Capstone Software Requirements Specification + Project Development Plan

Version 0
Group 7 - LM Compass
COMPSCI 4ZP6
Dr. Mehdi Moradi - Dr. Angela Zavaleta Bernuy
October 10th, 2025

Sohaib Ahmed	ahmes127@mcmaster.ca	Coordinator/Researcher
Madhav Kalia	kaliam1@mcmaster.ca	Backend Engineer
Aadi Sanghani	sanghana@mcamster.ca	Backend Engineer
Gulkaran Singh	singg36@mcmaster.ca	Fullstack Engineer
Rochan Muralitharan	muralr3@mcmaster.ca	Fullstack Engineer
Owen Jackson	jackso4@mcmaster.ca	AI Engineer
Aryan Suvarna	suvarnaa@mcmaster.ca	Frontend Engineer

Table of Contents

1. Contribution History

Name	Section
Sohaib	3, 4.1, 4.2, 4.3, 4.4, 7.1, 7.2, 8.3, 11.1, 11.2, 12
Madhav	7.1, 7.2, 7.3, 7.4, 13.5, 14, 16
Aadi	7.1, 7.2, 7.3, 7.4, 13.1, 13.2, 13.3, 13.4
Gulkaran	6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 10.1, 10.2
Rochan	5.1, 5.2, 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7
Owen	6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 8.1, 8.2, 8.3
Aryan	9.1, 9.2, 9.3, 9.4, 9.5

2. Revision History

Version	Description	Date
0	Initial SRS + Project Plan	2025-10-10

3. Glossary

The "System"	The overall application or platform being developed.
A "User"	An individual interacting with the system.
LLM	Large Language Model. A powerful AI model used for various language understanding and generation tasks. Generally proprietary
SLM	Small Language Model. A more compact and efficient AI model compared to an LLM, potentially used for specific, less complex tasks within the system. Generally open source.
LLM-As-A-Judge	A methodology where an LLM is used to evaluate the quality or relevance of outputs, often from other AI models or systems.
Generative AI	Artificial intelligence that can produce new content, such as text, images, or code, rather than just analyzing existing data.
Prompt	The input text or query provided to an AI model to elicit a specific response.
Query	A request for information or data, often in the form of a prompt to an AI

	model or a search request within the system.
Consensus	Agreement reached among multiple evaluators (human or AI) regarding the quality or correctness of an output.

4. Purpose of the Project

4.1 Background

Generative AI is expanding rapidly, but evaluating the quality of responses from Large and Small Language Models (LLMs/SLMs) remains a challenge, so it’s difficult to understand which models to use for a query. Current methods are mostly subjective or based on narrow benchmarks, limiting fair comparison and practical adoption.

4.2 Problem Statement

Users and researchers lack a scalable tool to receive the ‘best’ response to a query when evaluated against different models, and therefore cannot objectively determine which models are best suited to respond to a given query.

4.3 Project Objectives

The goal of the project is to:

- Research and determine how we can build a system which objectively decides on a ‘best’ response given a query and multiple models.
- Allow users to input queries and receive outputs from multiple models.
- Implement LLM-as-a-judge cross-evaluation where models assess each other’s answers.
- Implement human intervention/feedback for when the user disagrees with the system’s assessment
- Provide a consensus “best answer” and ranking of responses with a score-based grading of all responses
- Enable integration of new models through API keys.
- Evaluate the performance of our system through various metrics and situations built for testing LLM-as-a-judge systems

4.4 Proposed Solution

We will develop a web-based application that automates model response evaluation and presents results through a user-friendly interface. Using an LLM-as-a-judge based cross-model evaluation, the system will identify the ‘best’ response for a given query while remaining scalable and adaptable. We will also perform experiments and record our findings in a notebook to evaluate the effectiveness of our system.

5. Client and Other Stakeholders

5.1 Primary Stakeholders

- Project Supervisors
  - Dr. Angela Zavaleta Bernuy
  - Zezhu Yu
- Course Stakeholders
  - Professor Mehdi Moradi
  - Amirhossein Sabour
  - Sahib Khokhar
- End Users
  - Researchers and Developers interested in evaluating and comparing LLMs and SLMs
  - Organizations and AI teams that require an unbiased, scalable system to determine the best model response for a given query

5.2 Secondary Stakeholders

- *Model Providers:* Companies or institutions providing access to LLMs or SLMs via API keys (eg. OpenAI, Anthropic)
- *Academic Community*: Students and instructors who can use the platform as a learning tool to study model evaluation techniques (LM-as-a-Judge)
- *General Users*: Individuals who are interested in experimenting with generative AI performance or want to derive the best possible answer for their query

6. Project Constraints and Relevant Facts

6.1 Solution Constraints

Final product must have:

- Support to get query results from up to X LLMs and have them all do the analysis
- The application must present results in a standardized, user-friendly interface
- Will only handle general short answer questions, not discipline specific questions to avoid switching between different accuracy evaluation methods

6.2 Schedule Constraints

- A working prototype demonstrating query submission and multi-model evaluation must be available by Nov 21, 2025
- A near-final version should be ready for stakeholder testing by February 2026.
- The completed system must be ready for presentation by March 27, 2026
- The completed system must be submitted for course evaluation on April 4, 2026.

6.3 External Application Constraints

- The system must support multiple LLMs and SLMs (e.g., OpenAI, Anthropic, Meta, Mistral) through their APIs
- OpenRouter will be used to access multiple LLMs and SLMs
- The system is constrained to operating only with LLMs/SLMs for which the user or the development team has valid API keys

6.4 Environment Constraints

- The application must be deployed as a web-based solution to ensure accessibility across devices and operating systems
- Hosting is to use a cloud environment (e.g., AWS, Azure, or Google Cloud), but the final provider will depend on supervisor approval and cost considerations

6.5 Budget Constraints

- Cost for hosting and other RnD must be kept to a maximum of $125 by the end of the semester. Model usage is not covered by this budget.
- Additional infrastructure costs (e.g., hosting, databases) must be minimized and scalable.

6.6 Communication and Collaboration Constraints

- Weekly/Bi-Weekly supervisor check-ins on Mondays 11 AM - 12 PM EST
- Internal scrum meetings on Tuesdays 5 PM EST
- GitHub Projects Board for task tracking

6.7 Relevant Facts

- *Extension of Existing Software*: Existing evaluation platforms (e.g., LMSYS Chatbot Arena) provide similar benchmarking but lack the cross-model, automated consensus evaluation approach required for this project. The purpose of this system is to create an independent, open, and scalable alternative for accuracy-focused evaluation

7. Functional Requirements

7.1 P0 Requirements

- Submit Query
    - Description: Users must be able to submit a free-text prompt (and optional context) to start an evaluation run.
    - Frontend: A prompt textbox with max length (character count/limit), and a “Run” button
    - Backend: Validate payload and pass the query to the model.
- Model API Key Vault & Provider Routing
    - Description: Users must be able to input and manage API keys for various LLMs and SLMs
    - Frontend: Interface to insert API keys and select the models to be used
    - Backend: Functionality to securely store and retrieve API keys, and to route requests to the appropriate model based on user selection.
- Candidate Generation (Multi-Model)
    - Description: System sends the same prompt to all selected models and collects the answers
    - Frontend: Shows the prompt sent to all models and progress is shown per model.
    - Backend: Send the queries to all the selected models.
- Judging & Scoring Engine
    - Description: The system must implement LLM-as-a-judge to evaluate queries and pick the best response

    - Frontend: N/A
    - Backend: Define accuracy metrics and criteria, pass the responses to the judge.

7.2 P1 Requirements

- User input on non-consensus
    - Description: The system should allow users to select their preferred response if the system cannot come to a consensus.
    - Frontend: Interface to select the best response out responses A or B.
    - Backend: Functionality to record user's preferred response and surface the selected response.
- User input on responses
    - Description: Users should have the ability to up/down-vote answers, suggest corrections, and mark issues.
    - Frontend: Feedback widgets on each answer.
    - Backend: Store feedback with labels, and give the context to the LLM/SLM
- Rubric Management
    - Description: Users should have the ability define rubrics with criteria for the LLM-as-judge
    - Frontend: Rubric CRUD UI.
    - Backend: Store rubric versions and pass that information to the LLM-as-judge.
- Query and Result History
    - Description: The system should retain the information of past chats, and evaluations.
    - Frontend: UI that shows past history of the chats.
    - Backend: Store the chats to a DB.

7.3 P2 Requirements

- Share/Export Results
    - Description: Users should be able to easily copy, or export the response(s) (e.g., as text, markdown, csv)
    - Frontend: Copy/export button
    - Backend: Generate copied content; format and return data based on export type.
- User authentication system
    - Description: Users should be able to create an account and log in.
    - Frontend: Login form (username/email, password), "Sign Up" link, "Forgot Password" link, session management (cookies/local storage).
    - Backend: User registration, login validation, password hashing, session management, user data storage.
- Model Selection for open sourced models
    - Description: Users should be able to use and manage various open source models to query
    - Frontend: Multi-select of available models, dropdown menu
    - Backend: Persist selected models

7.4 P3 Requirements

- User feedback mechanism
    - Description: Users could have the ability to provide feedback on the overall system (Bug reports, requests, etc.)
    - Frontend: Feedback form with text area and submission button, success/error messages.
    - Backend: Store feedback, associate with user/timestamp, categorize (bug/feature), notify admins.
- Multimodal inputs
    - Description: The system could accept multimodal queries (i.e. images, files)
    - Frontend: Integrated in base chat function
    - Backend: System processes and interprets diverse input types (e.g., images, files) to fulfill multimodal queries.

8. Data and Metrics

8.1 Data for Training/Building Features

- No training will take place for this algorithm. The goal is to create an accurate way of evaluating models responses in relationship to each other, not with external data
- Data will be used in the process of testing our evaluation method's accuracy

8.2 Data Acquisition/Simulation Plan

- Data will have two main features, question and answer, so that correct answers can compared with the output produced by our model with the respective question input

8.3 Performance Metrics and Goals

- We will incorporate a research and evaluation component separate from our application to demonstrate the effectiveness of our system compared to a baseline LLM. This will involve identifying and selecting 2-3 specific domains with existing ground truth.
    - Our system and a baseline LLM will process queries within these domains, and their outputs will be compared against the established ground truth.
- For each domain, we will determine appropriate, problem-specific performance metrics (primarily quantitative), to assess the accuracy, relevance, and overall quality of our system's responses relative to the baseline LLM.
    - This will be an ongoing effort as we progress with this project.
- While not the primary focus, we will also evaluate certain qualitative characteristics of our system, which may include surveying human agreement with responses and assessing response consistency.

9. Non-functional Requirements

9.1 Look and Feel Requirements

9.1.1 Appearance Requirements:

- Clean, minimalist look with readable typography; supports light/dark themes; maintains accessible color contrast.
- Desktop-first responsive design (works well on laptop screens; usable on tablets)

- Clear results hierarchy: show the selected "winner" first, then the ranked model outputs.
- Outputs are easy to read and compare (monospace blocks, preserved formatting, quick "copy" action).
- Consistent buttons, cards, and icons across pages; obvious loading, empty, and error states.
- Supports 200% zoom with proper reflow (no overlap or two-direction scrolling).

9.1.2 Style Requirements:

- Utilitarian, research-tool vibe: low-chrome UI, neutral palette, clarity over decoration.
- Consistent typography: one clean sans-serif for UI, monospace for model outputs; sentence case labels (no ALL CAPS).
- Simple charts: no 3D or heavy effects; clear labels and legends.

9.2 Usability and Humanity Requirements

- *Learnable from walkthrough:* Guided "first run," example prompts, and an in-app glossary so users can submit, compare, and interpret results without prior setup.
- *Clear feedback & recovery*: Visible run states (queued/running/completed/failed), progress, cancel, and non-blocking toasts; errors preserve input and offer retry/details.
- *Accessible by default:* Full keyboard navigation, visible focus, screen-reader labels, and good color/contrast across light/dark themes; supports 200% zoom with proper reflow.
- *Consistent & efficient:* Predictable buttons/dialogs/tables; Enter = run, Shift+Enter = newline; shortcuts for copy/compare; templates/recents to avoid repetitive setup.
- *User control & respect:* Easy export/delete of runs; "do not retain my data" option at submission; plain-language labels and explanations for metrics.

9.3 Performance and Speed Requirements

- Core reads (runs list, run detail, scores) ≤ 250 ms; writes (submit/cancel) ≤ 400 ms (excluding model/provider latency).

9.4 Security and Privacy Requirements

- Store API keys encrypted at rest; never log raw keys or full model outputs in plaintext logs.
- Secret management via env vault; least-privilege cloud roles; audit trail for admin actions.

9.5 Legal Requirements

- Respect model provider ToS for routing/usage; display disclaimer that rankings are experimental and may reflect bias.
- User consent for storing prompts/outputs; link to privacy policy; optional "do not retain my data" toggle.
- If any open-source components are used, track licenses (MIT/Apache/GPL) and attribution in About page; no ingestion of copyrighted datasets without permission.

10. Risks and Predicted Issues

10.1 Technical Risks

Risk	Description	Likelihood	Impact	Mitigation Strategy
Evaluation Criteria Drift	The scoring algorithm or rubric may produce unstable rankings when testing the same inputs repeatedly.	Medium	Mediu m	Version all scoring logic and record which version was used for each evaluation for reproducibility.
Data Security and Privacy Risks	Storing API keys and model outputs poses risks of data exposure if not properly encrypted or managed.	Low	High	Store all secrets encrypted at rest, avoid logging sensitive data, and apply the principle of least privilege for database access.
API Reliability and Rate Limits	Third-party APIs (OpenAI, Anthropic, etc.) may fail, throttle requests, or change pricing/availability unexpectedly.	Low	High	Implement retry mechanisms, caching of recent responses, and allow fallback to open-source SLMs when possible.

10.2 Predicted Issues

Issue	Description	Planned Resolution
Bias in Model Scoring	Different LLMs may inherently favor their own output or language patterns during cross-evaluation.	Anonymize the responses during judging, and compare multiple judges (cross-model).
Subjective Disagreement with Rankings	Users may disagree with the “best” response identified by the system.	Allow users to override the consensus, and log disagreements (e.g. like/dislike buttons) to refine the judging algorithm.

Unexpected API Costs	Frequent API calls to paid models can exceed a reasonable budget.	Set hard limits per evaluation run and encourage testing with open-source or low-cost APIs. Implement test fixtures when testing our software.

11. Team Meeting and Communication Plan

11.1 Team Meeting Plan

The team will hold weekly online meetings with the supervisor via MS Teams on Mondays at 11 AM. Additionally, the team will connect with each other online via Discord on Tuesdays at 5 PM. For any team members unable to attend, 1:1 catch-up sessions will be scheduled as needed. All other work will be conducted asynchronously.

11.2 Team Communication Plan

The team will use a Discord group chat for internal communication. For discussions with the supervisor, a Teams group chat will be utilized. Should any urgent issues arise outside of scheduled meetings, the team has agreed to hold online working sessions via Discord. GitHub Projects/Issues will be used to track tasks, project requirements, and deliverables. All new tasks or bugs will be logged, tracked, and assigned based on domain expertise and availability.

12. Team Member Roles & Responsibilities

Name	Role	Responsibility
Sohaib Ahmed	Coordinator/ Researcher	Coordinating team communication, managing tasks/issues in GitHub. Researching cross-model evaluation logic (LLM-as-a-judge strategies), and evaluation metrics.
Madhav Kalia	Backend Engineer	Implementation of secure API routing, database integration, multi-model query handling.
Aadi Sanghani	Backend Engineer	Implementation of scoring/judging engine, and secure management of user data.
Gulkaran Singh	Fullstack Engineer	Implementation of query submission workflow, and response history.
Rochan Muralitharan	Fullstack Engineer	Implementation of user-intervention when the system cannot reach a consensus response.
Owen Jackson	AI Engineer	Refining and implementing model evaluation methods from research into experiments.

Aryan Suvarna	Frontend Engineer	Implementation of user experience elements, including prompt submission, model selection, and feedback components.

13. Workflow Plan

13.1 Version Control

For version control we will be primarily using Github, where we will have a main branch and subsequent feature branches. We have also configured our main branch so that there needs to be at least review done to a branch before that can be merged into main. Additionally, once we start hosting the application, we may need a staging branch as well to prevent directly pushing to production and to enable testing first on staging branches with other features.

13.2 Agile Methods

We follow Agile methods with weekly Scrum meetings to review the tasks completed in the previous week and plan priorities for the upcoming week. Our team is also divided into sub-groups based on specific sub-tasks, and these sub-teams hold planning sessions to align with the bigger project. For example, full-stack engineers meet to prioritize the UI tasks and coordinate with the Backend engineers to ensure integration between UI and backend.

13.3 Data Storage

For data storage we will primarily be using google drive to share files and documents with each other. In terms of data storage for our application, we will be using Supabase, which is PostgresSQL under the hood, where we will store relational data, and Supabase also has buckets, where we would store documents type objects.

13.4 Compute-Heavy Task Execution

In most cases for compute-heavy task execution we will be using the department issued GPU servers, which should be more than enough computing for our project. In the very worst case, we can also rely on AWS or any other cloud computing providers and use computing from one of the providers.

13.5 Tools and Methods for Requirements and Metrics

We will be using Github Projects to track our requirements. All the requirements are created as tickets with sub issues on Github and tickets are assigned to each individual. Github projects also track the progress of each ticket (Todo, In Progress, Review, Complete) which we are using. Each requirement will also have a priority associated with it and a deadline to make sure we follow our team's time line. For metrics we will be using Unit Testing and Integration Testing as we are building our model.

14. Proof of Concept Demonstration Plan

For the proof-of-concept we will demonstrate the end-to-end flow of how the system works. The user will submit a query and receive multi-model responses this demonstration will include:

- A mock web interface (not the final interface) that will be built with React and Next.js
- The system will send the query to at least two different LLMs/SLMs using API keys
- The responses from each model will be shown on the frontend
- A low-level/rough version of the LLM-as-a-judge will score these responses based on our predefined evaluation criteria/algorithm
- The "best answer" will be determined by the system

This proof-of-concept shows that queries can be submitted through our interface, multiple models can be called in parallel with their outputs being collected, and an evaluation algorithm that can run and produce a result. This algorithm/criteria we create will not be the final version since we will still have to improve it, but it will show a rough version that will show the workflow.

15. Technology

15.1 Programming Languages and Environment

- Next.Js (Application)
  - We will be following ESLint
- Reach.js (Frontend)
- Supabase (Database) for storing API keys, queries and results
- Python (For experimenting)
  - PEP8 for formatting
- Vercel (Hosting)

15.2 Machine Learning Libraries (if relevant)

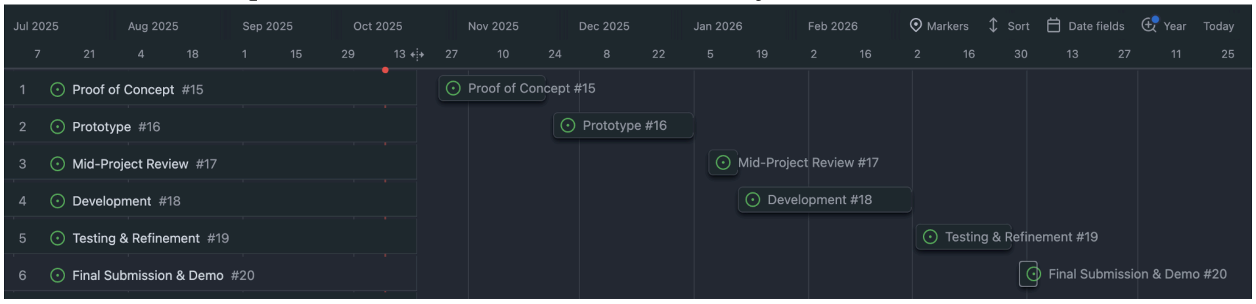
- Currently not required directly, since the system primarily calls external LLM/SLM APIs

15.3 Other Relevant Technology (e.g., GPU usage)

- AWS

16. Project Scheduling

Below is a Roadmap/Gantt Chart created in our Github Project:



Each issue has been detailed below:

Phase	Tasks	Duration	Timeline
Proof of Concept	Build query submission, multi-model response collection, mock evaluation	3 weeks	Oct 24 – Nov 21, 2025
Prototype	Implement full judging engine, response ranking, Supabase integration	6 weeks	Nov 15 – Dec 31, 2025
Mid-Project Review	Present high level working prototype to supervisor	1 week	Jan 5 – Jan 12, 2026
Development	Full-stack features (feedback, rubric management, history, APIs)	6 weeks	Jan 13 – Feb 28, 2026
Testing & Refinement	Unit/integration testing, bug fixes, performance testing	4 weeks	Mar 1 – Mar 28, 2026
Final Submission & Demo	Prepare documentation, final presentation	1 week	Mar 27 – Apr 4, 2026