Software Requirements Specification

Fall 2025 – Winter 2026 Computer Science Capstone

Our Project:

A Parallelized Cloud-Orchestrated Battery SOC Algorithm Evaluation Multitool

Intended for Algorithm *Processing*, *Transformation*, and *Comparison*
Built as a *website and progressive web app* for classroom and public sector use
Project under direct supervision of ***Phillip Kollmeyer, PhD***

Version: Version 0.1, updated October 10, 2025

Team Members and Roles

Name	Role	Email
Mohamed Elsharif	**Versioning and Maintenance Specialist**	elsharim@mcmaster.ca
Dason Wang	**Project Coordinator / Scheduling & Containerization**	wangx647@mcmaster.ca
Ellen Xiong	**Front-End Lead / User Experience**	xionge1@mcmaster.ca
Aidan McLean	**Parallelization Specialist / Performance**	mcleaa15@mcmaster.ca
Paarth Kadakia	**Back-End Lead / Reliability and Integrity**	kadakip@mcmaster.ca

Abbreviations and Notations

Term	Definition
SOC	State of Charge (of a battery)
IDE	Integrated Development Environment
SLA	Service Level Agreement
API	Application Programming Interface
CRUD	Create, Read, Update, Delete
PWA	Progressive Web Application

Table of Contents and Contributions

11-12	Non-Functional Requirements and Risks	Ellen (main)
13-16	Team Plans and Proof of Concept	Dason (main)
17-18	Technology and Scheduling	Paarth (main, including Gantt)

Purpose of the Project

Battery State of Charge (**SOC**) estimation requires specialized algorithms and standardized testing to determine which method performs best across various scenarios. A centralized, cloud-based platform for evaluating SOC estimation algorithms will accelerate research and development in battery management systems, benefiting researchers, engineers, and hobbyists. The project aims to deliver a public-facing platform that allows users to upload and benchmark SOC estimation algorithms through a scalable, parallelized cloud backend. The system will include user accounts, leaderboards, model visualization and comparison.

7.1 Primary Stakeholders

a. Client

Dr. Phil Kollmeyer – Requesting a parallelized cloud solution that allows users to create accounts, submit battery SOC estimation algorithms, test submitted algorithm performance.

b. Course Stakeholders

Dr. Mehdi Moradi, Amir Hossein Sabour

c. Dr. Kollmeyer's Teaching Assistants (TAs)

Teaching assistants supporting Dr. Kollmeyer may help evaluate submissions, test algorithms, or provide feedback on system usability and technical accuracy.

7.2 Secondary Stakeholders

a. Industry Users of the Battery SOC Estimation Tool

Individuals who use the website that hosts the SOC estimation tool to create accounts and test their submitted algorithm(s), common in the industry for product testing.

b. Students and Researchers

Students who take courses with Dr. Kollmeyer and researchers involved in battery management systems who will use the platform for testing and analysis.

c. Hobbyists and Battery Enthusiasts

Independent developers and battery hobbyists interested in submitting their own SOC estimation algorithms for comparison and testing.

8. Project Constraints

- The application will be **web-based,** and **publicly accessible.**
- Users will be able to **download the app on desktop** to run tests locally, allowing them to bypass submission limits. Two user tiers: *student* (public) and *admin*.
- The app will be designed to operate within resource and performance limits provided by **McMaster University** and cloud service platforms.
- Supervisor-approved technologies will be used, including **Python** for back-end development and **React** for front-end implementation.
- The system will adhere to **security and data privacy** best practices to ensure user submissions remain confidential.

9) Functional Requirements

Parallelization (P0)

Users need to simultaneously **submit models** (these models can range from as simple a procedure as a **one-line function**, or as complex as a high-dimensional **neural net**), and multiple cloud servers will need to evaluate those models (choose an optimal algorithm/environment to run them) to compare the efficacy.

We must guarantee a minimally acceptable **throughput** (to be later defined at the discretion of Dr. Kollmeyer, depending on future course sizes) for distributed cloud servers running several models concurrently. Servers must maintain bidirectional communication for persistent synchronization, to allow parallel computing on complex models.

- **Category:** Backend, Core System, Performance
- Requirements:
    - **Submission –** up to 100 students (estimated class size) may submit one model for SOC evaluation. The server must acknowledge each submission and determine whether to queue, distribute, or parallelize computation.
    - **Load-balancing -** container instances per server must be orchestrated to prevent resource monopolization.
    - **Throughput –** servers must periodically report computation progress and completion status to all active users. All users should see meaningful progress, otherwise their submission must be queued.
    - **Queue –** optimal synchronization method must be implemented for deferring execution of scripts and distributing workload.
    - **Algorithm Selection -** after interpreting the code, running on MATLAB or Python, an optimal evaluation algorithm must be selected (default is to run standard procedure with 100% CPU, all threads).

Model Interpretation (P0)

Users will typically submit models written in **MATLAB**, occasionally with **associated data files**. Not only is it necessary to read these models and files, but there must also be a method to **interpret MATLAB code to run as Python** on the servers.
*Optionally, as a potential, far-future add-on, it would also be beneficial to **install MATLAB directly on servers and offer native computation**. This can be separately defined as a very lesser P4 task, but for organization we'll simply include it as a side note here. **Note: All P4 tasks will be included as small mentions, as we have no true intent to implement them unless there is an unexpected excess of time.***

To evaluate user models, all of Dr. Kollmeyer's existing MATLAB algorithms will need to be refactored into equivalent Python implementations, with similar performance characteristics

Users also may submit models in Python directly.

- **Category:** Backend, Core System
- Requirements:
    - **Conversion –** MATLAB models interpreted and converted to Python in a reliable and performant fashion. Conversion time should not exceed 20% of the total evaluation time.
    - **(Advanced, extra) -** MATLAB models can run natively. Requires licensing details *(P4 Task).*
    - All the existing MATLAB algorithms → native Python implementations, similar performance (80% is satisfactory; we presume some optimizations possible).

Hosting a User Platform, with Submission and Leaderboards (P0)

The main objective of parallelization and model interpretation is to enable an **open platform for users to** intuitively **submit** their models and compare the evaluated performance to other users via **leaderboards** for each battery type and make.

Submission reading, submission interface, and Leaderboards are each defined as individual P0, P2, and P1 frontend tasks, respectively;

- **Category:** Backend, Hosting and Database
- Requirements:
    - **External Host –** A reliable, cost-effective hosting plan will need to be identified for the application which enables all other P0 functionality. Priority should be given to hosts offering affordability, scalability, and performance.
    - **Internal Host –** Project should be viable as an internally hosted application, using a McMaster ECE server or on a personal machine managed by Dr. Kollmeyer.
    - **Database –** Our chosen database, possibly bundled with hosting service, must be sufficient for all functional requirements from P0 to P2, with more details below of other tasks/requirements.

Model Submission File Reading System (P0)

Users need to be able to submit their models (these models can range from as simple a procedure as a **one-line function**, or as complex as a high-dimensional **neural net**) as a **file,**

alongside any **associated data** with their file. Models can be written in **Python**, or **MATLAB,** the industry-preference default.

As noted above, model interpretation will handle the reading for every submission. Must function both in an offline (local app) or online (distributed cloud computing) context.

- **Category:** Backend, Core System, Files
- Requirements:
    - **Reading Input Files –** fetch a file from the user, allow user to specify file format and properties
    - **(Advanced, extra) -** allow bulk upload of multiple files representing different models *(P4 task)*
    - *For the front-end version of this task, with user drag and drop, please see P2 tasks.*

Building a Website Downloadable as a Web App, Local Processing (P0)

Dr. Kollmeyer's existing interface must be updated to be accessible and downloadable as a standalone app, likely treated as a progressive web app. A core justification of this app is to enable **local processing** and **testing** on individual user machines, to save server resources and as a failsafe in events of outage.

Category: Frontend + Backend (standalone app), Core System

- Requirements:
    - **Website –** All content must be viewable from a publicly accessible website using a URL. Should be indexable by Google. App download may be directly available on the website, or through an external app store.
    - **Web App –** Website must be containerized into an app interface to run natively on Windows, Linux, and Mac. The most straightforward way to do this, and the current action plan is through a *Progressive Web App*, or similar analog. The web app variant will include all non-internet essential features (model sharing, leaderboards, comparisons) and instead focus on the single-user dashboard experience. Web App must be capable of **evaluating models using algorithms locally,** using the user's own CPU.

Security (P1)

The system requires robust and comprehensive measures in order to protect user information, internal application data, and overall system integrity. These measures include the implementation of industry-standard user authentication mechanisms (Multi-Factor Authentication, SSO, secure password policies, etc.), role-based access mechanisms (ensuring that only authenticated users can perform CRUD operations, only admin users can perform admin operations), and data encryption at rest and in transit.

To protect system integrity and resources, there will be constraints on models the user uploads (maximum model testing elapsed time, maximum file size, etc.), implementation of abuse-prevention mechanisms (rate-limiting, automatic threat & anomaly detection, including brute-force & DDOS detection), and implementation of resource preservation mechanisms (session timeout, database caching, etc.)

- **Category:** Backend, Data Integrity, User Experience
- Requirements:
    - **Authentication –** all users must authenticate via an OAuth 2.0-compliant flow. OpenID Connect will be used to retrieve identity claims, and tokens will be validated by the system before providing access.
    - **Access Control –** Only authenticated users may access protected endpoints. Fine-grained role-based access control will be enforced at the API level.
    - **Audit Logging –** All appropriate actions, such as user logins, API calls, and submissions, will be written to log files for traceability and transparency.
    - **Model Constraints -** The evaluations of uploaded models will be subjected to pre-defined limits on numerous attributes such as file size, and maximum evaluation time.
    - **Anomaly Response – U**ser activity will be monitored to detect abuse or anomalies. Upon abuse detection, measures will be in place to strip access from the offending user and discard their pending submissions if necessary.
    - **Resource Preservation** – user sessions will time out after a period of inactivity, requiring users to re-authenticate to resume use of the service. Users will be restricted by the middleware layer to a maximum number of submissions per hour. Database reads for frequently-accessed metadata will be cached in memory to reduce query load.

Bug-Fixing (P1)

Previous attempts at an interface for the SOC estimator, including (from initial inspection) various implemented algorithms in MATLAB and Django Python **are nonfunctional**. The non-working legacy code will need to be refactored into our stack, and the mistakes rectified. Currently, all essential functions are serviceable, but for the full implementation legacy code review sprints will need to be conducted.

- **Category:** Front-end, Back-end, Legacy Code
- Requirements:
    - **Bugs –** diagnose and patch bugs in the legacy code

Leaderboard Visualization (P1)

Users must be able to **compare their algorithm's performance against other users.** Includes several statistical tool packages, graphing options, and time-series with granularity up to second-by-second display.

- **Category:** Front-end, Back-end (P4 extension only)
- Requirements:
    - **Consistent updates –** Table will be filled with up-to-date information each time the page is loaded, with a refresh button available to update it manually. The leaderboard will accurately reflect the state of the database at load time.
    - **Search/Sort/Filter Functionality** – Users will be able to search, sort, and filter results quickly. Leaderboard querying will be fast, responsive, and provide up-to-date information.

Naming and setting visibility of their algorithms (P1)

When uploading a model, the user must be able to input the model's name and set its visibility to public or private. The input components must match the theme of the rest of the application, and the process must be simple and intuitive.

- **Category:** User Interface, User Experience
- Requirements:
    - **Input Components** – the input components must look consistent with the rest of the application. Name will be a text field, and visibility will be a switch that can change between "Public" or "Private".
    - **Validation** – For the name component, the input must be sanitized and validated, removing all illegal characters and ensuring a valid input before submission. Inputs will also be validated at the API level, and a default value for Visibility will be enforced, in case the value passed in is not an expected value to keep data consistent.

File Submission Interface (Drag-and-Drop Functionality) (P2)

The process for submitting user models should be intuitive, efficient, and easy to understand. The system should provide an interface that allows users to upload their models for performance testing with minimal effort. To facilitate this, a drag-and-drop interface will be implemented, enabling users to simply drag their model files into a designated drop-box for upload. After files are uploaded, users should have the option to review and verify the contents of the drop-box before final submission. This verification step ensures accuracy and prevents wasted time during performance testing, which may be lengthy for complex algorithms.

Category: Frontend, User Interface, User Experience

Requirements

- Implement drag-and-drop file upload functionality.
- Provide visual feedback during file upload (progress indicator, success/fail states).
- Allow users to review uploaded files before submission.
- Support replacement or deletion of incorrect uploads prior to final submission.

Submission History Dashboard (P2)

Each user will have access to a personalized dashboard displaying all their submitted algorithms, their current statuses, and performance results. This provides transparency and enables users to manage and re-run submissions easily.

Category: Frontend + Database Integration

Requirements:

_____- Display all models submitted by the logged-in user.
_____- Show relevant details (model name, upload date, score, runtime, visibility).
_____- Allow re-running, deleting, or downloading of previous submissions.
_____- Automatically synchronize new submissions in real time.

Result Comparison Page (P2)

Users will be able to compare multiple SOC algorithm submissions side-by-side in both visual and numerical formats. This will allow users to analyze algorithm accuracy and efficiency directly within the tool.

Category: Frontend + Data Visualization

Requirements:

- Select two or more algorithms for comparison from the leaderboard or dashboard.
- Compare performance similarities and discrepancies between models.
- Display comparison charts (accuracy, RMSE, runtime, etc.).
- Include a summary table listing key statistical metrics.
- Provide export functionality for charts (CSV, PNG). Charts to include, listed below:

    - **Graph of performance over time** – allow users to visualize SOC performance, as evaluated by an algorithm, over time
    - **Graph overlays, time series –** allow users to overlay preset, typical battery performance over their own model; this can be modified to be a preset of the optimum model on the leaderboard.
    - **Statistical overlays, time series -** allow users to run detailed statistical analysis on the performance as a time series, metric subject to recommendation by Dr. Kollmeyer.

Comments and Sharing (P3)

Users will be able to interact with one another through comments and sharing sections on models; enabling them to provide feedback, ask questions, and dialogue with each other. Users will be able to share a link to their results, post a new comment, reply to an existing comment, or upvote a comment.

- Category: User Experience
- Requirements:
    - **Submission –** each comment is associated with its poster and will display the user's alias and timestamp. The user that posted the comment will be able to delete it, and admin users will be able to delete any comment.
    - **Format –** the comment input will support basic formatting, including markdown and line breaks.

10. Performance Metrics

Note, this is NOT an ML project, so the terms will be defined differently than from lecture.

Accuracy – Defined as the % of SOC evaluations computed in (interpreted) Python matching the original MATLAB evaluations using the same algorithm and model. Since differences indicate interpreter errors, the goal is **100% accuracy** across *10 of Dr. Kollmeyer's sample models*.

Computational Complexity for Python Interpretation – Python analogs for models, algorithms, and any environment in containers must be efficient to ensure no **bottlenecking** in this step of the model submission. As an arbitrary measure, Python analogs (models, algorithms, containers) must run efficiently with no bottlenecks. Performance is acceptable if Python achieves ≥80% of MATLAB's runtime speed under identical conditions. Conversion should not exceed 20% of total evaluation time; otherwise, Python use becomes inefficient.

Parallelized Model Evaluation, under load – Performance will be validated through **q***ueue efficiency, throughput, and server load*. Success is defined by **100 simultaneous student submissions** (10 sample models of varying complexity) running *smoothly* (without interruption, every student satisfied). A stress test with Dr. Kollmeyer will confirm satisfaction and usability.

General Use – It is expected that under general use (no more than 1 user per day), the system should maintain 99.9% SLA uptime with stable backend, hosting, and database performance.

Cost – This is an especially important one that has not been directly addressed; although we do not have an absolute figure, Dr. Kollmeyer has indicated expenses should stay below **$200/month for public general use** or **$200/week** for intensive classroom testing, subject to adjustment after consultation with Dr. Kollmeyer.

11. Non-Functional Requirements

a. Look and Feel Requirements

- Must have a main landing page that introduces the battery SOC estimator tool, general information about how to use the tool, and examples of submissions. This informs visiting users of what the estimator tool is and how a user can get started
- Must display graphs that visualize the performance of the user's submitted SOC algorithm.
- Must display information about a user's submitted algorithms in an organized and informative manner, indicating details such as queue wait time, errors, and progress

b. Usability and Humanity Requirements

- Sign-up/Login is intuitive for the user, with a straightforward and understandable process.
- The algorithm submission process should be straightforward and easy to remember, with clear instructions to prevent confusion. Users should also have access to documentation explaining the submission steps and what to expect once their algorithm is processed.
- Results from the battery SOC estimation tool should be presented in a clear and organized format, with easy-to-read graphs and logically grouped data. Supporting documentation

should also be provided to help users interpret and compare results across multiple submissions.
- Error messages should be easy for users to understand, without overwhelming them with unnecessary details that could confuse the user.

c. Performance Requirements

- The system should be able to handle a large number of users accessing the website at a time, so that users won't experience any latency.
- The system should store a large number of user accounts along with their user data.
- The algorithm submission system should include safeguards to ensure that user-submitted algorithms are not too computationally intensive. To achieve this, the system should evaluate the runtime demands of each submission.
- The system should be capable of handling multiple submissions and testing processes simultaneously, supporting parallel execution across different user accounts.

d. Operational and Environmental Requirements

- Users are expected to run the application on a personal computer, either through a web browser or as a locally installed app
- The client is assumed to keep the service hosted and maintained on a cloud-based platform

e. Maintainability and Support Requirements

- Maintenance of the website platform and user accounts should be done through an admin account with the authority to change and maintain the website as needed.
- Maintenance of the server and continuous operation of the battery SOC estimation tool should be performed via the cloud-based platform under an administrator account.

f. Security and Privacy

- Account creation should require email authentication to ensure validity and prevent misuse.
- The system must ensure that all user data is kept confidential and securely protected.
- Only admins can be authorized to make changes to the platform, server, and system code.

12. Risks and Issues Predicted

Since the web system will execute multiple battery SOC estimation algorithms simultaneously through parallel processing, the size and time complexity of user-submitted algorithms could pose performance issues. Excessive or computationally intensive submissions may overload the system and potentially crash the web platform. To mitigate this, strict validation and cutoff mechanisms should be implemented to limit runtime and resource usage.

Extensive unit-testing will need to be done to verify the generalizability and performance of MATLAB -> Python interpretation, and parallelization performance in general. Underwhelming performance may lead to alternative proposals or project re-imaginings.

Additionally, an excessive number of active user accounts or simultaneous user activity could place significant strain on the system's memory and overall server health. This could result in slower performance, increased latency, or potential downtime. To maintain platform stability, proper resource allocation, user session limits, and server scaling strategies should be implemented.

Project Development Plan:

13) Team Meeting and Communication Plan

The 5-member team will use the AGILE methodology and work closely with Dr. Kollmeyer, meeting bi-weekly (except during intensive periods like midterms). All members must attend these collaborative sessions and join a follow-up huddle to review task progress.

Aside from these regular meetings, team members will communicate via **Discord**, where we have established a group chat. We will share temporary and working files (unfinished code, documents) via a shared folder in our Microsoft OneDrive, and we will commit finished code and documentation on our group's Github Repo. Collaborative coding sections are hosted via **VSCode's (**our most used IDE) **live share.**

Task management will not rely on specialized tools. Each member is expected to **track their bi-weekly assignments** and use meetings with Dr. Kollmeyer for **peer review and progress updates**. Monday.com will serve only as an external tracker for client visibility and not for internal coordination. To encapsulate these terms, our team consents to the agreement below:

Team Meeting and Communication Agreement

1. Every team member agrees to work on a *"three strikes and you're out"* model.
   For each member, we record a tally of misconducts, defined below. Each member has leeway of **3 penalties,** and upon accruing a multiple 3 penalties results in the disciplinary actions defined below:
   a. For 3 penalties, the offending team member will be asked to **compensate for the missing effort/work** that has occurred as a direct result of the penalty. Members will agree upon a suitable modified work schedule for adequate compensation.
   b. For 6 penalties, the inadequacy of the offending team member will be **reported to the instructor and TAs** by **ALL** other members.
   c. For 9 penalties and above, the team will make **all efforts to eject the offending group member from the group**, and all members will actively lobby for this with the professor and TAs.

   Any of these disciplinary actions or penalties can be waived via proof of a filed **MSAF**, **type A** or **type B**, for which the team will afford leniency based on evidenced extenuating circumstances.
2. The team will agree to communicate via **Discord,** share temporary files and working code, documents, or media on our **Microsoft Shared Drive,** and do versioning and commits for implemented code on **GitHub.**
   a. Attempts to communicate, file share, and commit changes on any platform other than these three, without explicitly informing other group members, will result in a **penalty**.
   b. Failure to interact with direct communications within a **week** of the time the communication was made, for example, ignoring multiple direct ***@mention*** in the Discord chat, will result in a **penalty**.

3. **On a bi-weekly basis,** the team will host a 30-minute **meeting on Monday at 3:00 pm, starting on Oct. 20** with Dr. Kollmeyer to present a **progress report.** After the progress report, the team (without Dr. Kollmeyer) will gather in a **huddle to assign new milestones and delegate new tasks,** which will take 30 minutes or more.
    a. **ALL MEMBERS** are required to participate in this meeting and the huddle afterwards. Members must provide a forewarning **1 hour before the meeting; failure to do so will result in a penalty** for missing this crucial meeting. Failure to *present anything at all* will result in a **penalty.**
    b. Members who present unfinished work, upon a majority vote of agreement, will suffer a **penalty** for missing work. This penalty will be applied **regardless of explanations or excuses,** with the **SOLE EXCEPTION** of a valid **MSAF.**
    c. During the huddle, each member will be assigned new tasks derived from the direct feedback obtained from Dr. Kollmeyer, supervised by the project coordinator.
4. Every member agrees to treat each other with respect and dignity in all communications. It is the right of any group member to raise violations of conduct to the instructor or the TA. Offenders, upon a majority agreement, will suffer a **penalty.**

14) Team Member Roles

The initial team member job assignments were done with the goal of completing this SRS document. The roles assigned are as follows, labelled by the SRS portions assigned:

- **Mohamed** 1-6, all: **Archivist**. This person handles the administrative details, proof-reading and grammar, making the final document up to spec, and rewriting all sections. Role later adapted into "Versioning and Maintenance Specialist"."
- **Aiden** 8-9, 10: **Requirements Writer**. This person works on the functional requirements. Will need to communicate with all other members. The 'hard' part of SRS. Most important will be the description of features, which will be shared with the Co-Requirements Writer. Role later adapted into "Parallelization Specialist."
- **Ellen** 7, 10-12: **Non-functional Requirements Writer**. This person works on user requirements. The 'soft' part of SRS. This role is later adapted into "Front-End Lead."
- **Dason** 13-16 (contributed to 9, 10): **Project Coordinator**. This person will basically do all the workflow and planning based on what Requirements Writer writes. This role remains the same but takes on parallelization responsibilities.
- **Paarth** 17-18, (contributed 8-9, 10): **Co-requirements Writer**. Complements the Requirements Writer by helping with half the stuff, focusing on technological implementation. Also does the Gantt chart. Role adapted into "Back-End Lead."

For the foreseeable future of the project, team members will take on roles aligned with their initial assignments—e.g., the Project Coordinator handling administrative tasks, or the front-end lead focusing on UX/UI. *These roles are **flexible**, not fixed, reflecting the AGILE nature of the project and the expected shifts in responsibilities throughout the software's life cycle.*

Name	Role	Suggested Task Assignments *(strengths)*
Mohamed	***Versioning and Maintenance Specialist,*** *code refactoring, including MATLAB -> Python*	*Code maintainability, merging PR, focused on* **Model Interpretation (P0);** *all tasks, also help Ellen with non-functional tasks.* *For all members, lesser P# tasks will be assigned naturally based on the corpus of knowledge and skill involved in each P0 task.*
Aiden	***Parallelization Specialist****, performance and implementation*	*Performance and Design of custom parallelized code solutions, focused on* **Parallelization (P0),** *load-balancing and throughput guarantees, optimizations*

Ellen	***Front-end Lead**, user experience*	*Lead front-end developer, guaranteeing user satisfaction and comfortable, focused on* **Building a Website Downloadable as a Web App, Local Processing (P0),** *various non-functional tasks,*
Dason	***Project Coordinator,** Parallelization containerization and scheduling*	*Chief task manager and team liaison, focused on* **Parallelization (P0),** *containerization and scheduling, model performance*
Paarth	***Back-end Lead**, reliability and integrity*	*Lead back-end developer, focused on* **Hosting a User Platform, with Submission and Leaderboards (P0)**

15) Workflow Plan

GitHub Mangement

Members are requested to keep their code up to date with the latest working main branch, with **Mohamed** tentatively responsible for major issues, PR requests, merge conflicts, and keeping track of different branches. Each member should create ***feature branches*** (P0–P1 individually; P2–P3 may be combined), named after the feature they intend to implement. Collaboration on branches is allowed. All commits and PRs must include ***clear messages.***

Pull requests should be made routinely to keep the main branch up to date, with the latest updates being integrated into the main branch and verified to be functioning (by Mohamed) before each bi-weekly meeting with Dr. Kollmeyer. ***Monday.com*** *will be updated for client visibility only*.

AGILE Methodology

Our collaborative structure is centered around the bi-weekly meetings with Dr. Kollmeyer, during which each “huddle” portion is essentially a sprint planning session with all members after freshly getting feedback from the client.

Task allocation and evaluation are collaborative, led by Dason, who oversees deliverables and records of member performance and penalties, to be kept public.

Data Storage

User data (e.g., models, rankings) will be hosted on the chosen hosting provider/database and partially on ECE servers accessible to teaching staff. Sensitive internal documents will be stored on the group’s OneDrive.
Documentation and other important files will be actively updated on the team’s GitHub.

Compute-heavy Tasks

As outlined in the very first P0 feature in our functional requirements section, computationally expensive operations will be run on servers from a chosen provider, and from ECE’s own dedicated servers or whatever personal analog Dr. Kollmeyer is willing to provide or invest in. Project operating costs were previously approved by him, tentatively.

Tools/Methods to Achieve Requirements and Performance Metrics

Parallelization: Kubernetes + Docker for container management, Parsl for parallel computing
Model Interpretation: NumPy / SciPy, MatPlotlib, *Small Matlab and Octave to Python*
Hosting a User Platform: AWS S3, Cloudfront, Api Gateway, DynamoDB
Model Submission File Reading System: Native Python
Website Downloadable as a Web App: React configuration as a progressive web app
Security: OAuth 2.0-compliant protocol, like *OpenID Connect*
Bug Fixing: Unittest + PyTest for unit testing on legacy code, also used for Model Interpretation
Leaderboard Visualization: React + Chart.js (front-end) AWS DynamoDB (backend)
Naming and Setting Visibility: React + Styled Components, AWS DynamoDB
File Submission Interface (Drag-and-Drop): React Dropzone, AWS Lambda
Result Comparison Page: React + Chart.js to visualize and compare
Comments and Sharing: React (parent and child comments), DynamoDB (storage)

16) Proof of Concept Demonstration Plan

We already, by the AGILE approach, generate many proofs of concept in the form of website/app prototypes and backend parallelization implementations ready to show Dr. Kollmeyer on a bi-weekly basis. This is a core part of our collaborative effort.

As a natural extension of this, it is our plan to repackage the latest completed demonstration for Dr. Kollmeyer and repackage it into a video proof of concept. Ideally, this will feature a mock website with functional cloud-based parallelization (as outlined in our P0 features), demonstrated live using battery models. If full parallelization isn't achieved, we'll showcase the models and algorithms that performed best. At minimum, we'll present a frontend mockup and backend code capable of SOC evaluation within about one minute for simpler models, even via command-line if necessary.

17) Technology

Front end: JavaScript will be used as our front-end programming language within the React framework. Styled-Components will be our styling library for the UI. React-router-dom will navigate between pages.
Unit Testing Framework: unittest or PyTest, heavily used for MATLAB to Python interpretation to guarantee accuracy. Container deployment and parallel computing will require additional testing measures yet to be decided upon.
Backend: We will be using an AWS (or ECE's servers, depending on if Dr. Kollmeyer is able to get resources) backend to support our project. The backend will be written in Python, including all parallelization, model/algorithm interpretation, and evaluation. There will be a Lambda and a Gateway created for the front-end to connect to the backend. We want users and admin to perform CRUD operations. Amazon Cognito will be used to authenticate users and add roles to admin users.

Containers: We will use Kubernetes + Docker
Queueing: We will likely build our own custom implementation due to the varied nature of the potential algorithm submissions. Where possible, we will use Celery and Redis for maintaining the queue, as well as task scheduling between different containers/workers/servers
Workers: We will instantiate Python workers for multiprocessing

Parallelized Stack Structure, admin details:
Regular users → CRUD on their own data
Admins → Full CRUD on all data.
Axios will be used to call for the API. Amazon DynamoDB Stores users, metadata, scores, and results. We will use Matplotlib within the AWS Lambda function to convert the MATLAB code to Python for visualization, we will use a modified version of *Small Matlab and Octave to Python's compiler/interpreter* to convert MATLAB code directly to Python.

React Frontend
↓ (POST data or file)
AWS Lambda, Kubernetes + Docker/ EC2 (Python Flask or FastAPI backend)
Distributed Clusters, in parallel
↓
NumPy / SciPy / Matplotlib
↓
Computation + Visualization
↓
Save result → S3 or return JSON to frontend

18) Gantt Chart