Meal by Words

Nina Duan, Shiyi Zhang, Lisa Xiong

Department of Electrical and Computer Engineering,
Carnegie Mellon University

***Abstract*—Inaccurate orders and long wait times are critical problems that discourage customers from ordering at quick-service restaurants. With the traditional ordering method, customers frequently have to raise their voices or even shout to ensure that the person taking their order can hear and understand it accurately. Some restaurants provide touch screens as an alternative which has the potential of spreading harmful bacteria and diseases. To tackle these problems, we propose the development of a food ordering system that employs speech processing technology to facilitate the process of placing orders at restaurants — Meal by Words.**

***Index Terms*—Design, verification, sensor, embedded system, signal processing, speech recognition, natural language processing, cloud database, user interface**

I. Introduction

The understaffing issue of some fast food restaurants results in either exhaustive customer wait times, or burnt out employees shuffling between the kitchen and the counter. The current approach of having cashiers take orders also heavily relies on employee training, which consumes both time and money.

Some fast food restaurants installed touch screen ordering kiosks to solve these problems; however, after the pandemic, the public raised health concerns about unsanitized shared utilities to a new level. Fox News reported that a swab test on eight random McDonald's restaurants in the United Kingdom resulted in the discovery of infection-causing bacteria in every test[1]. The existing ordering kiosks, with thousands of people making orders by touching their screens, will carry lots of bacteria if not sanitized properly on a regular basis.

Kiosks that can take verbal orders will reduce the burden on fast food restaurant staff and the customers' health concerns. The goal of our project is to build a voice-operated ordering kiosk that processes orders efficiently. The entire process except payment will be completed verbally. The confirmed order will be sent to the kitchen automatically, allowing the staff to start preparing as soon as an order has been placed. The ultimate goal is to create an ordering experience that mirrors the one a human employee offers, but with greater efficiency, accuracy and safety.

II. Use-Case Requirements

Available operations: A regular fast-food order interaction includes a customer requesting desired items as well as proceeding to checkout. The system will also provide two additional features — the ability to remove item(s) from the order and modify the quantity of items ordered — to ensure that incorrectly-recognized or incomplete orders will not progress to later stages of our system.

Service time: According to SeeLevel HX's 2016 research, the average service time in the U.S.-based fast-food restaurants is around 200 seconds[2]. Our system should offer a similar if not better experience. Hence, we expect the entire ordering process for one customer to be completed in less than 200 seconds.

Customer-staff communication latency: The kitchen staff should be able to see a newly placed order within 1 second after the customer checks out. This will ensure customers' food is prepared in a timely manner.

Voice reception: To accommodate fidgeting customers, the microphone should be able to receive inputs from a horizontal span of 120°, from a distance between 0m to 1.5m. The microphone and speech processing algorithms should accommodate all human voice frequencies (80Hz ~ 260Hz), at normal conversational volume (60dB ~ 80dB).

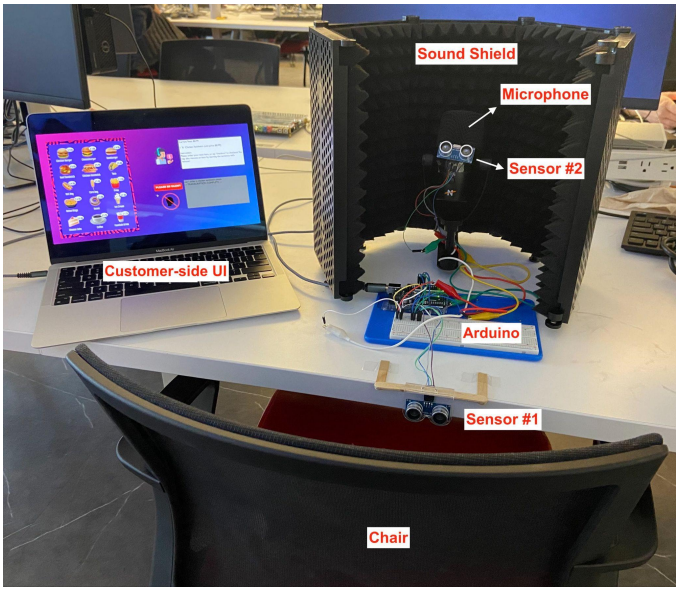
Noise tolerance: Since most fast-food restaurants have noisy environments, our system should be able to operate in a considerable level of background noise. To ensure a satisfactory experience, the system should reach 100% order accuracy by the time a customer checks out.

Process termination: The system should be able to recognize that a customer has left the kiosk, delete the current order, terminate the ordering process of this order, go into SLEEP mode, and be ready to start a new order. It should not terminate the current ordering process under any circumstances, except when the system's sensors detect that the customer has left, and a 30-second timeout period has elapsed. This is done to prevent accidental deletion of a customer's order, which could lead to frustration and inconvenience.

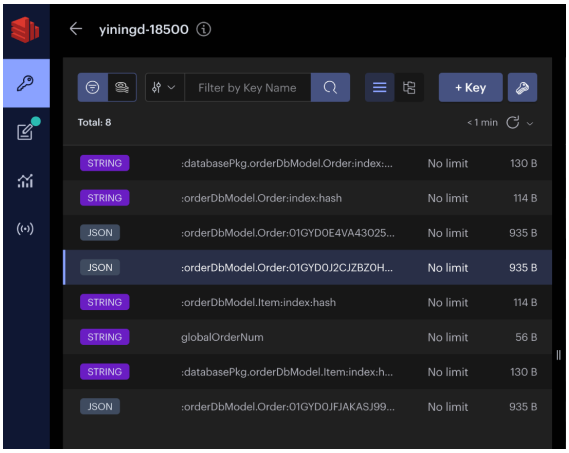
Power conservation: Our system aims to minimize power consumption to promote an environmentally friendly ordering experience. Therefore, it should process audio inputs only if a customer is present. If a customer is not present, the system goes into SLEEP mode where it only monitors whether a person is present. Once a person is detected, the system will switch to ACTIVE mode.

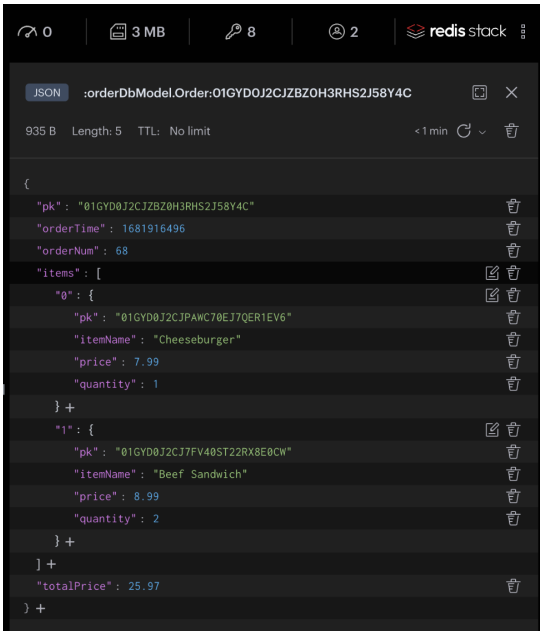
Accessibility: To ensure that our ordering system accommodates all customers, we need to consider two factors: the customer's height and how the customer can interact with our system. Our system should be able to detect the presence of a customer taller than 70 cm. This will accommodate those in wheelchairs, because the eye level of a person in wheelchairs is 109 cm to 129.5 cm[3]. To accommodate customers with sight disabilities, the system should be able to read out the instructions and order details during the ordering process. Similarly, displaying the instructions and order details on the screen allows customers with hearing disabilities to be able to interact with our system.

III. Architecture and/or Principle of Operation

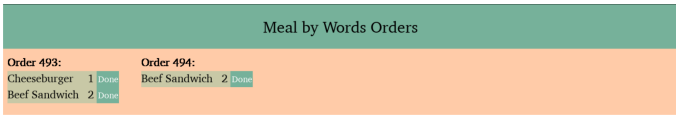


(a)





(b)



(c)

Fig. 1. Overall system. (a) Customer-side setup. (b) Redis Cloud Database. (c) Staff-side UI when two orders, #493 and #494, are yet to be prepared.

On a higher level, our project can be divided into a hardware component and a software component. The hardware component is responsible for handling the system's interaction with the physical world, which involves receiving physical data such as distance readings and sound waves. It provides the necessary information for the software component to process and respond to. The software component receives the mentioned information, converts them into meaningful tokens, and stores them in a non-local location for safekeeping. The software component is also responsible for fetching and removing stored data in the future.

In absence of a customer, the system will be in SLEEP mode and display animation on the monitor. An ultrasonic distance sensor, driven by an Arduino Uno, is installed on the edge of the table, horizontally facing the chair on which a customer should be sitting (Fig. 1a). An interaction with our system begins when a customer sits down on the chair placed in front of the kiosk. Once the system picks up the changes in distance, the system goes into ACTIVE mode and starts a new order.

We switched from using an RPi4 and infrared sensors to using an Arduino and two ultrasonic distance sensors. The switch from an RPi4 to an Arduino was due to the fact that RPi4 is not compatible with some of our speech recognition code. We replaced the infrared sensors with ultrasonic distance sensors halfway through the semester to improve the accuracy of customer detection. Previously, the infrared sensors would sometimes detect people passing by the kiosk and falsely activate the kiosk, and would sometimes detect a customer when there was actually none, which had resulted in orders being unable to be terminated. After switching to using the ultrasonic distance sensors, which produce more accurate readings, we can limit the detection range and activate the kiosk only when a person is likely sitting down on the chair, which reduces power waste and improves efficiency.

Information displayed on the screen includes the food menu, food items that have been added and their respective price, the order total, and instructions that are constantly updated to guide the customer through the ordering process. For a better user experience and to accommodate people with sight disabilities, the instructions are also read out by a synthesized voice. Customers will be prompted to place their order item by item, instead of providing the entire order in one long sentence. We made the change after the design report as it made detecting the end of speech easier, which, in turn, drastically improved the accuracy of our speech recognition system.

A directional microphone, surrounded by a professional sound shield, will receive speech inputs from the customer. This input stream is fed to a signal processing algorithm for noise reduction. The speech recognition module takes the sanitized signal and, using a trained machine learning model, converts it to a text string.

Another distance sensor, in addition to the one installed on the table, is attached to the microphone to check if the customer is speaking close enough to the microphone. Whenever it is the customer's turn to speak, and the customer is speaking too far away from the microphone, a warning will be displayed on the customer-side UI.

Our natural language processing algorithm parses the text string and extracts necessary order information, including the food item's name, the quantity of the item, and the customer's desired action (add, remove, check out, or confirm). All of the parsed menu items and quantities are stored as local objects before checkout. If the NLP module fails to recognize any item name in ten seconds, it will notify the UI of this failure. The customer-side UI will then display an error message and kindly ask the customer to repeat their request.

To avoid further complicating our NLP system, we decided to limit checkout and order confirmation keywords to a fixed set of phrases. Now, the customer simply has to say the keyword "check out" to checkout. When the NLP system detects this keyword, our system will instruct the customer to review their order, displayed on the customer-side UI. The customer can confirm the order by saying "yes", "correct" or "confirmed". To return to the ordering process, they can either say anything else or remain silent.

After the customer checks out, the entire order, previously stored as a local object, will be uploaded to the Redis cloud database (Fig. 1b). Information in this database is accessible to the staff-side UI. The database notifies the staff-side UI of the new order by publishing a notification message. The customer-side UI will also display the customer's unique order number, which they will use to pay for and pick up their order. After 10 seconds, the customer-side UI will go into SLEEP mode, waiting for the next customer.

When notified, the staff-side UI (Fig. 1c), used by the restaurant's kitchen staff, queries the database for new order information. For ease of use, the UI sorts, in ascending order, existing orders based on the time they were placed (i.e. the oldest will rank first and the newest last). There is a deletion button for all menu items on the screen to be removed after the kitchen is done with preparation. When an item entry has been removed, it both disappears from the screen and gets deleted from the database. The lifetime of an order ends when the kitchen staff removes all items in it from the database.

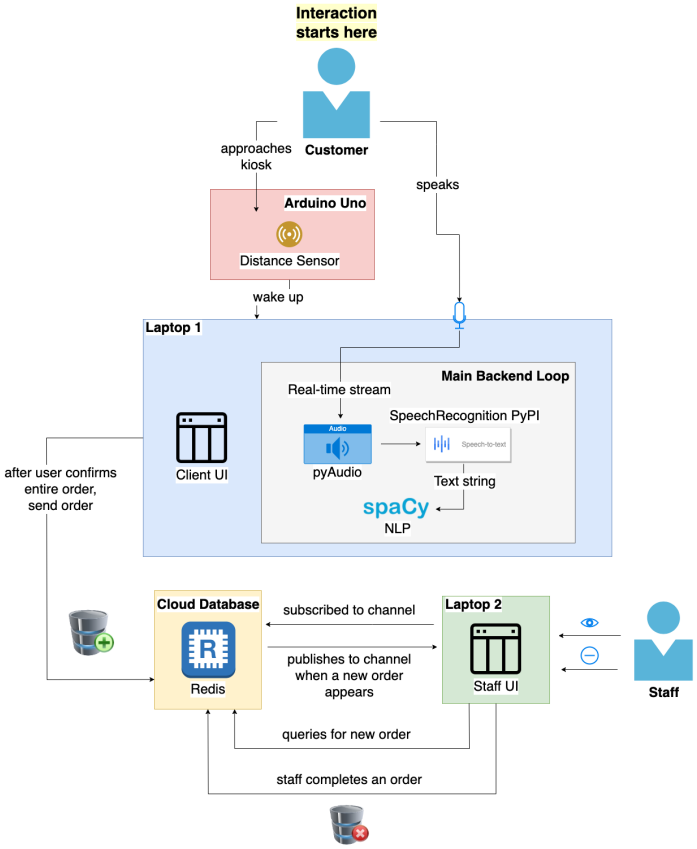


Fig. 2. System block diagram.

IV. Design Requirements

Customer detection: The sensor subsystem is required to detect the presence of a customer within 2 seconds since the customer shows up with 100% accuracy. To account for any slight movements, the subsystem considers that a person has left only after a 30-second timeout period has elapsed.

Noise tolerance: In the article "Noise in Restaurants: Levels and Mathematical Model," Dr. To states that a typical fast-food restaurant's noise level falls in the range of 69.1dBA to 79.1dBA[4]. Humans' hearing accuracy drops significantly when the noise level is greater than 70dBA, measuring a mere 50% average in Raghavan's paper[5]. Therefore, we expect an 85% speech-to-text accuracy at a noise level < 70dBA and a 50% speed-to-text accuracy at a noise level >= 70dBA. This will ensure a relatively high success rate for matching customer speech and order items.

Data storage: As a general design rule, the representational gap between a physical order and its stored counterpart should be low. The stored information should be tolerant against unexpected events such as hardware failures, power outages, and network disruptions. The data should be accessible from multiple sources at the same time, as there may be multiple kitchen staff preparing orders at the same time. Following our latency requirement, the data storage model should also guarantee a <= 1s latency between order placement from customer-side UI and appearance in staff-side UI.

Customer-to-Staff Latency: To ensure that the kitchen staff sees a newly placed order within 1 second of the customer checking out, the staff-side UI is subscribed to the Redis channel where the customer-side publishes orders in to get real-time notifications. When an order number is received, it immediately queries the database to retrieve order information and display it on the screen.

Backend-to-Customer Responsiveness: The backend must be able to recognize the end of the speech and deliver the speech to the speech recognition processor. After the speech is processed by the backend, it must immediately update the customer-side UI (<= 1 second) so that the UI can confirm the receival of the speech with the customer by displaying an icon, letting the user know that the speech has been received and please wait for further instructions. In addition, we will increase the customer-side UI rendering speed using AJAX to improve performance, reducing the traffic from and to the UI.

UI Readability: The customer-side UI should accurately display the customer's order items, their respective prices, and the order's total price. The texts should be clearly visible to people with 20/20 vision from at least 0.5m away. The staff-side UI should display orders in easily readable format, allowing an average person to comprehend an order's information within 3 seconds. The UI should group items based on order rather than item type, with the oldest order displayed first.

V. Design Trade Studies

Based on our use-case and design requirements, we made five key decisions.

A. Customer Detection Trade-Offs

By having two ultrasonic distance sensors, the speech recognition system is capable of going to SLEEP mode when there is no customer present and check if the customer is speaking close enough to the microphone when it is the speaking period.

Passive infrared sensor vs. ultrasonic distance Sensor: The infrared sensors would sometimes falsely activate the kiosk, and ignore a customer under certain circumstances, which had resulted in orders failing to be terminated. The ultrasonic distance sensors, which produce more accurate readings, can limit the detection range and activate the kiosk only when a person is detected sitting down on the chair, which reduces power consumption and improves efficiency.

Two distance sensor vs. one distance sensor: we could have used only one distance sensor to detect both the presence of the customer and whether the customer is speaking close enough to the microphone. However, to improve accuracy, we decided to use one distance sensor to detect the presence of the customer and a separate one for monitoring the distance between the customer and the microphone. Otherwise, we would have to adjust the angle of the only sensor.

B. Speech Recognition & Parsing Trade-Offs

Saying all order information in one go vs. saying entries one at a time: Our initial plan was to process voice inputs as they are received, allowing the customers to say all entries in the order in one go while our system parses the input. Nevertheless, determining the time to stop recognizing speech was extremely difficult. Due to the heavy background noise in restaurants, determining a decibel level as termination cutoff became unrealistic. Setting a timeout would also lead to troubles since the time it takes to place an order completely depends on the customer. If the customers can say item entries one at a time, the accuracy of speech recognition will be significantly improved.

C. Front End Technology Trade-Offs

An open-source front-end library or framework will make our UI visually consistent and appealing. We debated between the React library and the Bootstrap framework but, ultimately, chose the latter. Bootstrap not only provides a library of off-the-shelf components and styles, but also various other supports that help boot up a standard project quickly. It allows rapid prototyping of a web application without the design work React requires. Our UIs do not have complicated functionalities that need heavy customization, so Bootstrap is an effective and efficient solution.

D. UI Design Trade-Offs

Digital menu vs. printed menu: There are two options to choose from: displaying the menu on the user interface or hanging a printed copy somewhere near the kiosk. Although printed menus are larger than our UI screen (i.e. laptop screen), which is easier to read for customers, we proceeded with the digital menus for easy menu change and environmental friendliness. The menus no longer need to be

reprinted everytime there is a change in the menus, and we can reduce our paper consumption.

Visual instructions vs. verbal instructions: To improve our system's accessibility, we will both display the order instructions on screen and use a voice synthesizer to read it out loud. Verbal instructions are more intuitive to our customers because they simulate the human cashier's behavior at a regular counter. If there is no text display, people with hearing impairment may struggle to understand the instructions or navigate the UI, which leads to a frustrating user experience. That was how we decided to include both features in our customer-side UI.

Web-based UI vs. GUI: When developing the staff-side UI, the initial solution was to build a web application similar to the customer-side UI. This would have been a complicated and time-consuming process, even unrealistic given the short timeframe allocated for the staff-side UI. We soon realized that the Pygame library can implement all the desired functionalities locally, which became our alternative solution and saved us a lot of time.

E. Data Storage Trade-Offs

We mainly studied two options for customer data storage: local and cloud storage.

TABLE I. DATA STORAGE TRADE-OFFS

Criteria	Storage Options	
	Local Storage	***Cloud Storage***
Storage Formats	JSON file CSV file In-program global structures (e.g. map, set, dictionary, etc.)	NoSQL key-value pairs SQL tables (relational) Documents (JSON-like)
Difficulty	Low. Uses information team members have already learnt in other courses.	High. Requires extra ramp-up time.
Network	Doesn't require network access for storage. Requires network access for querying from a different machine.	Requires network access for storage and querying.
Storage Speed	Fast. Minimal latency.	Slow, but within sub-microsecond range.
Query Speed From Other Machines	Slow. Requires machines to send local data over the network.	Slow, but within sub-microsecond range.
Durability	Low. System failure results in data being inaccessible or permanently lost.	High. Cloud services provide fail-safe measures.

To reiterate, our main design requirements for data storage are 1) low representational gap between order and stored information, 2) high tolerance against hardware failures, 3) ability to access stored data from multiple machines, and 4) <= 1s latency between order placement from customer-side UI and appearance in staff-side UI.

Object-oriented programming is a good way to maintain a low representational gap between physical orders and the stored order information. Locally, JSON files, CSV files, and data structures such as dictionaries and sets can all store custom objects. On the cloud, this requirement disqualifies key-value storage, as the keys and values are generally both string or integer values. While the SQL table format supports mapping between tables (e.g. an entry in the Order table can contain a field that maps to an entry in the Item table), it establishes undesirably high coupling between them. Relational databases also maintain complicated internal data about the stored objects, which inherently makes insertions and deletions slower. In the end, the document format emerges as the most ideal cloud storage solution.

A major shortcoming of local storage is the vulnerability to hardware failures. If the hardware that stores our order information drops offline, the data will be inaccessible until the system goes back on line. If a file becomes corrupted during reboot, we run the risk of losing the data entirely. By using cloud storage, we mitigate this risk by creating a backup copy in a remote location.

Cloud storage also inherently provides the ability to access data from other locations. It decouples our two UIs, the customer-side and the staff-side UIs. Both UIs will send and request information from the database. Local storage, on the other hand, requires our system to send new order information from the customer-side UI to the staff-side as we receive them.

Given the benefits of cloud storage, we believe that it is reasonable to pay the price of slightly higher access latency. The cloud database we have chosen to work with, Redis, generally guarantees a sub-microsecond range latency, which is significantly lower than the one second latency we are aiming for.

Redis is an open-source cloud database that 1) supports both the key-value and the document storage models, 2) allows customizations on data structures, and 3) provides a notification system useful for communication between our customer-side and staff-side UIs. The third functionality is particularly useful, since this eliminates the need of busy-polling the database for new order information.

VI. SYSTEM IMPLEMENTATION

A. Customer Detection

To detect the presence of a customer, two hardware components are needed: an Arduino Uno and an ultrasonic distance sensor (Fig. 3). Whether a customer is present is determined by the distance reading. The distance is computed by a Python script running in a process independent from the main process. This independent process will be constantly polling the distance readings printed to the serial port of the arduino, which connects to the ultrasonic distance sensor. The main process will poll the distance readings from the independent process once every 1 second. The timeout logic replies on whether a signal was recently received. Therefore, the backend will have a loop that constantly validates the timestamp of the most recently received sensor signal.

Fig. 3. HC-SR04 ultrasonic distance sensor.

B. *Speech Recognition & Parsing*

Our system uses a USB directional microphone, Neat Bumblebee II, to capture voice inputs. To produce the best result, a professional sound shield, Moukey Microphone Isolation Shield, surrounds the microphone and limits its reception range to <=90° vertically and <=120° horizontally.

Input from the microphone is fed into a signal processing module, implemented with the open-source library PyAudio. The listen() function will capture signals from the microphone. This module uses Active Noise Reduction (ANR) to reduce background noise in real-time and relays the resulting stream to the speech recognition module.

The speech recognition system is implemented with the SpeechRecognition Python library. It calls upon an existing, trained speech recognition engine and converts the input to text. A while loop makes sure that our system continues to take user input when the sound level is above a certain decibel.

The NLP system is implemented with the spaCy library. The NLP pipeline first tokenizes all words in the user input text, and then numerizes the input text string to convert any numeral words to numbers. The grammar rules used to find item names, quantities and commands are all written into dependency matchers, which parse the tokens based on grammar structure (Fig. 4). After checkout, the information extracted will be uploaded to the Redis database and immediately retrieved by the staff-side UI.

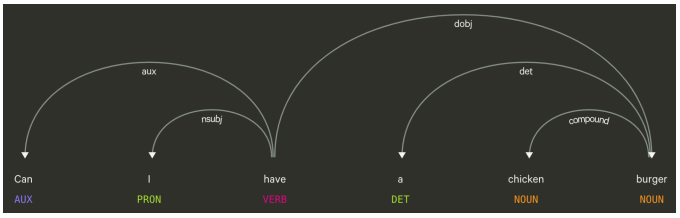


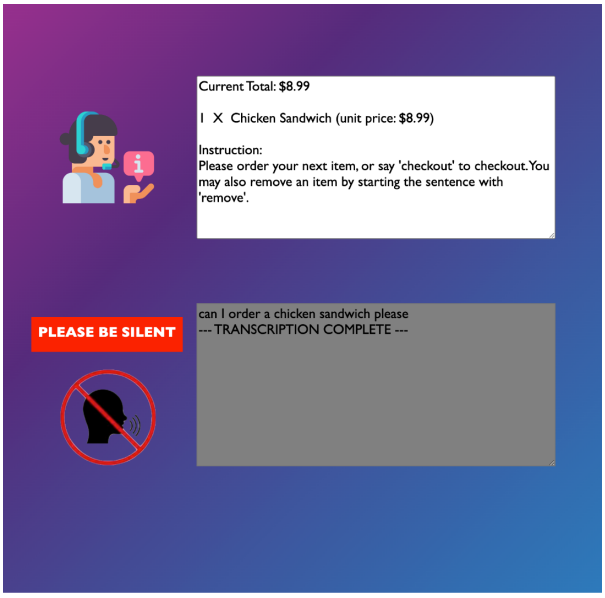
Fig. 4. Example spaCy parsing result. The sentence parsed here is “can I have a chicken burger?”

C. *UI & Front-end Framework*

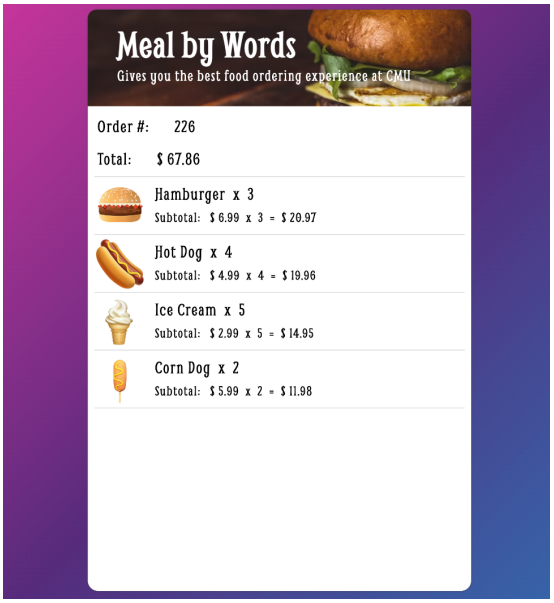
The UI regularly retrieves necessary variables from the backend and populates the reserved fields in the HTML file. To illustrate, when the speech recognition program finishes processing the ordered items, it should encode them into JSON format and send the data to the JavaScript body in the order summary HTML file to be decoded. JavaScript will then fill the HTML fields with the decoded variables.

Customer-side UI: On the customer-side UI (Fig. 5), the customer will be able to see the menu, the items they have ordered, their corresponding prices, and the total price of their order. A green microphone icon indicates the speaking period and that the customer should be speaking at the moment. When the system is processing, the icon turns red. In the meantime, a pre-synthesized voice recording of the instruction will be played. A textbox will display the same message to accommodate those with hearing disabilities.

(a)



(b)



(c)

Fig. 5. Customer-side UI. (a) Left side of the ordering page. (b) Right side of the ordering page. (c) Checkout page.

Staff-side UI: The Pygame library was used to render the staff-side UI page. In the main loop, the UI actively listens for updates in the Redis channel of orders; when a new order has been placed, the information will be appended to the current list of orders and be displayed after the current iteration. The "Done" button after each order entry allows the staff to delete completed items. When an entry is completed, it is both removed from the UI and the database; orders with no entries remaining will be automatically deleted as well.

D. Data Storage - Cloud Database

Our system communicates with the open-source database Redis to store and retrieve customer order information. The Redis Object-Mapping (redis-om) module, an open-source library, maps object-oriented data models in Python to those on the Redis Stack. Redis also provides a publish-subscribe (Pub/Sub) functionality that allows the publisher to send messages to one or more subscribers.

The menu is immutable and stored locally. A menu item's name is unique, and the menu contains its corresponding price.

Although Redis offers fast and reliable accesses, frequently retrieving data from the cloud will still stall the pipeline. In order to minimize communication with the database, order information will be kept in local storage before the customer checks out (Fig. 6).

A local order is responsible for storing and managing information about an on-going order. Each order is uniquely identified by an order number. The time the order is created (system time, in seconds) will be documented as the order time, which allows the staff-side UI to sort orders based on processed time. Menu items' names act as the keys to the local order's items dictionary. Their corresponding values are the ordered quantity. The total price of the order will be updated as customers add or remove items.

In the cloud, orders are stored in JSON format with models that are declared through redis-om. There are two models in total: order and item. Item is an embedded model representing a menu item belonging to a database order. A database order consists of attributes corresponding to those of a local order.

To upload a local order to the cloud, the downstream dependency only needs to call a local order's checkout method. This method automatically parses the local order's contents and creates its cloud counterpart, database order.

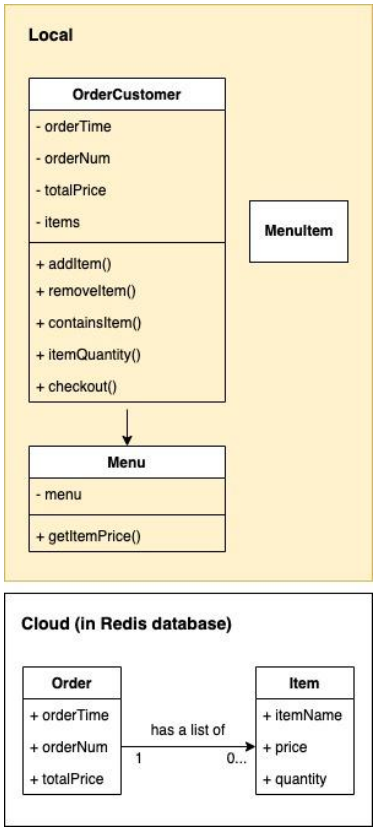


Fig. 6. Object model for local and cloud orders.

Once the order has been stored to the database, the customer-side UI needs a way to notify the staff-side UI for it to fetch the new order.

Redis's Pub/Sub functionality provides exactly this. The customer-side UI is set up as the publisher for the channel. When an order has been successfully added to the database, the customer-side UI will publish a message including the order's unique order number. The staff-side UI, subscribed to the same channel, can immediately query the database with the received order number and retrieve the new order information.

The staff-side UI's only other interaction with the database is order removal. A successful removal request completely erases the order information from the database.

VII. Test, Verification and Validation

Since our project is heavily customer-focused, we invited 5 volunteers with varied gender, heights, accents to experience the ordering process. During testing, we kept track of the

following criteria:

A. Results for Customer Detection

Our volunteers of different heights come up to the kiosk and sit on the chair, adopting postures that they felt comfortable within the context of the scenario. Our volunteers were free to move their bodies as long as they stayed on the chair. The success rate can be measured by (# of times the system successfully switches from SLEEP mode to ACTIVE mode / # of attempts made by the volunteers we have tested). The success rate was 100%.

B. Results for Audio-to-Text Accuracy

Correct audio-to-text transcription is the foundation of our entire system. Due to the nature of our system, we mainly focus on the speech recognition system recognizing the correct word, having some leniency on its verb tense or singularity. Therefore, verbs of different tenses will be considered as the same word (e.g. “wake,” “woke,” “waken” are considered the same). Similarly, we would not distinguish between singular and plural nouns (e.g. “hamburger” and “hamburgers” were considered the same).

To test audio-to-text accuracy, we speak phrases commonly used for ordering (e.g. “I’d like xxx,” “I want to get xxx,” “xxx, please,” etc.) into the microphone and compare the transcription against our actual words. Accuracy is then computed from the number of words correctly transcribed divided by the number of words spoken.

Table II. Audio-to-Text Accuracy

Sentence Spoken	# of Words Spoken	Sentence Recognized	# of Correct Words	Accuracy
“I’d like two hamburgers.”	5	“I like to hamburgers”	3	60%
“I want two cheeseburgers.”	4	“I want to cheeseburger”	3	75%
“One beautifully-packaged chicken sandwich, please.”	6	“1 beautifully packaged chicken sandwich please”	6	100%
“I want to order a hundred cheesecakes.”	7	“I want to order 100 cheesecake”	7	100%
“Get me two hamburgers.”	4	“Get me to Hamburg”	2	50%
“I’d like one fries and three fountain drinks.”	9	“I like 1 fries and 3 fountain drinks”	8	88.9%
“Check out.”	2	“Check out.”	2	100%
“Hello, let’s go with four tacos and three ice creams.”	11	“Hello let’s go with 4 taco and 3 ice creams”	11	100%
“I’d like one cup of coffee.”	7	“I like 1 cup of coffee.”	6	85.7%
“Fifty corn dogs.”	3	“50 corn dog”	3	100%
Avg.				87.9%

The average accuracy across 10 samples was 87.9%, which satisfies our use-case requirement of an 85% accuracy.

C. Results for Text-to-Command Accuracy

We fed test string templates into our NLP system to test the accuracy. For each test case, [menu item] was replaced by all the item names on our menu. To better simulate the output of the speech recognition system, all capitalizations and punctuations were removed. The text-to-command accuracy was measured by the number of correctly displayed item entries over the total number of entries in the text input.

Table III. Text-to-Command Accuracy

Text Input	Addition Accuracy	Removal Accuracy
can I have a [menu item (sg)] please	100%	N/A
two [menu item (pl)]	100%	N/A
two [menu item (sg)]	100%	N/A
five beautifully packaged [menu item (pl)]	100%	N/A
a box of [menu item (sg)] please	100%	N/A
twenty [menu item1 (pl)] and three [menu item3 (pl)]	100%	N/A
remove [menu item (pl)] please	N/A	100%
i will get rid of [menu item (pl)]	N/A	100%
one [menu item (sg)] oh wait can i get rid of a terrible [menu item (sg)] please	80%	100%
i want three [menu item (pl)] can i delete two [menu item (pl)]	80%	100%

(sg = singular noun, pl = plural noun)

The NLP system was able to reach 96% accuracy on average for the test commands, which exceeded our goal of 95%. All the simple commands only involving addition or deletions resulted in 100% accuracy; the only test cases which went wrong are the ones where addition and deletion commands appear in the same sentence. Fortunately, based on our requirement for the users to order item-by-item, the complicated sentences should not occur in the use case of our system. The reason why we decided to leave out some complicated edge cases without fixing was that different menu items can generate different dependency trees even in the exact same sentence structure. Accounting for one edge case often means giving up on another.

D. Results for Order Upload Latency

The latency is measured from the time when order is sent to the staff-side to when staff sees the actual order. We will record the system time, in milliseconds, that the order is sent and compare it with the system time, in milliseconds, that the order is received.

TABLE IV. AUDIO-TO-TEXT ACCURACY

Network: CMU-SECURE (4/12/2023)			
Trial #	***Time Sent (s)***	***Time Received (s)***	***Time Difference (s)***
1	1681311756.920146	1681311757.988204	1.068058
2	1681311787.9240708	1681311788.301242	0.377171278
3	1681312039.466178	1681312040.4595578	0.9933798313
4	1681312140.5965528	1681312144.366512	3.769959211
5	1681312195.167861	1681312197.255147	2.087286
6	1681312260.151395	1681312263.733936	3.582541
7	1681312359.745095	1681312360.1503391	0.405244112
8	1681312407.597444	1681312417.1346428	9.537198782
9	1681312475.726104	1681312478.991681	3.265577
10	1681312525.983286	1681312526.7069042	0.723618269
11	1681743612.060843	1681743612.5518022	0.4909591675
12	1681743614.6954062	1681743615.077455	0.3820488453
13	1681743618.92501	1681743619.474676	0.549666
14	1681743624.654081	1681743625.0546181	0.400537014
15	1681744253.4430232	1681744254.947194	1.504170895
16	1681744293.227913	1681744294.318719	1.090806
17	1681744319.980497	1681744320.395576	0.415079
18	1681744338.203062	1681744338.6219149	0.4188528061
19	1681744356.008338	1681744356.6730611	0.6647231579
20	1681744377.0767202	1681744378.1253068	1.048586607
Avg.			1.683
Median			1.021

The average accuracy across 20 samples, collected over 2 days, was 1.638s. The median was 1.021s. An outlier of a 9s upload time was present in the tests, which affected the average significantly. We believe that the median is a better representation of our system's performance.

Overall, the latency falls in an acceptable range but fluctuates depending on the latency of the network.

E. Results for Service Time

The service time is measured from when the customer sits down on the chair until they finish checking out. We tested how long the service time is for different volunteers with a varied number of order items (3, 5, 7, etc.). The expected average service time is 200 seconds or less.

TABLE V. SERVICE TIME

Volunteer #	# of items ordered	Time consumed (s)
1	3	53.19
	5	76.93
	7	116.74
2	3	92.12
	5	168.95
	7	204.69
3	3	56.1
	5	79.57
	7	135.08
4	3	77.8
	5	94.23
	7	103.31
5	3	71.02
	5	129.42
	7	193.84
Average time consumed	3	70.046
	5	93.906
	7	150.732

We were able to keep our average service time below 200 seconds.

F. Results for Noise Tolerance

To test the system's noise tolerance, we generated background noise by playing recordings of white noises at the desired decibel level on loop while ordering. Noise tolerance is essential for our system since the use setting is fast food restaurant, a relatively noisy environment. Our system's performance under 75dB was 100% overall order accuracy, exceeding the expected 85%. However, it is important to note that we did not limit the duration of the ordering process, which meant our volunteers were able to add or remove items until the order was fully correct.

G. Results for Order Upload Accuracy

This test checks that the order received by the staff-side is the same as the order uploaded from the customer-side. We hard-coded 10 different orders, with varying order items and quantity, and uploaded them to the database.

The resulting accuracy was 100%. We found no mismatch between the order received by the staff-side and the order uploaded from the customer-side.

H. Results for Process Termination

For checkout, we will require volunteers to use the keyword "check out". The system should be able to handle checking out an empty order or a non-empty order. With an empty order, the system is expected to be able to terminate the ordering process

but ignore the empty order (i.e., not upload to the database). The system should be able to push a non-empty order (including the total and all the ordered items along with their corresponding quantities) to the database.

To test for timeouts, our volunteers simulated various scenarios where an order should be terminated prior to checkout, including walking away after the system has just been activated, leaving the kiosk halfway through the ordering process, and not speaking or speaking in a low voice when it is their turn to speak. In all of the scenarios mentioned, the system was able to time out in 30 seconds successfully.

I. *Results for Overall Order Accuracy*

We use recall as the measurement for order accuracy, that is, the number of correct item entries seen on the staff-side UI divided by the total number of entries the customer says. We were able to reach 100% order accuracy but running slightly overtime.

J. *Results for UI Readability*

We asked for feedback from the 5 volunteers on both our customer-side and staff-side UI. The design and layout were modified based on the comments to make sure the UIs were self-explanatory and aesthetically pleasing.

VIII. Project Management

A. *Schedule*

Most of our individual work had been done by the first week of May. There were several tasks added such as Arduino Uno setup to account for design changes, and we extended the timeline for some tasks based on the progress. We worked on end-to-end testing until the week of the final report deadline. For a more detailed schedule, please refer to the Gantt Chart (Fig. 7, page 12).

B. *Team Member Responsibilities*

Tasks for team members were distributed based on each member's specialties. Nina was responsible for the database, speech recognition, and their integration with the system. Lisa built the NLP system and created the staff-side UI. Shiyi programmed the distance sensors and the Arduino, and created the customer-side UI. All members completed the unit tests for the subsystems we were in charge of; the team worked on overall integration and end-to-end testing together.

The distribution of work remained the same as the design report documented. However, instead of infrared sensors and RPi4, Shiyi used distance sensors and Arduino Uno as the main components.

C. *Bill of Materials and Budget*

For cost breakdown, see Table VI, page 12.

D. *Risk Management*

Customer Detection: We replaced the infrared sensors with ultrasonic distance sensors halfway through the semester to improve the accuracy of detecting the presence of a customer and reduce detection errors. Previously, the infrared sensors would sometimes detect people passing by the kiosk, which could trigger false activations of the kiosk, and would sometimes detect a customer when there was actually no one, which had resulted in orders being unable to terminate. After switching to using the ultrasonic distance sensors, which produce more accurate readings, we can limit the detection range and activate the kiosk only when a person is likely sitting down on the chair, which reduces power waste and improves efficiency.

Noise reduction: No one on the team has previous experience with sophisticated signal processing, and the noise tolerance requirement for the project is considerably strict. To mitigate this, we switched to the current sitting setup, which places the microphone closer to the customer's mouth compared to the previous standing design.

NLP: The majority of the risks associated with the NLP system comes from complicated sentence structures involving multiple operations. When we made the decision to shift from non-stopping customer input reception to listening command-by-command, the risks no longer applied since the sentences became shorter and simpler. The risk of customers saying unexpected commands was also reduced by limiting the possible commands by providing instructions.

Cloud storage: Using an open-source cloud database runs the risk of experiencing extremely high access latency and/or issues with internet connection. An unresponsive system results in poor customer experience, which violates our use-case requirements. If we experience poor WiFi connectivity when demoing, we plan on switching to using personal hotspots of our phones to keep the connection up.

IX. Ethical Issues

Our system runs the risk of compromising user information. While our expectation is that Redis is secured and that our database is only accessible to the staff, the database could still be vulnerable to attackers. If an attacker hacks into the database, they will have access to all order information and be able to change any data. This could expose the customers' personal preferences, money-spending habits, and allergy concerns. The order information could also be maliciously changed after checkout. In addition, if we were to add a payment component to the system, the customers' credit card information could also be compromised.

Another potential risk is that there might be customers who check out and leave without paying, causing the restaurant to prepare dishes that are not paid for and not consumed. This not only harms the restaurant, but also causes other customers who genuinely want to eat to wait longer. To mitigate this, the kiosk should be installed in an area with surveillance cameras, and a system to monitor whether orders have been paid for should be added.

X. Related Work

There has been an abundance of speech recognition and natural language processing developments in other markets, and some researches and products are focusing on integrating

these technologies into the food service industry.

A research conducted in Dr. Mahalingam College of Engineering and Technology in India[6] proposed a speech ordering system that allows the customers to place orders through phone communications. The research and our project have the common goal of establishing a speech-controlled food ordering system, sending orders to the staff's end after orders are completed. The research uses Naïve Bayes classification to convert text input to different entry intents of menu item names and quantities, and proceed with the intent that has the highest probability.

Speechly[7] also has an API for taking customer orders by speech. The system can transcribe both real-time speech input or uploaded audio files, and recurrent neural networks are utilized to process the text.

XI. Summary

Our system works as expected under ideal conditions, where the customer is speaking loudly and clearly, the environment is generally noise-free, and the internet is stable and fast. However, we understand that this may not be the case in most restaurants. We noticed that our system became extremely slow when the internet was spotty, as both our speech recognition and our data storage systems depend on the internet. Moreover, the accuracy of our speech recognition system degrades significantly under noisy environments. While it won't record incorrect items, it requires the customer to repeat themselves multiple times in order for the system to correctly detect the item ordered.

A. Future work

Although we are not planning on working on this project beyond this semester, we do have some ideas about how the system can be improved.

One possible improvement is, instead of relying on the speech recognition libraries to determine the end of a speech, writing our own script for that purpose using a language that runs faster than Python, such as C.

Another potential improvement of our system is allowing menu customization. When the staff updates the menu, the system should automatically accommodate for the changes without requiring software engineers to manually modify the code for speech processing, the database, or the UIs.

B. Lessons Learned

One lesson we learned was that it is perfectly fine to make significant design changes when the current technology does not suffice for desired outcomes. Halfway through the project when subsystem integrations started, we discovered that the Raspberry Pi's operating system was incompatible with a number of Python libraries required for other subsystems. Changing to using an Arduino was a tough decision to make, since it required us to transfer all our files to a new device and make substantial alterations to our code. This ended up saving us a lot of trouble, because later we used even more Python libraries that would not be compatible with either the 32-bit or the 64-bit Raspberry Pi OS.

Glossary of Acronyms

API - Application Programming Interface
NLP - Natural Language Processing
RPi - Raspberry Pi
UI - User Interface

References

[1] T. Sun, "McDonald's touchscreens test positive for traces of feces, deadly bacteria," Fox News, 28-Nov-2018. [Online]. Available: https://www.foxnews.com/food-drink/mcdonalds-touchscreens-test-positive-for-traces-of-feces-deadly-bacteria.

[2] "Speed of service," QSR magazine, 2016. [Online]. Available: https://www.qsrmagazine.com/content/speed-service.

[3] Figure A3. dimensions of adult-sized wheelchairs. [Online]. Available: https://archive.ada.gov/1991standards/descript/reg3a/figA3ds.htm#:~:text=Lap%20height%20is%20shown%20as,8%20inches%20(205%20mm).

[4] W. M. To and A. Chung, "Noise in restaurants: Levels and Mathematical Model," Noise & health, 2014. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/25387532/.

[5] A. M. Raghavan, N. Lipschitz, J. T. Breen, R. N. Sami, and G. D. Kohlberg, "Visual speech recognition: Improving speech perception in noise through artificial intelligence2020," Otolaryngology--head and neck surgery : official journal of American Academy of Otolaryngology-Head and Neck Surgery, 2020. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/32453650/.

[6] Sabarinathan, Swathi, V., & Pandi, D.M. "Smart Ordering System using Speech Recognition," International Journal of Creative Research Thoughts, 2-Feb-2020. [Online]. Available: https://www.semanticscholar.org/paper/Smart-Ordering-System-using-Speech-Recognition-Sabarinathan-Swathi/115e45fe61345a91d71ac4f7e8baa42e211f69f5#references.

[7] Speechly. 2023. [Online]. Available: https://www.speechly.com/solutions/quick-service-restaurants

	Items							
Name	Microphone Isolation Shield	Cardioid Directional USB Condenser Microphone	Ultrasonic Sensor	Alligator Clips	Raspberry Pi kit	Raspberry Pi 4	Arduino
Model Number	L(29.9"x12.8")	24 bit/96 kHz Bumblebee II	HC-SR04	N/A	/	Model B 8GB RAM	Uno Rev3
Manufacturer	Moukey	Neat	WWZMDiB	Romeda	SunFounder	Raspberry Pi	Smart Projects
Unit Price	$49.99	$59.95	$6.99 for two	$5.99	$13.99	From ECE inventory	From ECE inventory
Quantity	1	1	1	1	1	1	1
Is being used	Yes	Yes	Yes	Yes	No	No	Yes
Acquired near the end of the semester	No	No	Yes	Yes	No	No	Yes
Total (tax included)	$139.59						

TABLE VI. BILL OF MATERIALS

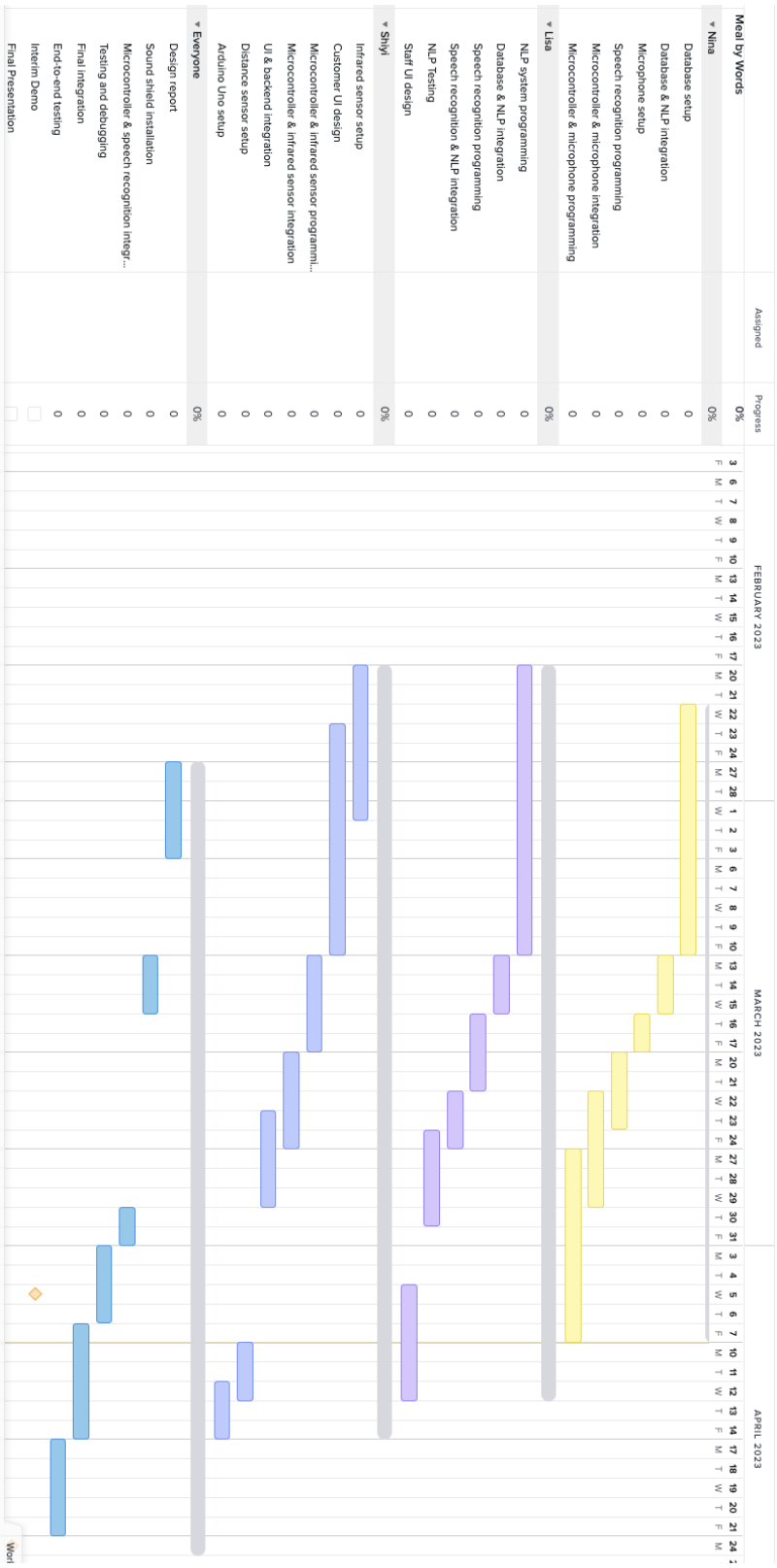


Fig. 7. Gantt Chart.