Consonant Keyboard: Final Report

Mark Lavrentyev Jed Fox Max Heller Kris Diallo

December 17, 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.

We assume that our users wish to communicate in English, are able to read printed legends on keycaps (or are capable of touch-typing on QWERTY with homing bumps on F and J), are able to read words displayed on the included LCD display at the standard size, are able to press low-resistance keyswitches and buttons, and do not type faster than one keystroke per 5ms (≈ 2400wpm assuming an average word length of 5 characters).

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 Iff 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. The interpreted letter shall come from the set of pressed keys, but may be any one of these 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.

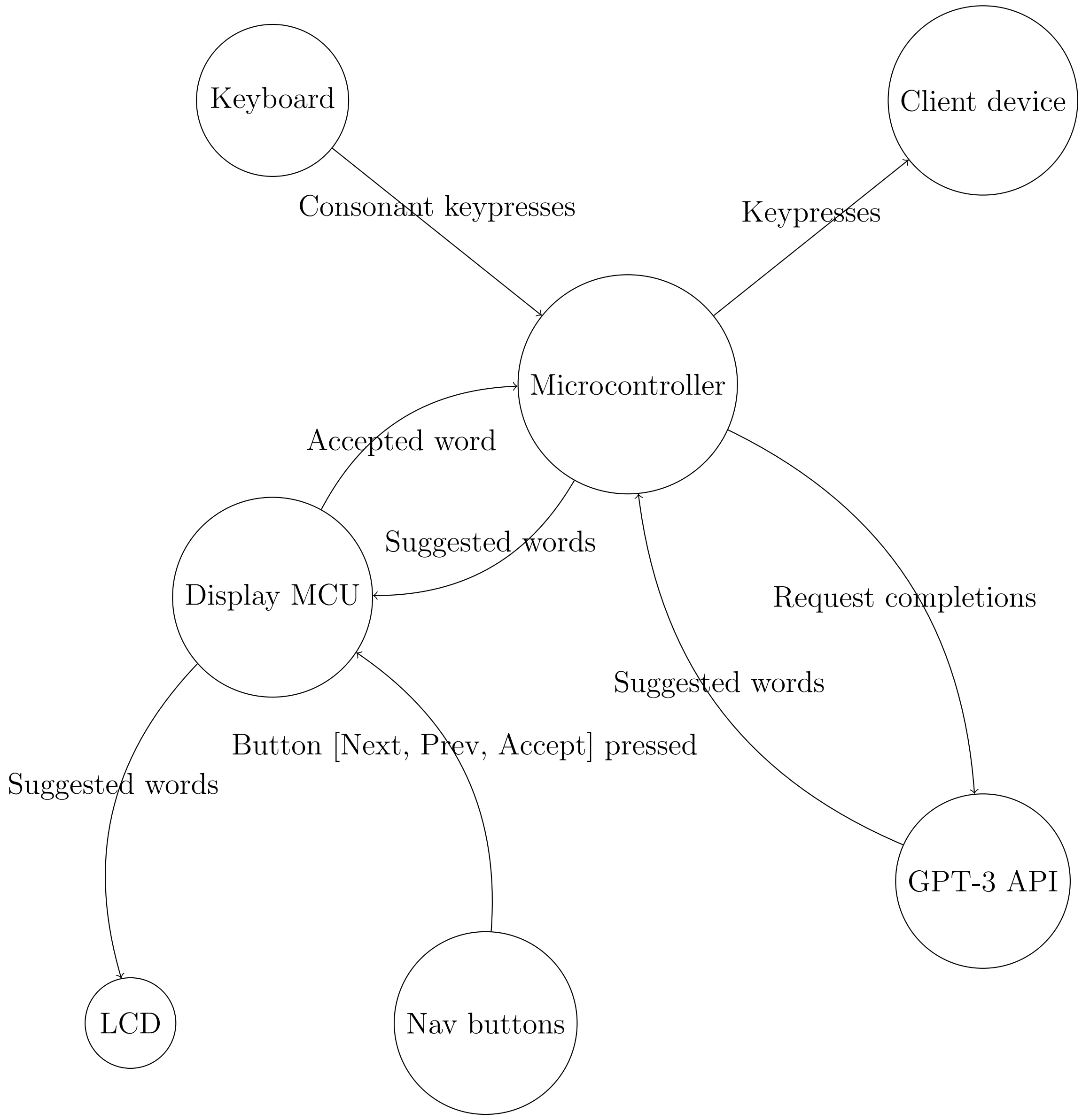
R3a If the LCD is displaying completions, it shall visibly highlight one 'focused' word.

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

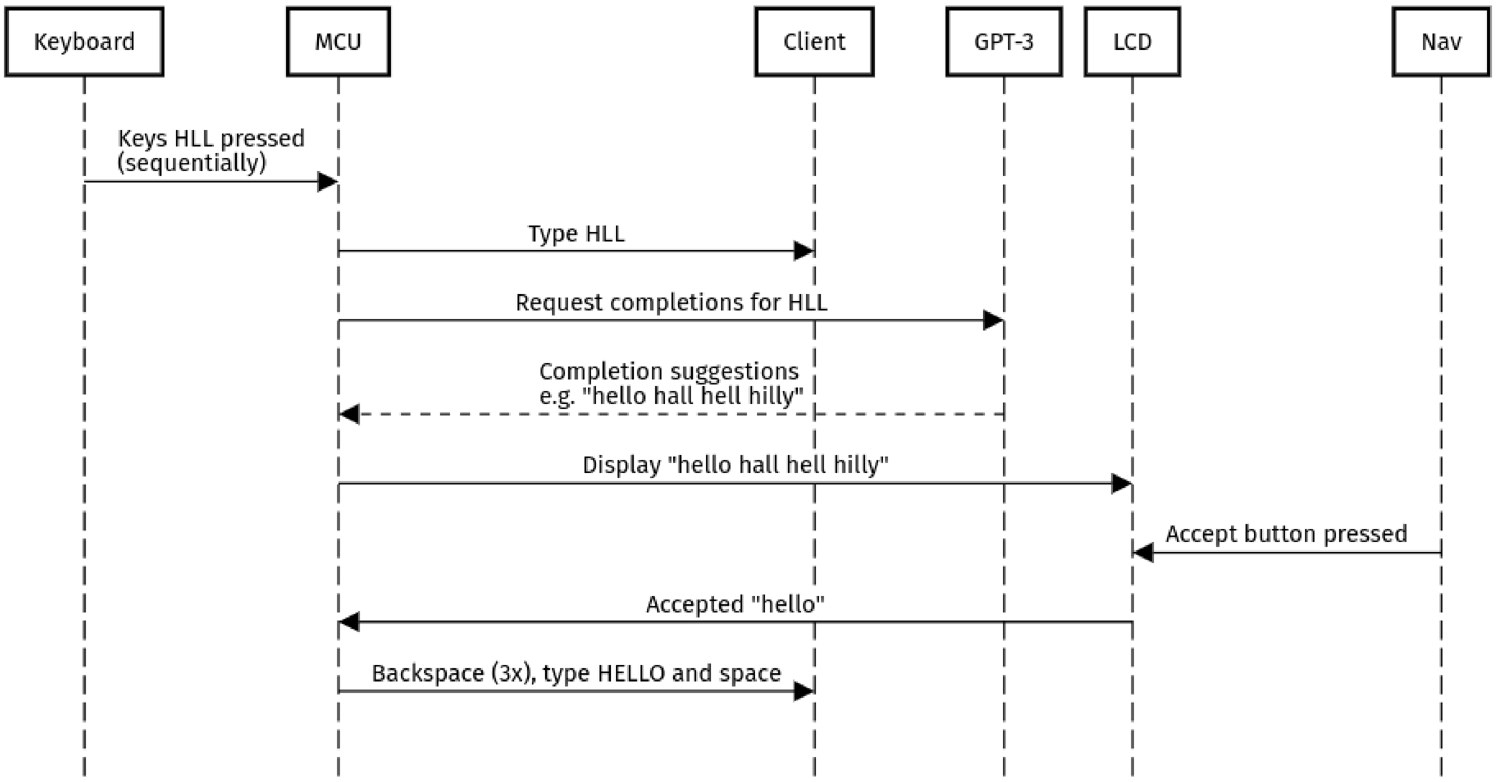


4 Use case scenarios

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).

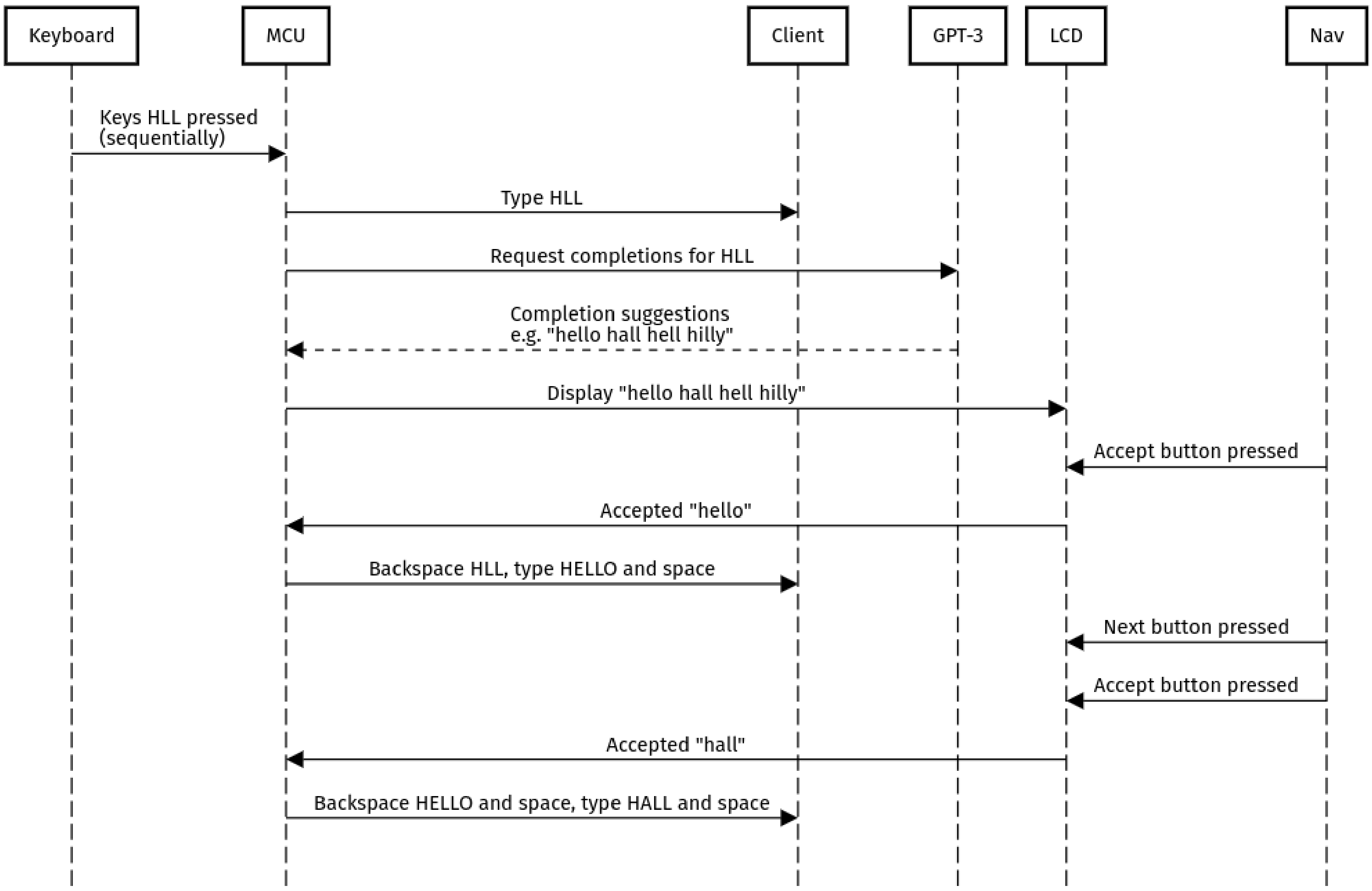
This diagram depicts the steps required to type ”hello” on the keyboard: the user types in the word's consonants (HLL), which are typed out on the client device, the system requests suggestions from GPT-3, which are displayed on the LCD, the user selects the appropriate word using the nav buttons, and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

Typing "hello"

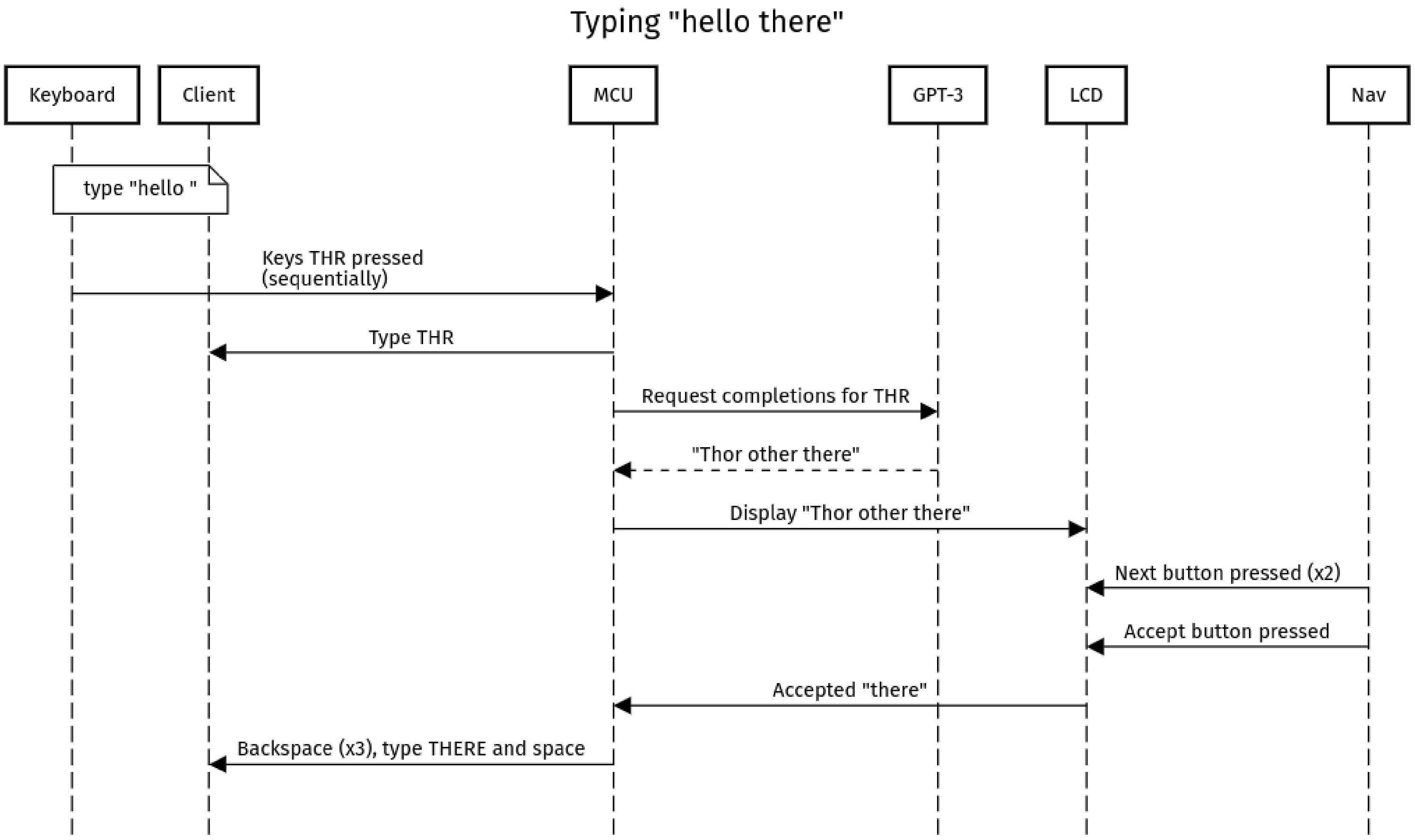

Keyboard
MCU
Client
GPT-3
LCD
Nav
Keys HLL pressed (sequentially)
Type HLL
Request completions for HLL
Completion suggestions e.g. "hello hall hell hilly"
Display "hello hall hell hilly"
Accept button pressed
Accepted "hello"
Backspace (3x), type HELLO and space

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


Keyboard
MCU
Client
GPT-3
LCD
Nav
Keys HLL pressed (sequentially)
Type HLL
Request completions for HLL
Completion suggestions e.g. "hello hall hell hilly"
Display "hello hall hell hilly"
Accept button pressed
Accepted "hello"
Backspace HLL, type HELLO and space
Next button pressed
Accept button pressed
Accepted "hall"
Backspace HELLO and space, type HALL and space

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 ”.

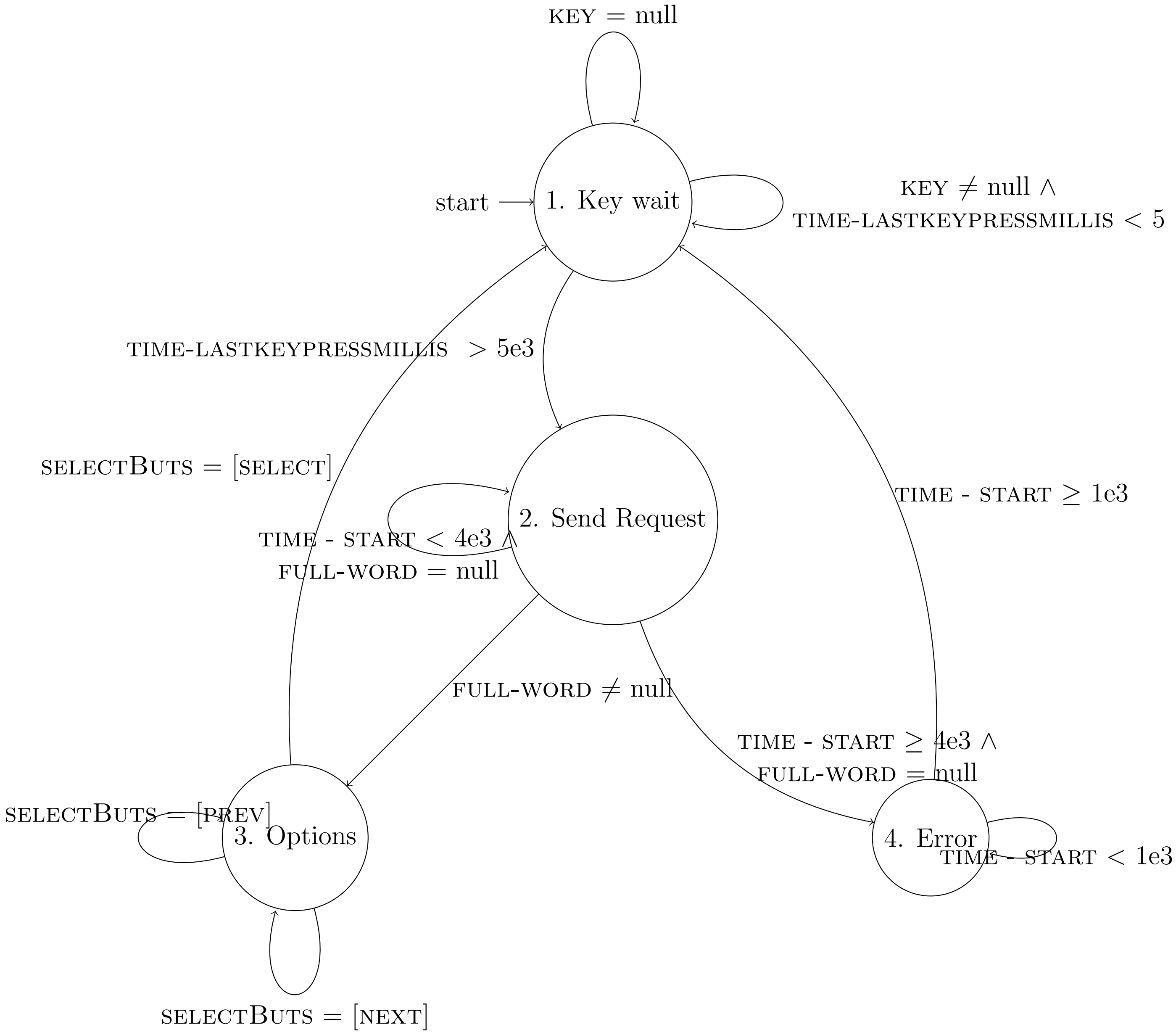


5 Finite state machine model

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 full word received by GPT-3 and selected by the user.

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).
- FULLWORDS: the full word options, with vowels inserted.
- LASTKEYPRESSMILLIS: time since last key was pressed
- SELECTEDOPTION: which option is currently selected


KEY = null
start
1. Key wait
KEY ≠ null ∧
TIME-LASTKEYPRESSMILLIS < 5
TIME-LASTKEYPRESSMILLIS > 5e3
SELECTBUTS = [SELECT]
2. Send Request
TIME - START < 4e3 ∧
FULL-WORD = null
TIME - START ≥ 1e3
FULL-WORD ≠ null
TIME - START ≥ 4e3 ∧
FULL-WORD = null
SELECTBUTS = [PREV]
3. Options
4. Error
TIME - START < 1e3
SELECTBUTS = [NEXT]

Transition	Guard	Explanation	Actions
1-1a	len(keys) = 0	No keys pressed.	-
1-1b	len(keys) > 0	Some letter keys were pressed.	buf += keys

1-2	time - lastkeypressMillis > 5e3	The word is sent for completion.	buf += keys - ” ”;
			recreate_word(buf);
			buf = ””;
			start = current_time();

2-2	time - start < 4e3 ∧ full-word = null	Waiting for recreated word.	full-words = get_recreated_words();

2-3	full-words ≠ null	GPT3 returned options for completions	selectedOption = 0;
			display_opts(selectedOption, fullWords);

2-4	time - start ≥ 4e3 ∧ full-words = null	Request to recreate word timed out.	start = current_time();
3-1	selectButs = [select]	Select the current option and type it out.	type_word(fullWords[selectedOption]);

3-3a	selectButs = [next]	Scroll to next completion option.	selectedOption = (selectedOption + 1) % len(fullWords);
			display_opts(selectedOption, fullWords)

3-3b	selectButs = [prev]	Scroll to previous completion option	selectedOption = (selectedOption - 1) % len(fullWords);
			display_opts(selectedOption, fullWords)

4-1	time - start ≥ 1e3	Reset the system.	buf = ””;
			full-word = ””;

4-4	time - start < 1e3	Displaying error message.	display_error();

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 vowel filler. To unit test the keyboard, we mocked out key presses and key states with the buffer that already exists as a part of our keyboard. Using these techniques we implemented the following unit tests:

- Test that no action occurs when no keys are pressed (via passing empty KEYS input). (Transition 1-1a).
- Test individual key pressed (passed into FSM KEYS) for each key and verify that the right letter is added to the buffer. (Transition 1-1b).
- Test that multiple simultaneous key presses add one of the two letters to the buffer.
- Test that accepting a recreated word types it out and returns the FSM back to state 1. (Transition 3-1)
- Test that it's possible to type out multiple words in series.
- Test that before 1 second, the error continues to display (Transition 4-4). After 1 second, the system is reset (Transition 4-1).

These tests cover our the scenarios depicted in our use case sequence diagrams. We did not test the actual API calls to GPT-3 and instead tested it as a part of our end-to-end system testing since the part is too non-deterministic and costs money to run.

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, request GPT-3 to complete what the user typed (Transitions 1-2,2-2), and receive a list of candidate words displayed on the LCD (Transition 2-3), which the user will then be able to cycle through and select (Transition 3-3a,3-3b).
We believe this amount of testing is adequate as we would achieve transition coverage of our FSM and coverage of our Sequence Diagrams.

8 Safety and liveness requirements

Below, being in state n is abbreviated as Sn.

8.1 Safety requirements

- $G(S1 \land \textsc{keys} = \varnothing \implies X(S1))$

  In every state, if we are in state 1 of the FSM ("Key wait"), and no key was pressed, then we will continue waiting for a key (in "Key wait") in the next state.

- $\forall b, \forall k, G(S1 \wedge \textsc{keys} = \{k\} \wedge \textsc{buf} = b \implies X(\textsc{buf} = b + k))$

  If we are in the "key wait" state and a single key is pressed, then the buffer will have that letter appended in the next state.

- $\forall s, G(\text{len}(\textsc{fullWords}) > 1 \wedge \textsc{selectButs} \notin \{\varnothing, [\textsc{select}]\} \wedge \textsc{selectedOption} = s$
  $\implies X(\textsc{selectedOption} \neq s))$

  If there is more than one option for completing the word and either the up or down selection keys are pressed, the next state's selected option will differ from the current selected option.

- $G(\forall w \in \textsc{fullWords}, w \in \texttt{[a..zA..Z]+})$

  The word completions are always non-empty alphabetic strings.

8.2 Liveness requirements

- $G(\textsc{keys} \neq \varnothing \implies F(\textsc{fullWords} \neq \varnothing))$

  If a key is typed, we will eventually have some completion options available for the user.

- $\forall k, G(\textsc{keys} = [k] \implies \exists w, k \in w \wedge F(\text{type\_word}(w)))$

  If a consonant is typed, we will eventually have a word typed containing that consonant.

- $G(\textsc{fullWords} \neq \varnothing \implies \exists w \in \textsc{fullWords}, F(\text{type\_word}(w))$

  If there are a non-zero number of completion options, then one of the options will eventually be typed in.

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.
- Time. Time informs when our system transitions, for example from state 1 to state 2, or state 2 to 4. Therefore, we should model time moving forward in order to be able to compose with with our time-dependent FSM. Such a model would need to be *hybrid*, *deterministic* system since time changes continuously, but deterministically forward.

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.
  Wifi/Serial/Timer Req: 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.

  PWM/DAC/ADC Req: 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.

  ISR Req: 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.
  Watchdog Req: 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. We mock some components such as writing row wires and reading column wires to we can test the keystroke detection algorithm but this is handled within the test setup and doesn't require setting any macros.

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 and 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 correspond 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	Fixed
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"	Fixed
2-2	May potentially want to add a check within the transiton 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 unitll the user has entered all of their desired consonants.	Fixed
2-3	I would recomend 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 transiton which simply resets based on time but doesn't take into account a user's potential action where they want to type in differnt words in a short time span.	Fixed
3-1	For the transiton 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

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?	Fixed
transition 1-2	Guard is "SPACE is in keys" but FSM only says "SPACE" on the guard	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?	Fixed
	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
Fixed	N	Inputs, outputs, and variables are defined
Fixed	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

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	Fixed
State 2	Full word seems like it shoud be reset to null if we transition to state 1 after state 2	Fixed

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