit for near-correct responses)

Validation of the Question Answering task also confirmed that the LLM-as-judge evaluator correctly assessed responses for correctness, completeness, and grounding relative to the provided context, and that all results were stored in the database as expected.

7. VERSION CONTROL

7.1 Approach

The team used GitHub for all version control throughout the project. The repository was initialized during Milestone 1 and served as the central collaboration point for all three team members across the full development lifecycle.

7.2 Branching Strategy

The team used feature branches throughout development, with each member working on their own branch to avoid conflicts. Once work was completed and reviewed, branches were merged together before being pushed to the main branch at the end of each milestone. The main branch always reflected the most recent stable version of the application.

8. SUMMARY

The Prompt Optimization Workbench for AI Benchmarking delivers a structured, reproducible platform for evaluating and comparing prompt strategies across generative AI tasks. The system implements a three-layer architecture consisting of an Execution Layer, Storage Layer, and Presentation Layer that supports modular development and clean separation of concerns.

The final platform supports three benchmark tasks, sentiment classification, text summarization, and question answering, evaluated across two prompt strategies, zero-shot and few-shot, using metrics including latency, throughput, token usage, energy consumption, and task-specific output quality scores. All model execution is handled locally through Ollama, keeping benchmarking cost-free and reproducible on any machine with the required models installed. The Streamlit-based interface allows users to configure experiments, view side-by-side comparisons, and export results, with tooltips and a metrics formula reference section supporting non-technical stakeholders.

Development was completed across three milestones, with each phase building directly on the previous one. Version control was managed through GitHub, with each team member working on their own branch throughout development and all final work merged to the main branch at the end of each milestone.

Future work includes running benchmarking experiments across larger datasets for more statistically meaningful results, adding chain-of-thought as a third prompt strategy, expanding chart-based visualizations, and improving experiment tracking and reproducibility options.

9. APPENDIX

A. Project Plan

1.0 Project Overview
As generative AI grows, organizations face increasing challenges in predicting costs, managing latency, and reducing environmental impact. This project presents a Prompt Optimization Workbench that benchmarks prompt strategies across LLMs. By executing side-by-side prompt technique comparisons such as zero shot and few shot prompting, the system measures token usage, execution latency, and estimated resource consumption. The results are visualized to help technical decision makers understand trade-offs and optimize prompt design. We conclude that prompt benchmarking is essential for responsible AI deployment.

2.0 Project website
https://destinyj621.github.io/

Deliverables - Specific To Your Project
- Project Selection document (Individual Assignment)
- Weekly Activity Reports (Individual Assignment)
- Team Status Report (Group Assignment)
- Peer Reviews (Individual Assignment)
- Project Plan (Group Assignment)
- SRS and SDD (Group Assignment)
- Bi-weekly Project updates with mentor (Group Assignment)
- Final Report Package (Group Assignment)
  - Final Report (Group Assignment)
  - Source Code (Group Assignment)
  - Website (Group Assignment)
  - Video Demo (Group Assignment)
  - Benchmarking platform
- C-Day Application Submission

Milestone Events (Prototypes, Draft Reports, Code Reviews, etc)
#1 Requirements & Benchmark Design (Weeks 1-4)
- Requirements & Design Document
- Defined task types and datasets
- Prompt strategy taxonomy
- Metrics definition and justification
- Evaluation rubric for output quality
- UI wireframes
- Sponsor review and approval

Key Focus:
Framing the problem correctly and agreeing on what *"better"* means.

#2 Execution Engine & Data Collection (Weeks 5-10)

- Prompt execution engine (local + estimation mode)
- Metrics logging subsystem
- Strategy comparison framework
- Initial experiment results
- Backend API (FastAPI or Flask)
- Database schema and persistence

Key Focus:
Reliable, repeatable benchmarking.

#3 Visualizations, Analysis & Final Demo (Weeks 11-15)

- Interactive UI dashboard (Streamlit or React)
- Comparative visualizations
- Final benchmark runs
- Written analysis and conclusions
- Documentation and setup instructions
- Final sponsor demo and presentation

Key Focus:
Clear communication of results and insights.

Meeting Schedule Date/Time
Meetings are scheduled for Saturdays at 1:00 PM.

Collaboration and Communication Plan
The team will conduct weekly meetings via Discord, beginning at 1:00 PM. Currently meetings are scheduled for Saturday; however, this day may be adjusted as needed. The frequency of these meetings may increase should project demands arise.

Project Schedule and Task Planning

Phase	Tasks	Complete%	Current Status Memo	Assigned To	01/26	02/02	02/09	02/16	02/23	03/02	03/09	03/16	03/23	03/30	04/06	04/13	04/20	04/27
Requirements	Meet with stakeholder(s) SH	100%	Complete	Faith	1													
	Define requirements	20%		Destiny	2	2												
	Review requirements with SH	20%		Jamia		2	2											
	Get sign off on requirements	0%		Faith			1											
Project design	Website Design	0%		Faith						4								
	UI Wireframes	0%		Jamia				1										
	Database design	0%		Destiny				2	2	2								
	Metrics definition and justification	0%		Faith					2	2	2							
	Prompt strategy taxonomy	0%		Jamia					3									
	Develop working prototype	0%							5	5	5	5						
	Test prototype	0%										5						
Development	Review prototype design	0%										3	3	3				
	Rework requirements	0%										2	2	2				
	Document updated design	0%												2	2			
	Test product	0%												2	2	2		
Final report	Presentation preparation	0%															3	
	Poster preparation	0%															3	
	Final report submission to D2L and project owner	0%																1
			Total work hours	87	3	4	3	3	12	13	7	15	5	9	4	2	6	1

Other Plans – Like Risk Assessment (if applicable)
This project will include a risk assessment to identify, evaluate, and mitigate risks associated with prompt-level benchmarking of AI workloads. Risks associated with this project may include

limited availability to APIs, security risks, and compliance risks. These risks will be mitigated by limiting external API usage to approved environments. Ethical risks in this project may include the potential storage of harmful or sensitive data in the database.

Version Control Plan

The team will utilize a single GitHub account for managing the project. The version control plan will consist of several key branches to facilitate development, testing, and the release processes.

Branch Structure:

- Final Branch: This branch will remain empty only to be populated once the project is fully completed. This branch is a representation of the user ready benchmarking platform.
- Development Branch: This branch will hold code that is actively being developed and not stable for production.

Commit Practices:

Commits should be made after the completion of a feature. Each commit message should Include the developer's name and a brief description of the implemented feature. This will help in tracking changes and maintaining clear documentation on the progress of the project.

Merging Guidelines:

Development Branch: The development branch will be merged to the main branch after all testing is finalized and the platform is working as intended. This signifies that the platform is ready to be deployed.

Generative AI tools, including ChatGPT 5.2, were used to assist with wording refinement, structural clarification, and editing of this document. Figma AI features were used to generate initial UI wireframe drafts. All design decisions, system architecture, benchmarking methodology, and implementation details were developed and verified by the project team.