

Consonant Keyboard: Final Report

Mark Lavrentyev

Jed Fox

Max Heller

Kris Diallo

December 14, 2022

1 Project overview

The consonant keyboard is a QWERTY-layout keyboard with all non-consonant keys (vowels, numbers, symbols, etc.) omitted that can be attached via USB to any client device that supports ordinary USB keyboards. As users type consonants, the keyboard attempts to determine the user's intended words by making requests to GPT-3, a textual machine learning model. These possible completions are displayed on an LCD and the user can switch between the options using buttons on the side of the keyboard. Once they've found the word they had in mind (or a decent alternative), the user accepts the suggestion using an enter-like button. While largely a novelty item, this could be useful in speeding up the typing speed of users who get enough practice with it.

2 Requirements

Note: 'current word' is defined as the characters following the last non-trailing space character e.g. `world` in `hello_world_` (spaces represented as `_` for emphasis).

R1a If a key is pressed, its associated consonant shall be sent to the client.

R1b If a key is held down, it shall be interpreted as a single press of the key.

R1c Multiple keys pressed at the same time shall be interpreted as a single press of one of the pressed keys.

R2a If 100ms has passed since the last keystroke, the device shall send the current word to the GPT-3 model and wait for a response.

R2b If the GPT-3 API responds within four seconds, the list of suggested completions produced by GPT-3 shall be displayed on the LCD.

R2c If the GPT-3 API fails to respond within four seconds, the device shall write an error message to Serial and reset itself.

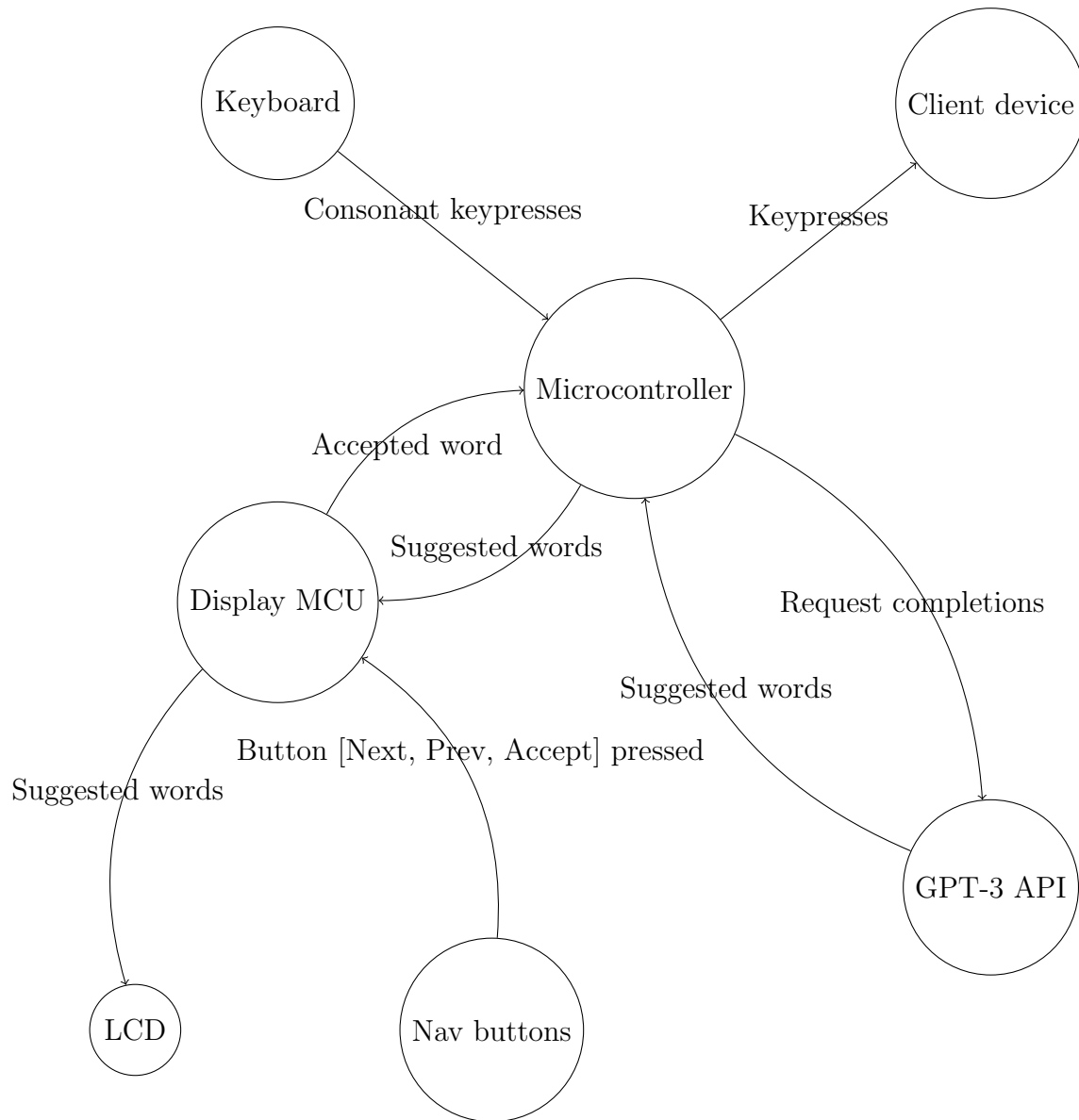
R3a If the LCD is displaying completions, it shall visibly highlight one 'focused' word.

R3b If the LCD is displaying completions and the Next button is pressed, it shall focus on the next completion, wrapping around at the end of the list.

R3c If the LCD is displaying completions and the Prev button is pressed, it shall focus on the previous completion if the currently focused completion is not the first in the list.

R3d If the LCD is displaying completions and the Accept button is pressed, the current word shall be replaced with the focused word on the client.

3 Architecture

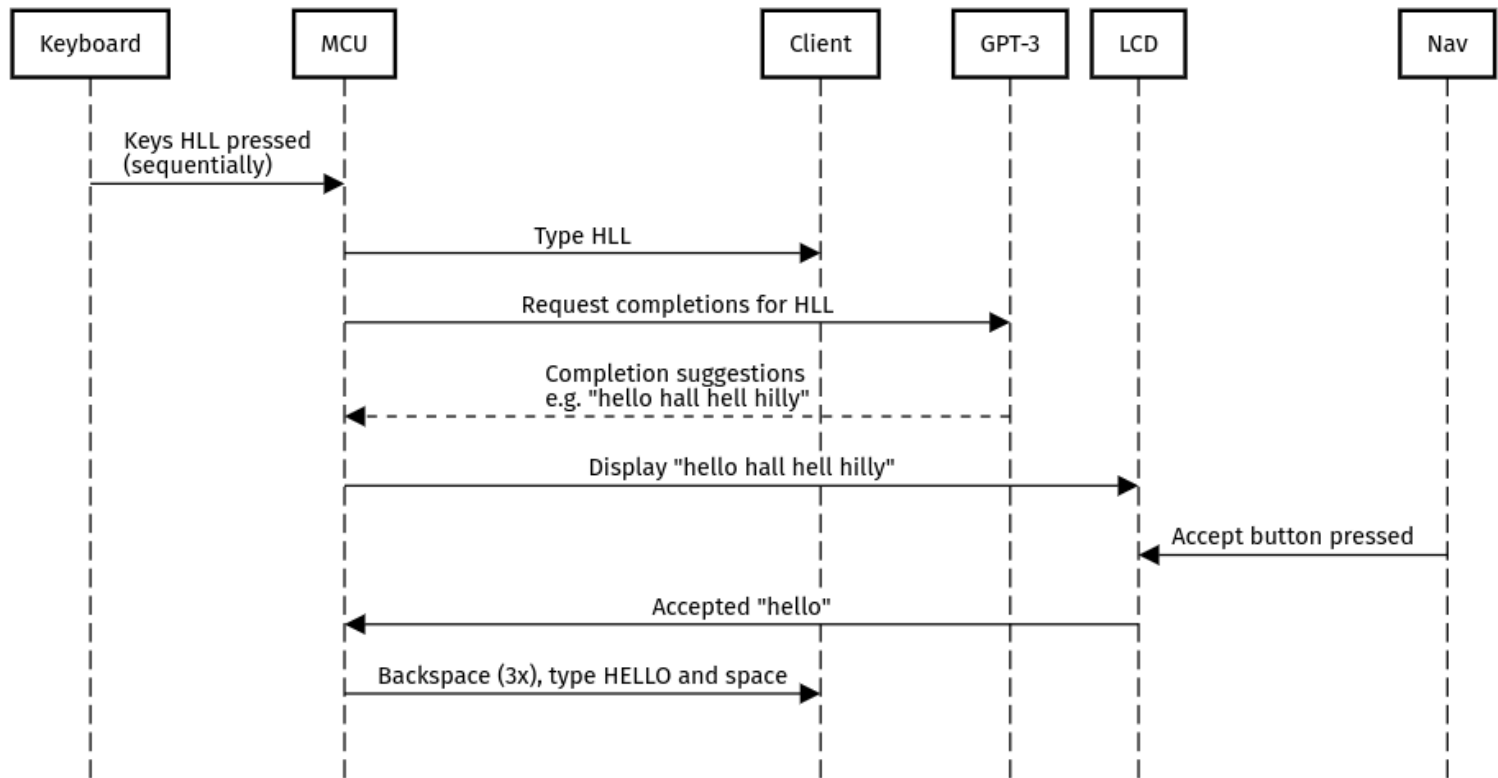


4 Use case scenarios

Because our system is a keyboard, it can be used in infinitely many ways (a user can type any arbitrary sequence of consonants, arbitrarily switch between and accept suggestions, etc.). We depict representative use cases of typing a single word, switching between completion suggestions, and typing multiple words – these can be composed to type any sequence of words (assuming GPT-3 cooperates in suggesting the words the user had in mind).

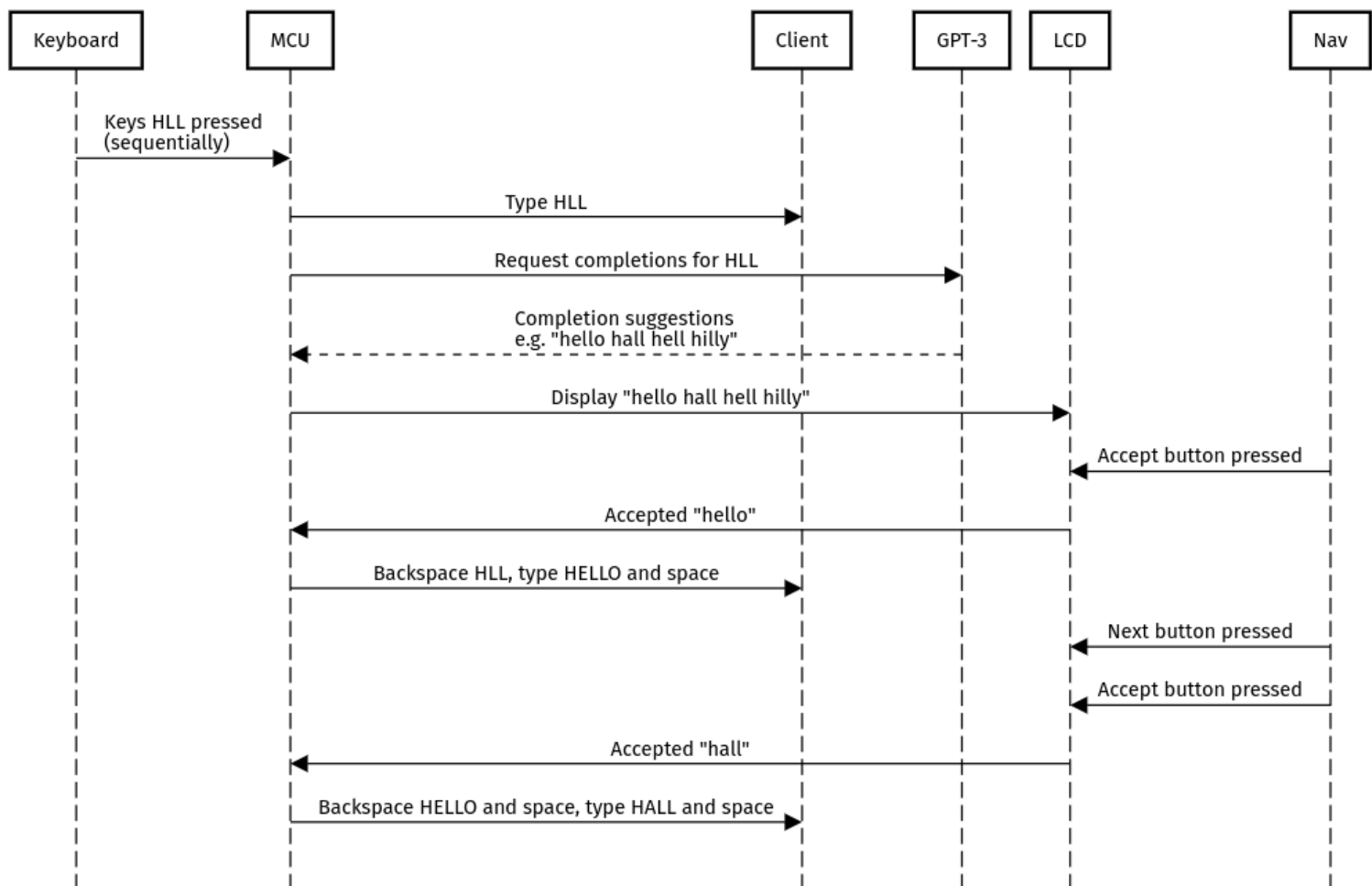
This diagram depicts the steps required to type "hello" on the keyboard: the user types in the word's consonants (HLL), which are typed out on the client device, the system requests suggestions from GPT-3, which are displayed on the LCD, the user selects the appropriate word using the nav buttons, and "hello" is typed out on the client device, replacing "hll".

Typing "hello"

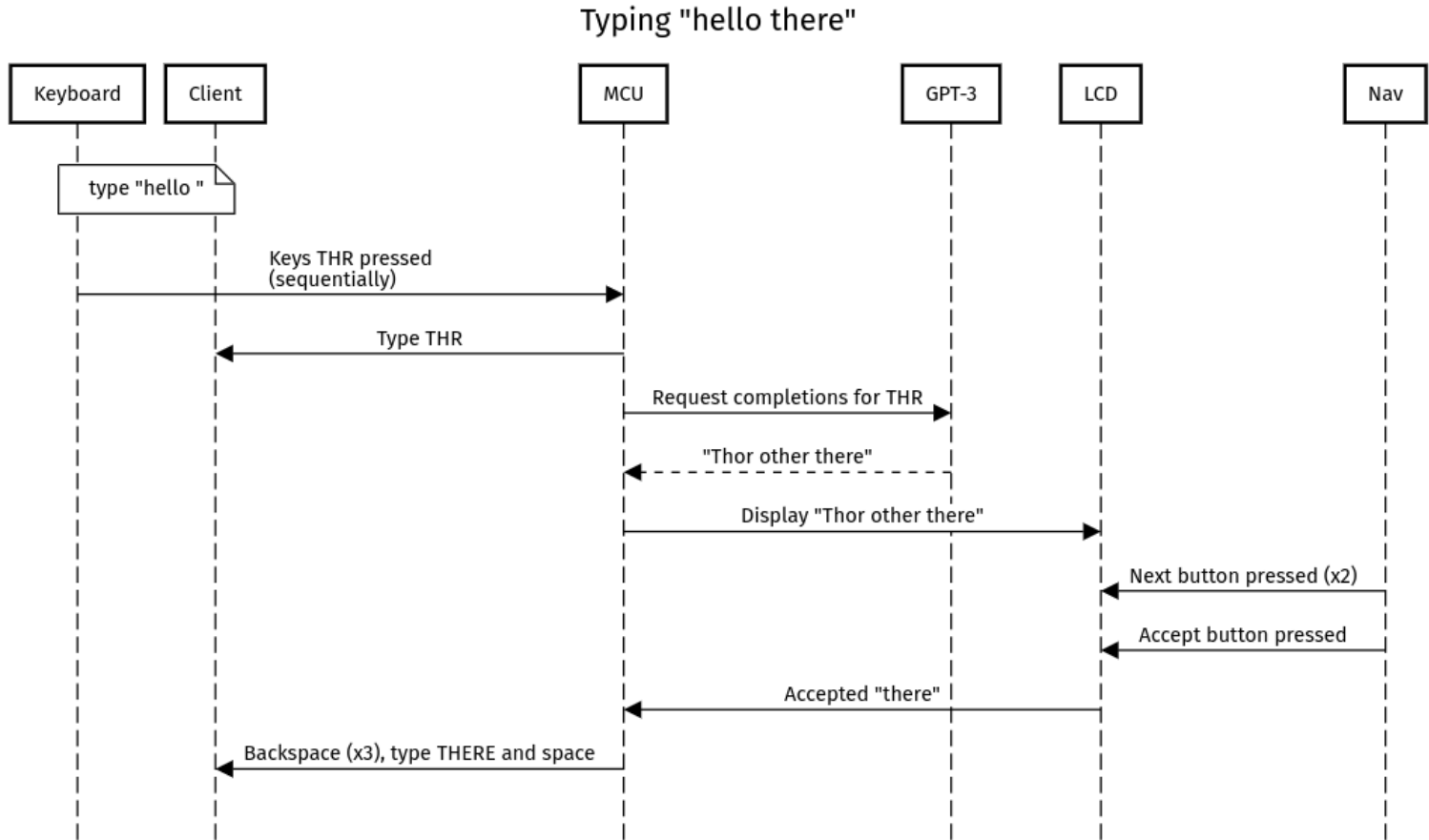


This diagram depicts the user selecting the first suggestion then changing their mind and switching to a different suggestion. The same steps are executed as before, but after "hello" is typed, the user presses the next button and the accept button, replacing "hello" with the second suggestion, "hall".

Switching between suggestions



This diagram depicts the user typing multiple words. First, they type "hello " as in the first sequence diagram. Then, they repeat the process of typing consonants and accepting a suggestion to additionally type "there ", resulting in a final output of "hello there ".

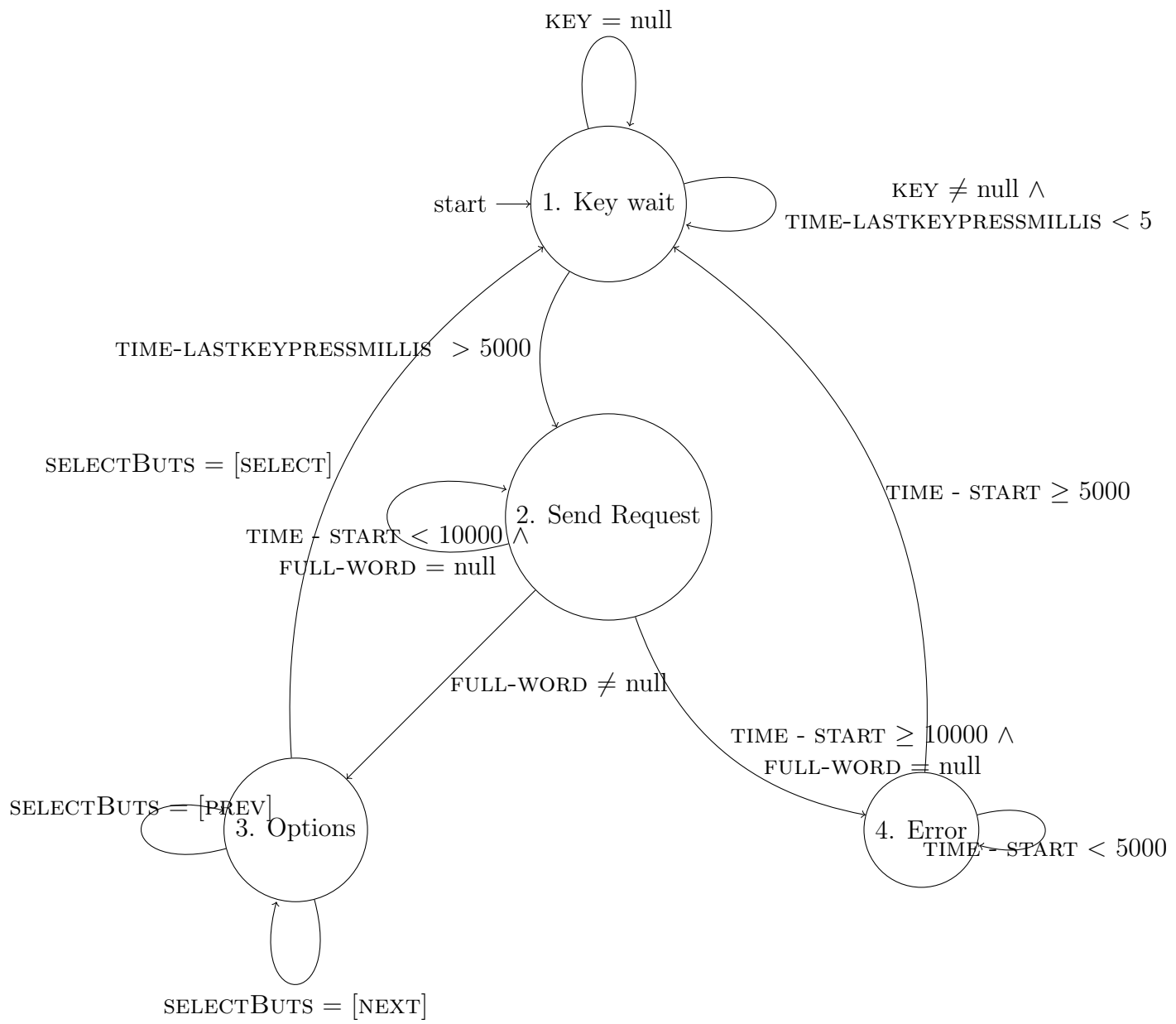


5 Finite state machine model

Our finite state machine will be hybrid time- and event-driven. It needs as input the current time (TIME) and the currently pressed letter key (KEY, equal to “null” if nothing is pressed), as well as the set of selection buttons pressed SELECTBUTS. In state 1, it will be run whenever the KEY input is changed. In state 2, it will be run every 10ms. It will produce as outputs the the full word produced and

It has the following variables:

- BUF: the current letters the user has typed in, but not yet sent for vowel insertion.
- START: time since the state started (for states 2 and 3).
- FULLWORD: the full word options, with vowels inserted.
- LASTKEYPRESSMILLIS: time since last key was pressed
- SELECTEDOPTION: which option is currently selected



Transition	Guard	Explanation	Actions
1-1a	$\text{len}(\text{KEYS}) = 0$	No keys pressed.	-
1-1b	$\text{len}(\text{KEYS}) > 0 \wedge \text{SPACE} \notin \text{KEYS}$	Some letter keys were pressed.	$\text{BUF} += \text{KEYS}$ $\text{BUF} += \text{KEYS} - "$ "; $\text{recreate_word}(\text{BUF});$ $\text{BUF} = "";$
1-2	$\text{TIME} - \text{LASTKEYPRESSMILLIS} \geq 5000$	The word is sent for completion.	$\text{START} = \text{current_time}();$ $\text{FULL-WORDS} = \text{get_recreated_words}();$ $\text{SELECTEDOPTION} = 0;$ $\text{display}(\text{SELECTEDOPTION}, \text{FULLWORD});$
2-2	$\text{TIME} - \text{START} < 10 \wedge \text{FULL-WORD} = \text{NULL}$	Waiting for recreated word.	
2-3	$\text{FULL-WORDS} \neq \text{null}$	GPT3 returned options for completions	
2-4	$\text{TIME} - \text{START} \geq 10 \wedge \text{FULL-WORDS} = \text{NULL}$	Request to recreate word timed out.	$\text{START} = \text{current_time}();$
3-1	$\text{SELECTBUTS} = [\text{SELECT}]$	Select the current option and type it out.	$\text{type_word}(\text{FULLWORD}[\text{SELECTEDOPTION}]);$
3-3a	$\text{SELECTBUTS} = [\text{NEXT}]$	Scroll to next completion option.	$\text{SELECTEDOPTION} = (\text{SELECTEDOPTION} + 1) \% \text{len}(\text{FULLWORD});$ $\text{display_opts}(\text{SELECTEDOPTION}, \text{FULLWORD})$
3-3b	$\text{SELECTBUTS} = [\text{PREV}]$	Scroll to previous completion option	$\text{SELECTEDOPTION} = (\text{SELECTEDOPTION} - 1) \% \text{len}(\text{FULLWORD});$ $\text{display_opts}(\text{SELECTEDOPTION}, \text{FULLWORD})$
4-1	$\text{TIME} - \text{START} \geq 5$	Reset the system.	$\text{BUF} = "";$ $\text{FULL-WORD} = "";$ $\text{display_error}();$
4-4	$\text{time} - \text{start} < 5$	Displaying error message.	

6 Traceability

	R1a	R1b	R1c	R2a	R2b	R2c	R3a	R3b	R3c	R3d
1-1a	X									
1-1b	X	X	X							
1-2				X						
2-2				X						
2-3					X					
2-4						X				
3-1					X					X
3-3a							X	X		
3-3b							X		X	
4-1						X				
4-4						X				

7 Testing

There are essentially two main subsystems that are interacting in this project: the keyboard (and reading user keystrokes into a buffer), and the GPT-3-based word recreater. To unit test the keyboard, we will test the following scenarios:

- Test that no action occurs when no keys are pressed (via passing empty KEYS input).
- Test individual key pressed (passed into FSM KEYS) for each key and verify that the right letter is added to the buffer.
- Test that multiple simultaneous key presses (without space bar) add one of the two letters to the buffer. Verify that this works for any length of KEYS, from 2 to 20.
- Test that receiving a recreated word types it out and returns the FSM back to state 1.
- Test that if the GPT-3 request times out, an error is displayed (advances to state 3).
- Test that before 5 seconds, the error continues to display (transition 3-3). After 5 seconds, the system is reset (3-1).

For the GPT-3 subsystem, we will implement the following unit tests:

- Test that the correct prompt is built given a consonant-only word.
- Test that a GPT-3 API request is sent with the built prompt.
- Test that the completion word is correctly extracted from the API response.

For end-to-end system tests, we will test using a user typing normally. We expect that the user will be able to type with consonants only, and have the word completed when pressing the space bar.

8 Safety and liveness requirements

Below, being in state n is abbreviated as Sn .

8.1 Safety requirements

- $G(S1 \wedge \text{key} = \text{null} \implies X(S1))$
- $G(F(S3 \wedge \text{selectButs} = [\text{SELECT}] \implies X(S1))$
- $G(\text{FULLWORD} \in [\text{a}..z\text{A}..Z])$

8.2 Liveness requirements

- $G(\text{len}(\text{keys}) = 1 \wedge \text{space} \notin \text{keys} \implies X(\text{buf}) = \text{buf} + \text{keys}[0])$
- $G(\text{len}(\text{fullWord}) > 1 \wedge \text{len}(\text{selectButs}) > 0 \wedge \text{selectButs} \neq [\text{SELECT}] \implies X(\text{selectedOption}) \neq \text{selectedOption})$
- $G(\text{keys} \neq \emptyset \implies F(\text{fullWord} \neq \emptyset))$
- $\forall k, G(\text{keys} = [k] \wedge \text{space} \neq k \implies F(\text{type_word}(\dots + k + \dots)))$

9 Closed system analysis

To implement a closed-system analysis of our consonant keyboard, we would need to model the following environmental processes:

- The internet connectivity status of the Arduino, a *discrete* model *non-deterministically* outputting connected or disconnected.
- The GPT-3 API that our consonant keyboard interacts with, which consumes a prompt and outputs a list of completions or an error. Since completions are non-deterministic (we use a non-zero ‘temperature’ parameter in our calls), the model of this system would need to be *non-deterministic*. It can, however, be *discrete*, since the output is capped at a certain length, so the response of the system could be modeled as returning a fixed-size array of characters.
- Keypresses from the user. Such a model would be *non-deterministic* – since keypresses come in at arbitrary times and for arbitrary consonants – and *discrete*, since there is a fixed set of 20 character keys that can either be pressed or not pressed at any point.

Composing these environmental process models with the model of the keyboard system itself, we could do reachability analysis to confirm that the error state is only ever reached in the correct scenarios (e.g. when there is no internet connection present or the GPT-3 API is down).

10 Code Deliverables

DISPLAY : Arduino project for the display MCU that handles displaying suggested completions on the LCD, navigating between suggestions, and sending accepted suggestions to the keyboard MCU.

- **DISPLAY.INO** : Responsible for controlling the LCD display. It contains logic that bring functionality to the three black button, which allow you to cycle through and select from the list of completed words that is displayed.

KEYBOARD : Arduino project for the main (keyboard) MCU that handles processing keystrokes, communicating with GPT-3, and transmitting keystrokes to the client.

- **ENV.INO** : Configuration variables for connecting to WiFi (SSID, password) and communicating with the proxy (server hostname and an access token required by the proxy to ensure authorized access).
- **GPT.INO** : Functionality for configuring WiFi and making requests to/parsing responses from GPT-3 through the proxy. This file satisfies the WiFi requirement (lines 13/15 connect to WiFi and line 34 makes a socket connection).
- **KEYBOARD.INO** : Controls the user-interactions and processing of the physical keyboard. It contains the character buffer and the mapping of keys to rows and columns.

It also contains the logic for the LED which updates the User on the state of device. The LED is controlled via PWM in the `ledIndicatorIdle()` method on line 44.

Keystrokes are processed with ISRs attached to each column of keys, configured in `initializeColumnPins()` on line 75.

- **KEYBOARDTESTS.INO** : Tests the functionality of the keyboard. Calls on helper methods and uses assertions to ensure the keyboard is tested thoroughly.

- WATCHDOG.INO : Enables the watchdog timer that triggers when a response to a request to GPT-3 has not been received within 4 seconds. This file fulfills the watchdog timer requirement.

PROXY : A proxy server that sits between the MCU and the GPT-3 API so that prompt formatting and HTTPS communication with the API can be done in JS, which is easier than in C.

- PACKAGE.JSON : Defines the Javascript package implementing the proxy.
- SERVER.JS : Implementation of the proxy server. Listens for HTTP requests containing partial words and sends prompts requesting suggested completions for partial words to GPT-3. The prompt contains instructions of what GPT-3 should do along with example input/output pairs to guide its answers. Also keeps track of how much cost we've incurred with our use of GPT-3.

11 Running tests

You can run our test suite by simply uploading the keyboard code to any Arduino. It will run all tests before beginning the interactive user-session.

12 Reflection

We met most of the goals that we set out in the original project proposal, with a few modifications:

- Instead of our original keyboard design where each switch received power and was connected to its row and column with two diodes, we shifted to powering each row and connecting each switch to its column with one diode. With this new design, power flows through the switch from the row to the column, triggering an ISR, and we detect which key was pressed by powering one row at a time until a column receives a signal – this row/column pair corresponds to the keypress. This is slightly more complicated in software, but significantly reduced the amount of wiring and soldering we needed.
- Instead of our original 4x5 grid, we transitioned to a 3x7 grid because the layout ended up making more sense.

We faced several challenges:

- As anticipated in the proposal, connecting the easy button turned out to be not easy. Although we were able to identify a contact on the PCB that carried a signal when the button was pressed, and were able to hook the easy button up to the Arduino's VCC and ground, we were ultimately unable to detect button presses for unknown reasons. Attempting to hook the button up was also a time sink because the PCB was a material not conducive to holding solder.
- Getting HTTP requests working with the Arduino WiFi library was difficult to debug. We went on a wild goose chase trying to debug our request-generating code only to realize that it was hanging because interrupts were disabled.
- Getting GPT-3 to produce even remotely sensible suggestions, especially when we added support for multiple suggestions, was tricky. Even after a lot of tweaking of the prompt, it still produces some headscratchers like words that don't contain all of the typed consonants.

13 Appendix

Project name:	Consonant Keyboard
Defect record	
Transition or state #	Defect
State 1	(example -- delete this and replace with your own) Seems like it should be the initial state, but there is no initialization arrow
1-2	Could be more descriptive about transition on FSM diagram regarding how space is a member/one of the keys which is pressed as diagram currently just says "space"
2-2	May potentially want to add a check within the transition guard which ensures that full word is only not equal to null only for when the user is entering in the first consonant as this condition within the guard will never be true after the user has entered their first desired consonant but we want to be able to remain in this state until the user has entered all of their desired consonants.
2-3	I would recommend potentially adding an "end" state where there is a clear indication for a user that they are able to enter in a new word other than the current 3-1 transition which simply resets based on time but doesn't take into account a user's potential action where they want to type in different words in a short time span.
3-1	For the transition between these states, you could potentially add a variable that's a boolean which lets you know that you are in the "error" state in order to make the transition from the error to start states more robust and ensure that the guard is mutually exclusive.

Fixed

Fixed

Fixed

Fixed

Fixed

Project name:	Consonant Keyboard
Defect record	
Transition or state #	Defect
State 3	FULL-WORD labelled as NULL in FSM but "" in explanation - which is it's start value?
transition 1-2	Guard is "SPACE is in keys" but FSM only says "SPACE" on the guard

Fixed
Fixed

Project name:	
Defect record	
Transition or state #	Defect
3-1	If there's an error after you write a sentence or sequence of words, it will erase the entire word buffer. Thus, if there's any error, all words will be deleted. Unless the buffer/FULL-WORD stores only one word at a time?
	Everything else looks pretty good to me!
Y	Start state is indicated
Y	All states are numbered with a unique number
Y	All states have a unique, short, descriptive name
N	Inputs, outputs, and variables are defined
N	Input and variable initialization is included
Y	All transitions out of a state are mutually exclusive
Y	Only inputs and variables are checked in guard conditions
Y	Only outputs and variables are set in actions
Y	FSM behavior aligns with behavior/requirements put forth in team presentation

Fixed
Fixed

Fixed

Project name:	
Defect record	
Transition or state #	Defect
All	No state transition appears to correspond to RO2. It's also not immediately clear what any of the ROs are in this case.

Fixed

Y	Start state is indicated
Y	All states are numbered with a unique number
Y	All states have a unique, short, descriptive name
Y	Inputs, outputs, and variables are defined
Y	Input and variable initialization is included
Y	All transitions out of a state are mutually exclusive
Y	Only inputs and variables are checked in guard conditions
Y	Only outputs and variables are set in actions
Y	FSM behavior aligns with behavior/requirements put forth in team presentation

Project name:	Consonant Keyboard
Defect record	
Transition or state #	Defect
Transition 1-2	empty word can be inputted which would trigger system which may not be intended
State 2	Full word seems like it should be reset to null if we transition to state 1 after state 2

Fixed

Fixed

Fixed

Fixed

<input checked="" type="checkbox"/>	Start state is indicated
<input checked="" type="checkbox"/>	All states are numbered with a unique number
<input checked="" type="checkbox"/>	All states have a unique, short, descriptive name
<input type="checkbox"/>	Inputs, outputs, and variables are defined
<input type="checkbox"/>	Input and variable initialization is included
<input checked="" type="checkbox"/>	All transitions out of a state are mutually exclusive
<input checked="" type="checkbox"/>	Only inputs and variables are checked in guard conditions
<input checked="" type="checkbox"/>	Only outputs and variables are set in actions
<input checked="" type="checkbox"/>	FSM behavior aligns with behavior/requirements put forth in team presentation