System Verification & Validation Report for MobiCharged

Team Super Charged (No.33)
Nashit Mohammad - mohamn31
Eric Nguyen - nguyee13
Samuel De Haan - dehaas1
Eamon Earl - earle2
Mustafa Choueib - choueibm

November 2nd, 2022

Contents

List of Figures

List of Tables

1 Revision History

Table 1: **Revision History**

Author	Date	Version	Description
All	March 8th, 2023	Rev 0	Created first draft of document
Mustafa Choueib	March 9th, 2023	Rev 1	Fixed formatting issues, revised tex document, and made grammatical corrections
Mustafa Choueib	March 15th, 2023	Rev 1	Updated Non-Functional Requirements naming convention (as changed in SRS)

2 Definitions

Table 2: **Naming Conventions and Terminology**

Word	**Definition/Context**
Functional Requirement	Requirements that describe what the product is supposed to do
Non-functional Requirement	Requirements that describe qualities that product will have
Data Smoothing	The process of using old data as well as "future" data in order to predict designs.
ML	"Machine Learning" algorithm.
SRS	Software Requirements Specification
Cryptography	The practice and study of techniques for secure communication
Asymmetric Key Cryptography	An encryption and decryption system that includes a public and private key pair
Functional Testing	Type of software testing that validates the software system against the functional requirements/specifications
Structural Testing	Type of software testing that uses the internal design of the software
Dynamic Testing	Test cases that are executed at run-time
Static Testing	Testing that does not require program execution
Manual Testing	Tests written and executed manually by team members
Automated Testing	Testing that makes use of software tools that execute tests automatically
System Testing	Testing that tests the system as a completed program and is based on the requirements
PROCESS-BENCHMARK	The average time required for completion of the current process (6 hours)
MAX-CHARGE-TIME	The maximum allowed time required to remotely charge a target device (4 hours)

3 Objective

The objective in this document is to establish a validate & verify of certain aspects of the system that correlate with the successful completion of satisfying requirements as well as ensuring the system is built as per intentions. The key objective with this document is to build confidence in the outputs produced by the Mobicharged system as well as establish confirmation of ease of navigation for our users when using the system. A selected key objective that should not be ignored is the aim to not only output application variables that will work successfully for the application, but to specifically output the most optimized solution.

4 Functional Requirements Evaluation

To ensure the system satisfies what it was intended to do, it was verified in relation to the SRS document; in particular section 9, *Functional Architecture*. In conjunction, the plan outlined in the V&V Plan document section 4, *System Test Description*, was used to guide testing. Evaluation of the functional requirements can be found in Section 5. Tests were run to determine successful completion of requirements.

- **SR1: ML Model must optimize inputs faster than the existing process**
- **SR2: ML Model must be able to develop "new" simulations based on previous optimal models**
- **SR3: ML Model must be able to encrypt optimized data before exporting**
- **SR4: The software system must determine and output optimized and correct solution**
- **SR5: ML Model must be able to process incoming simulation data from multiple source devices**
- **SR6: ML Model must be able to interpret data exported directly from MATLAB simulations**

- **HR1: The system must be able to use Phased-Wave interference to produce a visual output**

- **HR2: The system must be able to provide data to an external system**

5 Non-Functional Requirements Evaluation

In the same vein as the functional requirement evaluation, it was verified in relation to the SRS document; in particular to section 9, *Functional Architecture*. In conjunction, the plan outlined in the V&V Plan document section 4, *System Test Description*, was used to guide testing.

In certain cases where the non-functional requirements pertains to a non-quantifiable attribute, testing was not applicable (i.e, APR1). In these cases, group, supervisor, and peer feedback were taken in a casual constructive form.

- **NFR1: The hardware system will be packaged neatly such that all wiring is hidden and not exposed to others**

- **NFR2: The software system will be produced with front end design colors such that strains to the eye are minimized**

- **NFR3: The system will consist of a simple user interface by minimizing unnecessary and complex functionalities**

- **NFR4: Authorized users will have access to the system while unauthorized users will not**

- **NFR5: The system must be able to store its current state locally in the event of a failure**

- **NFR6: The individual components of the physical system must be inspected and tested**

- **NFR7: The system shall be simple to install within 10 steps and within one hour**

- **NFR8: The system shall be understandable within an hour of use**

- **NFR9: The system must compute optimal configuration within 6 hours**
- **NFR10: The hardware system must have a fail safe option such that the system shuts off at the event of a failure to reduce potential harm**
- **NFR11: The system must have a relative accuracy of 5% compared to current MATLAB simulation**
- **NFR12: The system must be available at all times**
- **NFR13: The system must be able to discard any corrupted data without adding it to the database**
- **NFR14: The hardware system must be able to withstand an input of an upper limit of 15 volts**
- **NFR15: The system must be functional on Windows and macOS**
- **NFR16: The system must encrypt all exported data**

6 Background

6.1 Software System

The purpose of the software system, MobiCharged, is a machine learning algorithm that will be used by Mobilite-Power, engineering consultant groups, general contractors, and building maintenance teams to optimize the design process required to effectively and efficiently produce the most viable remote charging system. In doing so, this will negate the current process of manually conducting simulations (that requires lengthy computerized numerical calculations), ultimately minimizing cost, manual labour, and the time necessary to produce the required results.

This system will provide users with the optimal configuration of a remote charging device based on the desired output, encrypt data protecting users when producing design results, and use data smoothing to ensure the accuracy of the system in a time efficient manner.

6.2 Hardware System

The purpose of the hardware system is to root our algorithms optimization in the real world environment. The production of a physical model will assist in determination of the absolute boundaries that can be fed into the machine learning algorithm. Variable parameter ranges will be derived from the physical model to determine the magnitude to which the boundaries can be pushed within the simulation. The physical system provides a secondary purpose in the form of data collection and verification. In order to increase the breadth of data that we can feed into the algorithm, we must determine the degree of computational error within the simulation results. A physical model will aid in determining this range and lead to further optimization through the machine learning algorithm.

7 Test Cases

7.1 Hardware Test Cases

Table 3: **Hardware Test Cases**

Test Number	**Description**	**Requirement Reference**	**Modules Referenced**	**Inputs**	**Expected Outputs**	**Actual Outputs**	**Results**
1.	Hardware visual effectiveness test	HR1	HW Power Supply, HW Microcontroller, HW Transducer Circuitry	Styrofoam ball	Levitation of object	Levitation of object	Pass
2.	Hardware visual effectiveness test	HR1	HW Power Supply, HW Microcontroller, HW Transducer Circuitry	Plastic ball	Levitation of object	No levitation of object	Fail

7.2 Software Test Cases

Component	**Test Plan Test Factors**
Server-Side Application	The main reason for testing the server-side application is to ensure that the server is launching correctly, allowing/accepting access given the connection is authorized, receiving input/output pairs from the client, communicating and transferring data to the database, generating new random input to send back to the client autonomously, and storing data to a local database that is kept until a data transfer is initiated. Aspects of the application that exhibit the same or similar functionality were grouped and tested. For example, initialization of the server was tested as a group. In addition to this, groups were tested sequentially in the order they would typically execute.
Client-Side Application	The main reason for testing the client-side application is to ensure that a client is able to calculate simulation results, connect to the server, and transmit and receive data to and from the server. Thus, the majority of testing is done to ensure correctness and reliability of the client-side application to ensure that it is operating as intended. The way the client-side application was tested is similar to the testing plan of the server-sided application. Different functions within this application were grouped and tested sequentially based on the order they would normally execute. The testing consisted mainly of testing the communication between the client and server, and ensuring the expected data being transferred and received was correct.
Server Initializer Application	The main reason for testing the server initializer is to ensure that the server will never start up with broken or incorrect configurations. This application is basically testing input types and ensuring that all the values passed in are parsed correctly and are the expected types.

Table 4: **Software Server-Client Components**

7.2.1 Server-Side Initializer Application

Test Number	**Description**	**Requirement Reference**	**Inputs**	**Expected Outputs**	**Actual Outputs**	**Results**
1.	Testing initial server configuration (# of unique inputs, # of unique outputs, range for input 1,2,3, and 4)	N/A	4,3, [1,2],[3,4], [5,6],[7,8]	inputSize == 4 && outputSize == 3 && inputList == [[1,2],[3,4],[5,6], [7,8]]	inputSize == 4 && outputSize == 3 && intputList == [[1,2],[3,4],[5,6], [7,8]]	Pass
2.	Testing that initial server configuration only accepts integer values	N/A	a,b,c,d,e,f, g,h,i,j	"Please enter an integer value" prompt	"Please enter an integer value" prompt	Pass

Table 5: Server-Side Initializer Test Cases

7.2.2 Server-Side Application

Test Number	Description	Requirement Reference	Inputs	Expected Outputs	Actual Outputs	Results
1.	Testing that the server checks local database and acknowledges it is empty	NFR5	Local database is empty	"Clean Local Database, Current Queue is empty!" prompt	"Clean Local database, Current Queue is empty!" prompt	Pass
2.	Testing that the server retrieves the data inside the local database when it is not empty	NFR5	Local Database = {{'ID': '884a5913-47a1-4610-bda4-5db03aa8f425', 'Input': [1.6998700894027414,3.688886129396 8028,1.0939982029550088 ,3.3251658927677354], 'Output': 13},{'ID':'7cf59c41-acb3-4e03-9ade-6f7235c7ca97','Input':[1.47963341984527 2,3.1059753927838574,1.0795084653703 89,3.409734962180615], 'Output':13}}	output_q == [{{'ID': '884a5913-47a1-4610-bda4-5db03aa8f425', 'Input': [1.6998700894027414,3.688886129396 8028,1.0939982029550088 ,3.3251658927677354], 'Output': 13},{'ID':'7cf59c41-acb3-4e03-9ade-6f7235c7ca97','Input':[1.47963341984527 2,3.1059753927838574,1.0795084653703 89,3.409734962180615], 'Output':13}}]	output_q == [{{'ID': '884a5913-47a1-4610-bda4-5db03aa8f425', 'Input': [1.6998700894027414,3.688886129396 8028,1.0939982029550088 ,3.3251658927677354], 'Output': 13},{'ID':'7cf59c41-acb3-4e03-9ade-6f7235c7ca97','Input':[1.47963341984527 2,3.1059753927838574,1.0795084653703 89,3.409734962180615], 'Output':13}}]	Pass

Table 6: Server-Side Test Cases (1-2)

Test Number	Description	Requirement Reference	Inputs	Expected Outputs	Actual Outputs	Results
3.	Testing that the server is appending each new input/output pair to the local database	NFR5	pair ={'ID':'7vq92b61-bdc8-1y15-9jqh-4f4367b3ba97', 'Input':[1.4631879642130127,3.567941357964125 4,1.5786135784296312,3.9634781254631236], 'Output': 10}	Local Database = {{'ID':'7vq92b61-bdc8-1y15-9jqh-4f4367b3ba97' ,'Input':[1.4631879642130127,3.5679413579641254,1.5786135784296312,3.9634781254631236], 'Output': 10}}	Local Database = {{'ID':'7vq92b61-bdc8-1y15-9jqh-4f4367b3ba97' ,'Input':[1.4631879642130127,3.5679413579641254,1.5786135784296312,3.9634781254631236], 'Output': 10}}	Pass
4.	Testing that the server can handle a client connection	NFR4	Client requests connection	connected_clients[0][0] = ('99.235.234.43')	connected_clients[0][0] = ('99.235.234.43')	Pass
5.	Testing that the server can handle multiple client connections	NFR4	Multiple clients request connection	connected_clients[0][0] = [('99.235.234.43'), ('99.235.234.43')]	connected_clients[0][0] = [('99.235.234.43'), ('99.235.234.43')]	Pass

Table 7: Server-Side Test Cases (3-5)

Test Number	Description	Requirement Reference	Inputs	Expected Outputs	Actual Outputs	Results
6.	Testing that the server can handle multiple client connections	NFR4	4 clients request connection	len(connected_clients) == 4	len(connected_clients) == 4	Pass
7.	Testing that the server rejects incorrect authorization from the client	NFR4	Client sends incorrect authorization message	authSucecss == False && "Incorrect Authorization key, disconnecting!" prompt	authSucecss == False && "Incorrect Authorization key, disconnecting!"	Pass
8.	Testing that the server accepts correct authorization from the client	NFR4	Client sends correct authorization message	authSuccess == True && connected_clients[0][0] == ('99.235.234.43') && "Correct Authorization key, Accepting Connection!" prompt	authSuccess == True && connected_clients[0][0] == ('99.235.234.43') && "Correct Authorization key, Accepting Connection!" prompt	Pass
9.	Testing that the server can receive input from the client	N/A	Client sends optimal output to server == 10.0	received_data.pickleloads(received_data) == 10	received_data.pickleloads(received_data) == 10	Pass

Table 8: Server-Side Test Cases (6-9)

Test Number	Description	Requirement Reference	Inputs	Expected Outputs	Actual Outputs	Results
10.	Testing that the server generates new input within the range specified on launch	SR2	Server receives optimal output && output_q.isFull() == False	$(1 \leq newResponse[0] \leq 2 \&\& 3 \leq newResponse[1] \leq 4 \&\& 5 \leq newResponse[2] \leq 6 \&\& 7 \leq newResponse[3] \leq 8) == True$	$(1 \leq newResponse[0] \leq 2 \&\& 3 \leq newResponse[1] \leq 4 \&\& 5 \leq newResponse[2] \leq 6 \&\& 7 \leq newResponse[3] \leq 8) == True$	Pass
11.	Testing that the server calls for a data transfer once the output_q is full	SR3	output_q.isFull() == True && received_data	Local Database. isEmpty() == True	Local Database. isEmpty() == True	Pass
12.	Testing that the server does not lose data if interrupted during data transfer	NFR13	output_q = {1,2,3,4,5,6,7, 8,9,10} && finished transferring {1,2,3,4} then interrupted during the 5th value	continue transfer from output_q == {5,6,7,8,9,10}	continue transfer from output_q == {1,2,3,4,5,6,7, 8,9,10}	Fail

Table 9: Server-Side Test Cases (10-12)

Test Number	Description	Requirement Reference	Inputs	Expected Outputs	Actual Outputs	Results
13.	Testing if the server is functional on all operating systems	NFR15	Running the server-side application on multiple operating systems	"Mobicharged is now running" prompt on all devices	"Mobicharged is now running" prompt on all devices	Pass

Table 10: Server-Side Test Cases (13)

7.2.3 Client Side Application

Test Number	Description	Requirement Reference	Inputs	Expected Outputs	Actual Outputs	Results
1.	Testing that the client-side application attempts to connect to the server	N/A	Python client _controller.py	"Please enter the authorization key: " prompt	"Please enter the authorization key: " prompt	Pass
2.	Testing that the client-side application is refused connection if failed authorization	NFR4	authorizationKey = "incorrect-password"	"Authorization Failed. Disconnecting." prompt	"Authorization Failed. Disconnecting" prompt	Pass
3.	Testing that the client-side application is accepted connection on successful authorization	NFR4	AuthorizationKey = "mobicharged"	"Authorization Successful, Connecting to server" prompt	"Authorization Successful, Connecting to server" prompt	Pass

Table 11: Client-Side Test Cases (1-3)

Test Number	Description	Requirement Reference	Inputs	Expected Outputs	Actual Outputs	Results
4.	Testing that the client-side application prompts user to input an initial value to start autonomous chain	SR2	Connection is successfully established	"Input parameter for the optimization problem:" prompt	"Input parameter for the optimization problem:" prompt	Pass
5.	Testing that the client-side application must take a float value for the initial input	SR2	initial_input = "StringInsteadOfFloat"	"Input must be numeric 0-9" prompt	"Input must be numeric 0-9" prompt	Pass
6.	Testing that the client-side application takes the input in "inputParams" and starts a MATLAB simulation	N/A	initial_input = [30.0,40.0, 50.0,60.0]	MATLAB simulation is ran given the initial_input as parameters	MATLAB simulation is ran given the initial_input as parameters	Pass

Table 12: Client-Side Test Cases (4-6)

Test Number	Description	Requirement Reference	Inputs	Expected Outputs	Actual Outputs	Results
7.	Testing that the client-side application receives and optimal value from the MATLAB simulation	N/A	initial_input = [30.0,40.0, 50.0,60.0]	MATLAB simulation is ran with initial_input as parameters && optimal_output = 10.0 (hard coded response from MATLAB for testing)	MATLAB simulation is ran with initial_input as parameters && optimal_output = 10.0 (hard coded response from MATLAB for testing)	Pass
8.	Testing that the client-side application provides a uniqueID to eery optimal output produced by MATLAB	N/A	Received optimal_output = 10.0 from MATLAB	Data["ID"] != Null	Data["ID"] != Null	Pass
9.	Testing that the client-side application sends the input/output pair to the server	N/A	data_string = pickle. dumps(data) && soc. send(data_ string)	Manual Check: received_data 1= Null (on server side)	Manual Check: received_data != Null (on server side)	Pass

Table 13: Client-Side Test Cases (7-9)

Test Number	Description	Requirement Reference	Inputs	Expected Outputs	Actual Outputs	Results
10.	Testing that the client-side application receives new input from the server	SR2	soc. send(data_string)	data_received != Null	data_received != Null	Pass
11.	Testing that the client-side application computes optimal configuration within 6 hours	NFR9	y = eng.unknown_ poly_type (inputParams, nargout = 1) && start$_stopwatch$	(Elapsed_time $\leq$ 21600 $(seconds)) == True$	(Elapsed_time $\leq$ 21600 $(seconds)) == True$	Pass

Table 14: Client-Side Test Cases (10-11)

7.2.4 Database Module Test Cases

Component	Test Plan Test Factors
Database Module	The database module requires testing to ensure that the functionality of the database module and that both the Server-Client and Machine Learning Blackboard module are able to communicate with it. Examples of the aspects being tested are the backup/recovery abilities of the database in the scenario of a major failure, and correct data formatting/indexing of all read/write functions.

Table 15: **Database Components**

Test Number	Description	Requirement Reference	Inputs	Expected Outputs	Actual Outputs	Results
1.	Test server connection. Determine if the server module is able to write to the database	NFR12	[ID:"006dd8fc-44f3-4e9a-9b15-e12557df1a48", Input:0:1,1:2,2:3, 3:4, Output:1.219222]	A new document is added to the "MATLAB_Simulations" collection	A new document is added to the "MATLAB_Simulations" collection	Pass
2.	Test client connection. Determine if the client module is able to read from the database	NFR12	Call batched_read()	Returns a dictionary containing all stored data in database	Returns a dictionary containing all stored data in database (tested with 1 entry and with 1000 entries)	Pass
3.	Test security protocol of Firestore. Ensure that non-authorized users are unable to read/write to database	NFR4	Call batched_read() without Firestore authentication	"Unauthorized request" prompt	"Unauthorized request" prompt	Pass

Table 16: Database Test Cases (1-3)

Test Number	Description	Requirement Reference	Inputs	Expected Outputs	Actual Outputs	Results
4.	Test data structure consistency during read/write functions. Ensure that the data being written to the database has the same data structure as data being read	N/A	N/A	[ID:"006dd8fc-44f3-4e9a-9b15-e12557df1a48", Input:0:1,1:2,2:3,3:4, Output:1.219 222] is written and read	[ID:"006dd8fc-44f3-4e9a-9b15-e12557df1a48", Input:0:1,1:2,2:3,3:4, Output:1.219 222] is written and read	Pass
5.	Test backup and recovery capabilities. In the case of a corrupted database, make sure that a local copy or previous version can be reestablished	NFR12	Call recover_last_db_version()	Wipes corrupted dataset and re-writes most recently stored database	Wipes corrupted dataset and re-writes most recently stored database	Pass

Table 17: Database Test Cases (4-5)

Test Number	Description	Requirement Reference	Inputs	Expected Outputs	Actual Outputs	Results
6.	Test database size functionality. Ensure that the database is still functioning after storing a large amount of data	NFR12	1. Call write_large_dataset() 2. Call batched_read()	Even with 1000 entries, the database read and write functions should still work	Both read and write functionalities continue to work. Output is not written because it is too large	Pass
7.	Test incorrect data structure handling. The database should reject incorrectly formatted data	NFR13	Call write_data() with [ID:"12"]	"Incorrect data structure inputted. Server should input [ID:"", Input:, Output: __]"	"Incorrect data structure inputted. Server should input [ID:"", Input:, Output: __]"	Pass
8.	Reading data from an empty database	N/A	Call batched_read()	"Database is empty" prompt, and returns an empty dictionary	"Database is empty" prompt, and returns an empty dictionary	Pass

Table 18: Database Test Cases (6-8)

7.2.5 Machine Learning Blackboard Test Cases

Component	**Test Plan Test Factors**
Machine Learning Blackboard	The critical behaviour that we wish to test for is the synchronicity of the threads and the interwoven processes, and that their CPU sharing is fair and allows for a reasonable user experience when combined with the UI. We also want to ensure that the main loop of the machine learner works as intended; beginning once a pre-determined threshold of data has been met in the associated database, at which point that data is funneled into the model. We then want to make sure that the user-side predictive stream is opened upon model completion, and that accuracy / progress graphs are saved in the correct locations. We also want to ensure that these graphs match the processes and final outputs of the machine learner, and that the accuracy found is representative of the accuracy displayed on these graphs. We also ultimately want to ensure that the machine learning model converges and finds a relatively accurate solution. Upon full implementation of the multi-model pruning system discussed in our demo, we will also be testing that the final model was truly the best option out of all that were attempted. ***Note:** much of this behaviour requires experiential testing, as the relative accuracies and inputs to the simulations we are attempting to mimic with our machine learning model are randomly generated, and as such can't be quantitatively verified. As such, the other behaviour, like creation of threads at the correct times, the execution of the learning process once the threshold has been met, and saving the graphs in the correct locations have also been experientially validated up to date, but these tests will be added to suite further into the development process.

Table 19: **Machine Learner Components**

Test Number	Description	Requirement Reference	Inputs	Expected Outputs	Actual Outputs	Results
1.	Upon pinging the DB and seeing a COUNT ¿ current_threshold, machine learning model is executed on current DB data points	N/A	Network response from DB	”Simulation started” prompt	”Simulation started” prompt	Pass
2.	Testing that the model terminates	N/A	DB input/output pairs	Matrix with weights of ’relative optimality’	Matrix with weights of ’relative optimality’	Pass on some functions, fails on others
3.	Testing whether the model generates an accuracy / epoch graph and saves it locally	N/A	DB input/output pairs	New local .jpg file showing the accuracy over epochs	New local .jpg file showing the accuracy over epochs	Pass

Table 20: Machine Learning Blackboard Test Cases (1-3)

Test Number	Description	Requirement Reference	Inputs	Expected Outputs	Actual Outputs	Results
4.	Testing that the predictive stream is opened on completion of the learning process	N/A	None	"Please input your desired values on which you'd like to predict:" prompt	"Please input your desired values on which you'd like to predict:" prompt	Pass
5.	Testing that upon completion of the learning process, the module starts poling the DB again	N/A	None	Current DB count	Current DB count	Pass
6.	Testing that the predictive stream call/response from the user is not stalled by parallel computations, or consistently responds in a reasonable amount of time ($\leq 5s$)	N/A	None	None	None	Fail - Stalls on startup without indication to user

Table 21: Machine Learning Blackboard Test Cases (4-6)

Test Number	**Description**	**Requirement Reference**	**Inputs**	**Expected Outputs**	**Actual Outputs**	**Results**
7.	Testing that currently optimal model weights and current_threshold are saved in the database after each successful pass through the learning process	N/A	[Weight Matrix, current_threshold]	Entries entered into DB	Entries entered into DB	Pass
8.	Testing that upon new best performing model found, this model is then used in the predictive stream	N/A	None	None	None	Fail - Yet to achieve correct signalling logic

Table 22: Machine Learning Blackboard Test Cases (7-8)

Test Number	Description	Requirement Reference	Inputs	Expected Outputs	Actual Outputs	Results
9.	Test that input/output numbers and boundaries set by the user through the initializer module are properly set in the predictive stream	N/A	[list of boundaries for each input]	None	None	Fail - values currently hard coded in file
10.	Testing that the currently sole model consistently produces accurate results that are correct within a 0.01 relative error rate	NFR11	None	None	None	Pass

Table 23: Machine Learning Blackboard Test Cases (9-10)

8 Changes Due to Testing

8.1 Server-Client Module

During testing, it was discovered that there is a major security flaw within the server-client connection regarding the IP addresses. A future change to improve security is to only allow clients that are connecting from an approved IP address list to connect.

8.2 Database Module

Previously, the Firestore database was configured to allow any user with the API key to read/write to the database. However, this was recognized to be a major security flaw and Firestore Authentication SDK was implemented to fix this. This SDK provides methods to allow users to sign in using email addresses and passwords. This user management is then used to determine which users are granted read/write permissions.

8.3 Machine Learning Blackboard

From the latency issues found in the predictive stream, and the cluttered messaging seen in the terminal, we found that proceeding with the development of a simple front-end system, to improve the user's ability to oversee the process of the learner, would be necessary to achieve a wholly useful project. This requirement was listed in our initial SRS (APR1), but its importance was eclipsed at the time by other services that required development. This could also help in thread management and notify updates to the predictive model, and allow the user to see progress without manually opening the saved graphical models.

References

We will be referring to documentations provided by Mobilite-Power, however, as of now there are no references to mention.