Project MobiCharged:
Verification and Validation

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

March 8th, 2023 (R.1)

0 Contents

1 Revision History

Date	Version	Notes
3/8/2023	1.0	-

2 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 our system as well as establish confirmation of ease of navigation for our users when using our 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.

3 Functional Requirements Evaluation

To ensure the system satisfies what it was intended to, 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.

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 the 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

Non-Functional Requirements Evaluation:

- Summarize Test conducted in relation to non-FR
- Relate back to Test #'s

APR1: The system will consist of a simple user interface

ACR1: Authorized users will have access to the system

IR1: The system must be able to store its current state locally in the event of a failure

IR2: The individual components of the physical system must be inspected and tested

EUR1: The system shall be easy to use

EUR2: The system shall be easy to install

LR1: The system shall be understandable within an hour of use

SLR1: The system must compute optimal configuration within 6 hours

PAR1: The system must have a relative accuracy of 5% compared to current Matlab simulation

RAR1: The system must be available at all times

RFTR1: The system must be able to discard any corrupted data without adding it to the database

PER1: The hardware system must be able to withstand harsh weather

ADAR1: The system must be functional on all operating systems

PVR1: The system must encrypt all exported data

Comparison to Existing Implementation:

- Note failed test/changes in design choice

4. Background

4.1 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 labor, 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.

4.2 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 the determination of the absolute boundaries that can be fed into the machine learning algorithm. Variable parameter ranges will be able to 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.

5 Test Cases

5.1 Hardware Test Cases

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

5.2 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 output 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, 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.

5.2.1 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 && inputList == [[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

5.2.2 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	**IR1**	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	**IR1**	Local Database = {{'ID': '884a5913-47a1-4610-bda4-5db03aa8f425', 'Input': [1.69987008940414, 3.68888612939680 27, 1.0939982029550088,	output_q == [{'ID': '884a5913-47a1-4610-bda4-5db03aa8f425', 'Input': [1.69987008940 27414, 3.68888612939 68027, 1.09399820295 50088, 3.32516589276 77354], 'Output': 13}, {'ID': '7cf59c41-acb3-4e03-9ade-6f72 35c7ca97', 'Input': [1.47963341984 5272, 3.10597539278 38574,	output_q == [{'ID': '884a5913-47a1-4610-bda4-5db03aa8f425', 'Input': [1.699870089402741 4, 3.6888861293968027 , 1.0939982029550088 , 3.3251658927677354], 'Output': 13}, {'ID': '7cf59c41-acb3-4e03-9ade-6f7235c7ca97', 'Input': [1.479633419845272 , 3.1059753927838574 , 1.079508465370389, 3.409734962180615] , 'Output': 13}]	Pass

			3.325165892767735], 'Output': 13}, {'ID': '7cf59c41-acb3-4e03-9ade-6f7235c7ca97', 'Input': [1.47963341984572, 3.105975392783857, 1.079508465370389, 3.409734962180615], 'Output': 13}}	1.079508465370389, 3.409734962180615], 'Output': 13}]		
3.	Testing that the server is appending each	**IR1**	Input/ output pair = {'ID': '7vq92	Local Database = {{'ID': '7vq92b61-bdc8-1y15-9jqh-4f4367b3ba97',	Local Database = {{'ID': '7vq92b61-bdc8-1y15-9jqh-4f4367b3ba97', 'Input':	Pass

	new input/output pair to the local database		b61-bdc8-1y15-9jqh-4f4367b3ba97', 'Input': [1.46318796421301 27, 3.5679413579641254, 1.5786135784296312, 3.9634781254631236], 'Output': 10}}	'Input': [1.4631879642130127, 3.5679413579641254, 1.5786135784296312, 3.9634781254631236], 'Output': 10}}	[1.4631879642130127, 3.5679413579641254, 1.5786135784296312, 3.9634781254631236], 'Output': 10}}	
4.	Testing that the server can handle a client-connection	**ACR1**	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	**ACR1**	Multiple clients request	Connected_clients[0] = [('99.235.234.43'), ('99.235.234.43')]	Connected_clients[0] = [('99.235.234.43'), ('99.235.234.43')]	Pass

	client connections		connection			
6.	Testing that the server can handle multiple client connections	**ACR1**	4 clients request connection	Len(connected_clients) == 4	Len(connected_clients) == 4	Pass
7.	Testing that the server rejects incorrect authorization from the client	**ACR1**	Client sends incorrect authorization message	authSuccess == False && "Incorrect Authorization key, disconnecting" prompt	authSuccess == False && "Incorrect Authorization key, disconnecting" prompt	Pass
8.	Testing that the server accepts correct authorization from the client	**ACR1**	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	**N/A**	Client sends optimal output to	Received_data.pickleloads(received_data) == 10	Received_data.pickleloads(received_data) == 10	Pass

	from the client		server == 10			
10.	Testing that the server generates new input within the range specified on launch	**SR2**	Server receives optimal output && output_q.isFull() == False	(1 <= newResponse[0] <= 2 && 3 <= newResponse[1] <= 4 && 5 <= newResponse[2] <= 6 && 7 <= newResposne[3] <= 8) == True	(1 <= newResponse[0] <= 2 && 3 <= newResponse[1] <= 4 && 5 <= newResponse[2] <= 6 && 7 <= newResposne[3] <= 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	**RFTR1**	Output_q = {1,2,3,4,5,6,7,8,9,10} && finished transferring {1,2,3,4} then interrupted during	Continue transfer from output_q == {5,6,7,8,9,10}	Continue transfer from ouput_q == {1,2,3,4,5,6,7,8,9,10}	Fail: This test failed as the local database is cleared once all values are trans

			5th value			ferred, thus, it restored all values
13.	Testing if the server is functional on all operating systems	**ADAR1**	Running the server-side application on multiple operating systems	"Mobicharged is now running" prompt	"Mobicharged is now running prompt"	Passs

5.2.3 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	**ACR 1**	AuthorizationKey = "incorrectpassword"	"Authorization Failed. Disconnecting." Prompt	"Authorization Failed. Disconnecting." Prompt	Pass
3.	Testing that the client-side application is accepted connection on successful authorization	**ACR 1**	AuthorizationKey = "mobicharged"	"Authorization Successful, connecting to server" prompt	"Authorization Successful, connecting to server" prompt	Pass
4.	Testing that the client-side	**SR2**	Connection is successfull	"Input parameter for the	"Input parameter for the optimization problem:" prompt	Pass

	application prompts user to input an initial value to start autonomous chain		y established.	optimization problem:" prompt		
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
7.	Testing that the client-side application receives an optimal value from the	**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 simulation is ran with initial_input as parameters && optimal_output = 10.0 (hard coded response from	Pass

	MATLAB simulation			MATLAB for testing)	MATLAB for testing)	
8.	Testing that the client-side application provides a unique ID to every 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 != Null (on server side)	Manual Check: Received_data != Null (on server side)	Pass
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	**SLR 1**	Y = eng.unkown_poly_type(inputParams, nargout = 1)	(Elapsed_time <= 21600 (seconds)) == True	(Elapsed_time <= 21600 (seconds)) == True	Pass

	configuration within 6 hours		&& start_stopwatch			

5.2.4 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 back-up/recovery abilities of the database in the scenario of a major failure, and correct data formatting/indexing of all read/write functions.

Database Module Test Suite:

Test Case	Description	Requirement	Inputs	Expected Outputs	Actual Output	Result
1.	Test server connection. Determine if the server module able to write to the database.	RAR1	[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_Simulati-ons" collection.	A new document is added to the "MATLAB_Simulati-ons" collection.	Pass

2	Test client connection. Determine if the client module able to read from the database.	RAR1	Call batched_read()	Returns a dictionary containing all stored data in database.	Returns a dictionary containing all stored data in database. Tested with only 1 data entry, and with 1000.	Pass
5	Test security protocol of Firestore. Ensure that non-authorized users are unable to read/write to the database.	ACR1	Call batched_read() without Firesto-reauth-enthication	"Unauthorized request." Error should be returned.	"Unauthorized request."	Pass
6	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		[ID:" 006dd8fc-44f3-4e9a-9b15-e12557df1a48", Input: {0:1,1:2,2:3,3:4}, Output: 1.219222] is written and read	[ID:" 006dd8fc-44f3-4e9a-9b15-e12557df1a48", Input: {0:1,1:2,2:3,3:4}, Output: 1.219222] The input and output data have the same data structure.	
7	Test backup and recovery capabilities. In the case of a corrupted database, make sure that a local	RAR1	1. 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

	copy or previous version can be re-established.					
8	Test database size functionality. Ensure that the database is still functioning after storing a large amount of data.	RAR1 PER1	1.Call write_ large_ dataset () 2. call batche d_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
9	Test incorrect data structure handling. The database should reject incorrectly formatted data.	RFTR1	1.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
10	Reading data from an empty database	N/A	Call batche d_read ()	“Database is empty” message, and returns an empty dictionary	“Database is empty” message, and returns an empty dictionary	Pass

5.2.5 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 behavior, like the 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 a suite further into the development process.

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”	“Simulation started”	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, fail 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

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: ”	“Please input your desired values on which you’d like to predict: ”	Pass
5.	Testing that upon completion of the learning process, the module starts polling 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	N / A	None	None	None	Fail – stalls on startup without indication to user

	consistently responds in a reasonable amount of time (< 5s)					
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

9.	Test that input / output #s 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	PAR1	None	None	None	Pass

6 Changes Due to Testing:

6.1 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.

6.2 Database Module

Previously, the Firestore database was configured to allow any user with the API key to read and write 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.

6.3 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 frontend system, to improve the user's ability to oversee the process of the learner, would be necessary to achieve a wholly useful product. 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.