EchoBudget

Lynn Sun, Yuxuan Xiao, and Yixin Yang

Department of Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—The system is a web-based application that provides money tracking functionalities with audio input. Traditional money tracking apps are not accessible for visually impaired people and the elderly due to lack of visual aids and complex user interfaces. To meet their needs for money tracking, we propose to implement an application that records, shows, and analyzes spending solely with audio input and output.**

***Index Terms*—Design, natural language processing, noise reduction, signal processing, speech recognition, text-to-speech conversion, user interface, web application.**

I. Introduction

WITH the development of social welfare and humanitarianism, the living standard has been greatly enhanced for visually impaired and elderly people. However, their need for accessible money tracking methods is not satisfied because the traditional apps require the user to know what functionality each component has in a complex UI, which is too sophisticated for visually impaired and elderly people to manipulate. The voice assistants in smartphones are not capable of converting voice input to a specific command interpretable by a traditional money tracking app. In addition, many visually impaired and elderly people do not possess smartphones and thus have no access to the traditional apps.

A portable device with an app supporting audio-command conversion and content reader can address this issue. Our project is to develop a voice-controlled money tracking app on a low-cost device. The system can parse the audio input into parameterized commands and provide audio output of the generated contents to best satisfy our users' need for recording their expenses without visual help.

II. Use-Case Requirements

Available Operations: Our system EchoBudget would support common functionalities of money management apps in the market. Users are able to create, edit, and remove entries to record their daily expenses. In addition, users could ask our system to generate spending reports to acquire more information about their spending habits. While other applications require users to input all the information by tabbing, our system could be controlled fully with voice commands. There would be a button on each page and users could push the button and begin to give their instructions. Our system would then complete the tasks assigned by users.

Voice Input: Users would communicate with our system like sending voice messages through their phones. Therefore, when users are holding our system and talking to it, our system would be able to receive the voice input. According to Gitnux, the average adult female arm length is about 27-31 inches (68.58-78.74 cm) and the average adult male arm length is about 28-34 inches (71.12-86.36 cm)[5]. Therefore, our system should be able to receive inputs with distances from 0 to 100cm. According to the Centers for Disease Control and Prevention, the average sound level for normal conversation is about 60 dB, and that for whisper is about 30 dB. Therefore, our system should work well with human voice ranges from 30 to 70 dB. Since we expect our users to use our system in places including restaurants and supermarkets, our system should also work in a relatively noisy environment.

Latency: According to WebsiteBuilderExpert, 25% of users would leave if the websites did not load within 4 seconds[7]. Therefore, we would expect our websites to give any update within 4 seconds. To be more specific, our system would create, edit, and delete the entry within 4 seconds. Additionally, our system would generate a report within 4 seconds.

Portability: It will be inconvenient for customers to bring a large and heavy device with them. Thus, the size of our system is designed to mimic the size of a mobile phone (iPhone 13 pro) and the weight of our system would be similar to that of an iPad Air (500g).

Battery Life: With the monitor on, our device should be able to operate for at least one hour. Since customers would only use our device for money management, one hour of operation would be enough to record daily spending. When the monitor is off, we would try to minimize the energy consumption to allow customers to charge the device less often and we would expect our system to operate for at least 24 hours with the monitor off.

Accessibility: Our system is designed to be friendly to both people with visual impairment and elder people. To enhance accessibility for visually impaired individuals, our system will offer active voice responses in accordance with their commands. For instance, when our system successfully completes an action, such as creating, modifying, or removing an entry, the system will audibly confirm with phrases like "entry created/modified/removed". This confirms would let users realize that their actions are successful without looking at the monitor. Furthermore, when users navigate to the record page, the system will audibly announce the details of each entry displayed. For example, "entry number 12: apple, categorized under food, priced at 5 dollars." For the user interface, we aim to make it concise and clear. We would expect the elderly people to master our system within one day. Moreover, the button used to initiate commands is designed to be large and consistently positioned on every page. This would be friendly for visually impaired groups because they can easily find the button.

III. Architecture and/or Principle of Operation

EchoBudget is a device that executes a web application on Raspberry Pi and allows users to interact with it via a touchscreen and a microphone. The design could be distributed into a hardware section, a signal processing section that could be further split into speech recognition and NLP processes, and a software section. The hardware and signal processing sections transform interactions in the physical world into data that can be interpreted via software. The software section analyzes the data and provides corresponding performances based on user requirements.

A. Hardware

Our system performs all functionalities on a Raspberry Pi 4 with a touchscreen monitor and a USB microphone. Customers could interact with the system via finger tapping and audio. The primary home page user interface would be displayed on the monitor screen, and a “help” command is recommended for users to receive an initial audio-driven tutorial that provides them with a basic introduction to the web application through the built-in speaker. Customers could use the microphone to deliver voice command inputs and the audio signals would be transmitted to the speech recognition subsystem.

A lightweight and portable power bank is connected to the Raspberry Pi as the power supply when using the device in outside environment. The power bank could guarantee a 4-hour battery life when the touchscreen is on.

B. Speech Recognition

The speech recognition pipeline is designed to convert the voice commands from the customers to text strings that would then be fed to the natural language processing models. The audio input stream goes through a noise reduction algorithm to guarantee that the voice commands can be effectively distinguished under environments with volume from 30 to 70 dB. The treated stream is then delivered to a trained speech recognition model and outputs as text strings.

C. Natural Language Processing

Once the audio is successfully converted to text strings, two natural language processing models will be used to parse the strings, extract key information including command words, item names, price numbers, entry numbers, or dates, and classify the distinguished item names to different categories based on the command inputted.

Command Parsing: The converted text string will be passed to two command-parsing models instead of one to parse all information and send it to each web page. An action verb is essential for all received text strings, and potential item names, price numbers, item numbers, or date information will also be required for different pages. The corresponding web page would be rendered based on the information captured. If the models fail to recognize the action verb or corresponding parameter words, it will send a notification to the customers so that they can repeat their commands. Specifically, the model would let the customers confirm whether the parsed item name and price number are correct or not.

Classification: There are six standard categories for the entries including food, housing, necessities, entertainment, transportation, and others. These categories should be able to cover most of the daily spending classes, and the “others” choice could always serve as a backup choice if no other categories are appropriate. After the customer enters the item name and price, the new entry will automatically be assigned a category based on its name. The customers could modify the classification if needed by providing new commands.

D. Web Application

The Django framework web application is the major interface for the users to interact with the systems. The application allows customers to input new expenditure entries to the database, view existing entries, modify or remove entries if needed, and learn the spending statistics via financial reports. To benefit the visually impaired groups, a large button is designed on the right-hand side of all pages of the application, and the customers could give their voice commands by tapping the button. The customers could also speak up after a “start recording” audio notification, and we changed the original “end recording” button clicking to an automatic stop after 6 seconds.

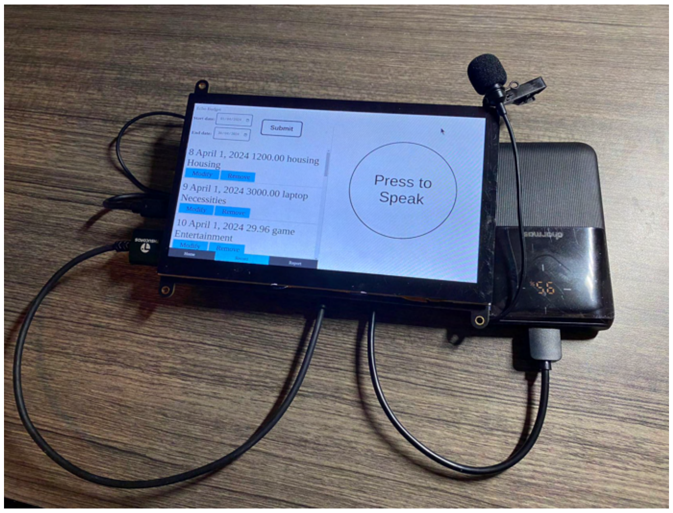
Based on the received commands, different pages of the web application would be rendered. Action verb “Enter” will render the Enter Item page, and an item name and cost value are expected to be parsed as the parameters for a new entry. An audio notification that repeats the entered information will be played via the speaker, and the customers could use the “Confirm” command to save the entry to the database. If any field is incorrect, a new voice command with action word “Change” and the corresponding field name and expected value should be given.

Customers could be able to view their entries via the command “Get” or “Get entries”. The page that displays entries for the current month will be rendered and each entry will be assigned an id number. All listed entries and their information will be read out during the page is rendering. The customers could use “Modify” with the id number to adjust a specific entry. The modification page for that entry should then be rendered, and the customers could make changes via the same approach as the Entry Enter page adjustment operations. Command “Remove” with the id number parameter of a specific entry is used when the customer wants to delete an existing entry.

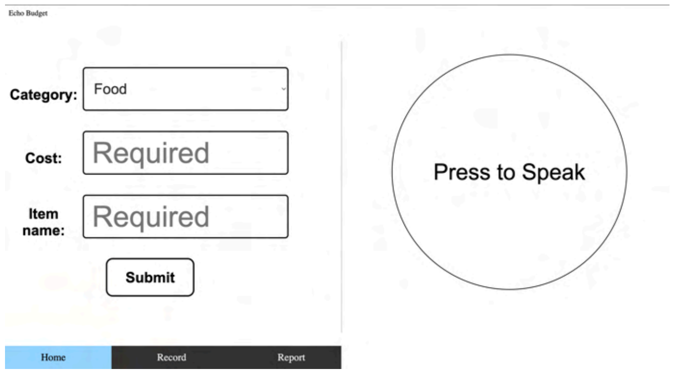
Command “Generate”, or “Generate report” refers to the Financial Report page, and the customer could use the command “Generate report from start month-start year to end month-end year” with the corresponding date information parameter to get the report of that time range.

Customers could always use “Help” to learn about available action words.

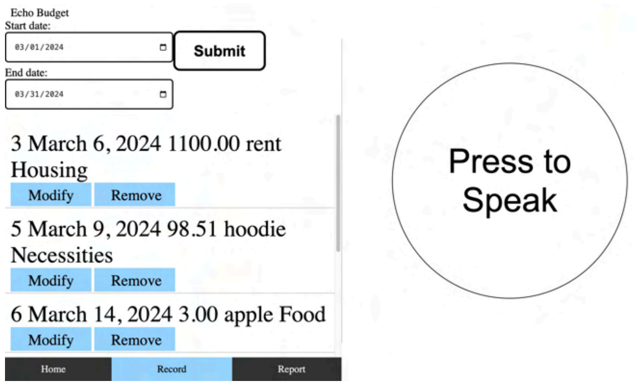
An additional set of ordinary UI is designed for customers to perform all operations using touch and is displayed on the left-hand side of the screen.



(a)

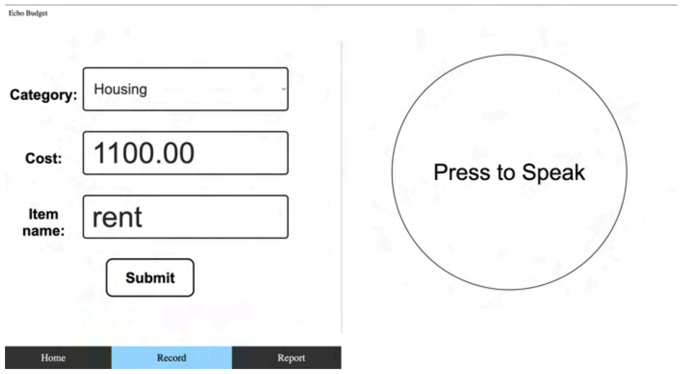


(b) Home Page



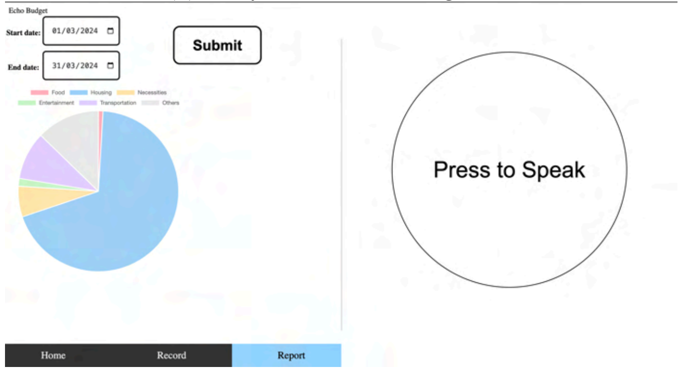
3 March 6, 2024 1100.00 rent Housing
Modify Remove
5 March 9, 2024 98.51 hoodie Necessities
Modify Remove
6 March 14, 2024 3.00 apple Food
Modify Remove

(c) Entries Page



Category: Housing
Cost: 1100.00
Item name: rent

(d) Entry Modification Page



(e) Financial Report Page

Fig. 1. Overall system. (a) Photo. (b)-(e) Web App User Interface.

TABLE I. AVAILABLE VOICE COMMANDS

Voice Command	***Action***	***Response***
“Enter {item name} for {price} numbers”	Parse item information to the Entry form	“Entering {item name} for {price} dollars, please confirm”
“Confirm”	Submit or update the information of the current entry	“Entry created/updated”
“Get entries (from {start month, start year} to {end month, end year})”	Get entries created in the given time range	Read all entries with id, item name, price, and category
“Modify No. {id}”	Enter the entry modification page of the specific entry	“Modifying entry no. {id}, you can change the category, item name, or price by saying ‘change’ {field} to {new value}”
“Change {field} to {new value}	Change the value in the specific field to the new value	“Changing {field} to {new value}, please confirm”
“Remove No. {id}”	Remove the specific entry from database	“Entry removed”
“Generate report (from {start month, start year} to {end month, end year})”	Generate financial report for entries created in the given time range	Read the percentage of each category

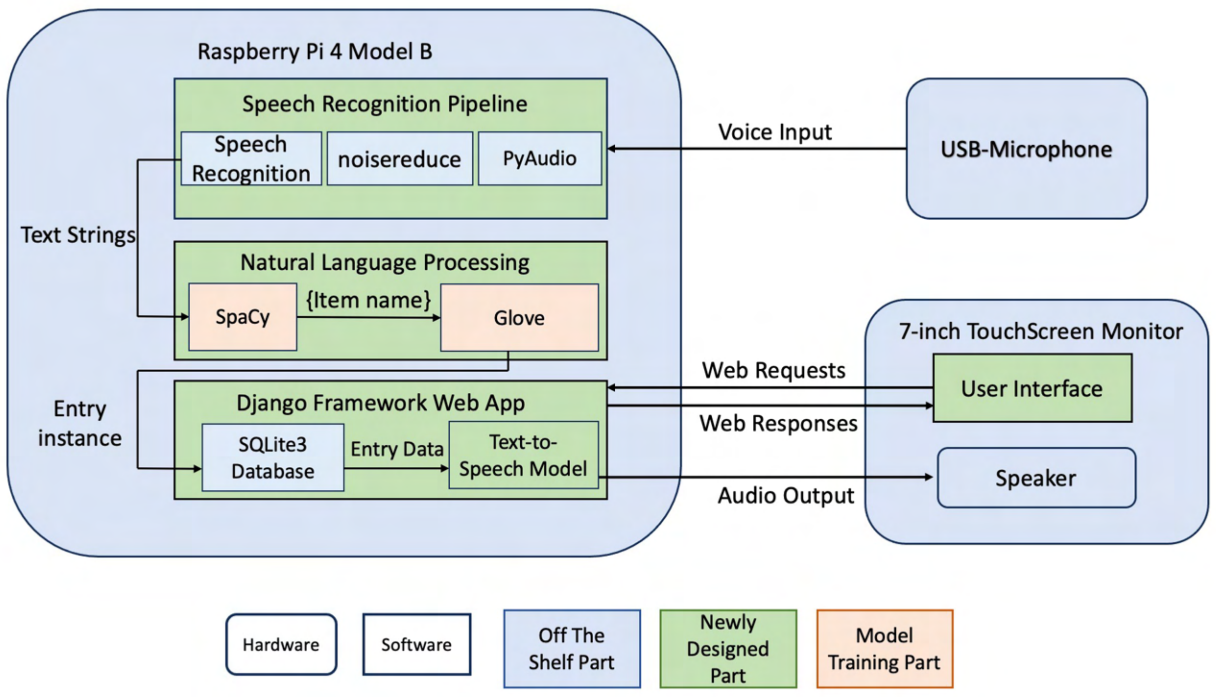


Fig. 2. Block diagram of the system.

IV. Design Requirements

Speech Recognition: According to Statista, the error rate of speech-to-text technology of Amazon, Microsoft, and Google Video is 18.42%, 16.51%, and 15.82%[8]. Therefore, we expect our system to have less than a 20% error rate for the whole sentence. The important information for our system is the action verbs (e.g. generate the report, create, delete, etc.), item names, and prices. We would expect at least 90% accuracy for these important words to ensure our system works smoothly.

Voice Input Parsing: Our task for this part would be to extract important information (including action verbs, item names, prices, etc.) from the whole sentence. The action verbs determine the main task our system would handle and help us navigate to the correct page. Item names and prices would be the primary information we need for each of the entries, and the item names would be the inputs for categorization. Since the information we get from this part would directly influence what would be displayed in the web application, we would expect the accuracy for this part to be at least 95%.

Item Categorization: We would have six basic categories: food, housing, necessities, entertainment, transportation, and others. It would be burdensome if the auto-categorization fails too often, so we would ensure at least 90% accuracy. We would not allow customers to add personalized categories. This is because the newly created category would have negligible training datasets, which would substantially decrease the accuracy of classification.

Power Bank: We would use the battery life of mobile phones to help estimate the size of our power bank. The battery capacity of iPhone 15 Pro Max would be 4441 mAh[9]. The battery test shows that a fully charged iPhone could supports 11 hours and 28 minutes of social media browsing. Using our system may require more power than browsing social media, because our system would consistently provide audio output. In addition, raspberry pi would heat up when running the application, which will consume additional power than a normal iPhone. Therefore, we would choose to try a power bank with 10000 mAh capacity (about 2.2 times as the size of iPhone power bank). With this power bank, we would expect our system to be active at least two hour every day and the battery life would be at least 24 hours.

Active Interaction: We would want our system gives constant feedback to the users so that they would know the system is actually processing their request. For the voice input part, we would have voice notifications when button is pressed to inform the users that the following commands would be recorded. Additionally, we would repeat the information provided by the users and let the users confirm what they said. For example, when adding new entry, if the user says “enter apple for five dollars”. Our system would have audio responses like “do you want to enter apple categorized in food for five dollars?”.

UI Design: For every page of our system, we would divide the page into two parts. The left part would handle all the functionality of a money tracker app, including editing, displaying, and deleting the entries and showing the reports we generated. The right side would always be the button for voice input, which ensures that users could easily use voice commands on every page.

Financial Reports: We would generate reports for a given periods. One would be the pie charts, which show how a user’s spending is divided among different categories.

Default Display: Users could provide a time range for the record page (where we display all the entry) and for the report page (where we display the report). However, if the user does not provide the time range, we would display the entry or report for current month by default.

V. Design Trade Studies

Based on our use-case and design requirements, 5 crucial choices are made and the trade-offs are analyzed.

A. Hardware Selection

The hardware that carries the signal processing modules and web application is crucial for the general product. An embedded system consisting of RPi 4 and a touchscreen monitor is preferred over laptops and smartphones in our design. The major factor that leads to this decision, other than a laptop, is portability. Customers should be able to use our systems in various scenarios, with or without the condition of using laptops. For example, one of the expected use cases would be inputting entries at grocery stores. Therefore, the hardware should be lightweight and portable, allowing the product to fit in hand and to satisfy the use case requirements.

On the other hand, although a smartphone could be an appropriate device that guarantees portability, we decided not to choose it as the carrier hardware due to the target customer group's specific requirements. The target user group of our design is visually impaired people, and the majority of this group either do not have a phone or simply use phones with basic functions. As a result, a smartphone application fails to provide the service to a large group of our customers.

Based on the research and discussions, a portable embedded system implemented using Raspberry Pi and a monitor is finalized as the hardware selection for our design. The total weight of the design is about 500 grams (including RPi, monitor, USB microphone, and power bank), and customers can interact with the system via the touchscreen.

B. Speech Recognition

As the primary module of the signal processing system, the speech recognition pipeline should convert the voice input from the customers to text strings for NLP process in environments under 30 to 70 dB volume. Therefore, three Python libraries are used to construct the pipeline script and guarantee a high performance of the speech recognition module. PyAudio is selected to convert audio signals to streaming data because this library simplifies the audio-capturing process, making it ideal for microphone and speaker interfaces. Although there are several libraries that could handle noise reduction such as pydub and PyAudio, the noisereduce library is selected because it is designed specifically for identifying and reducing background noise in audio signals, which is crucial for the speech recognition process under high-volume environments. The library SpeechRecognition is chosen for the actual speech-to-text component in the pipeline because it could support multiple engines and APIs, which provides flexibility and versatility. Therefore, backup scripts implemented by different engines could be utilized when the primary engine fails.

C. Command Matching

For parsing the whole sentences, we are deciding between two models: spaCy and NLTK. We choose to use spaCy for the following reasons.

Firstly, spaCy provides out-of-the-box support for Named Entity Recognition (NER) which is more advanced and accurate compared to NLTK. This feature of spaCy can effortlessly identify and classify proper nouns, including product names and monetary values, directly from the text, which is essential for our use case.

Moreover, spaCy's processing pipeline is designed for production use and can handle large volumes of text rapidly, making it highly scalable and efficient for real-time applications. NLTK, while powerful for academic and research purposes, is not optimized for speed and may not perform as well in a production environment where quick processing is critical.

Another advantage of spaCy is its superior part-of-speech tagging and dependency parsing which are crucial for understanding the grammatical structure of sentences. This allows for a more accurate extraction of action verbs to the items and prices mentioned, providing a clearer understanding of the sentence's intent.

D. Item Categorization

In the item categorization process, our main goal is to assign one of the predefined categories to the item name obtained from the first half of the NLP pipeline. There are many text classification algorithms such as Random Forests (multiple Decision Trees), Logistic Regression, and Decision Trees. However, these algorithms are used to handle long texts with many features, while we only aim to parse single words or short phrases. Therefore, models that implement the above algorithms are not under consideration of this task.

Word vector representation is what we can make use of. It is a method of converting words into high-dimensional vectors so that similar words are close to each other in the vector space. The two most popular word vector representation methods are Word2Vec and GloVe, both of which can potentially be customized for our word classification task. Although Word2Vec and GloVe differ in how they train the model, they are similar in the result, so there is little difference for us using one or the other[2]. For convenience, we initially adopted the pre-trained Google News Word2Vec model, which contains 300-dimensional vectors for 3 million words and phrases and is readily available for download and use through Python's gensim library. GloVe, on the other hand, provided an alternative since the performance of Word2Vec was not ideal.

E. Database

Our project is a local app with a single user and does not contain complex user data to be stored in the database. Therefore, we will be using the SQLite that comes with the Django framework. SQLite is easy to set up and stores data in a single file, making it handy to use for a standalone application like ours. Other databases supported by Django provide features, such as authentication, distributed systems, and duplication of databases, which are unnecessary to our project.

VI. System Implementation

The whole system of our project runs on a Raspberry Pi 4. A touchscreen monitor that displays the web application UI and a USB microphone for voice input is connected to the RPi.

A. Hardware Setup

There are four major hardware components in the design: a RPi 4, a 7-inch RPi touchscreen monitor with speaker, a USB microphone, and a power bank. The RPi could be mounted on the back of the touchscreen with 4 screws. A 3-pin GPIO cable is connected to RPi pin 2 and pin 4(5V power) via the red wires and pin 6(Ground) via the black wire to output power from the RPi to the monitor. Thus a separate power for the monitor is not required and portability is guaranteed. To enable the touch feature of the monitor, a microUSB to USB-A cable between the USB port of the RPi and the 5V-touch port on the monitor is connected. An FPC cable with HDMI connector is also needed between the RPi and the monitor to transmit audio and video signals. The device could be charged either through a receptacle or a portable charger. A 10000mAh 5V/3A power bank is chosen for portable charging. The USB microphone is connected to the RPi directly.

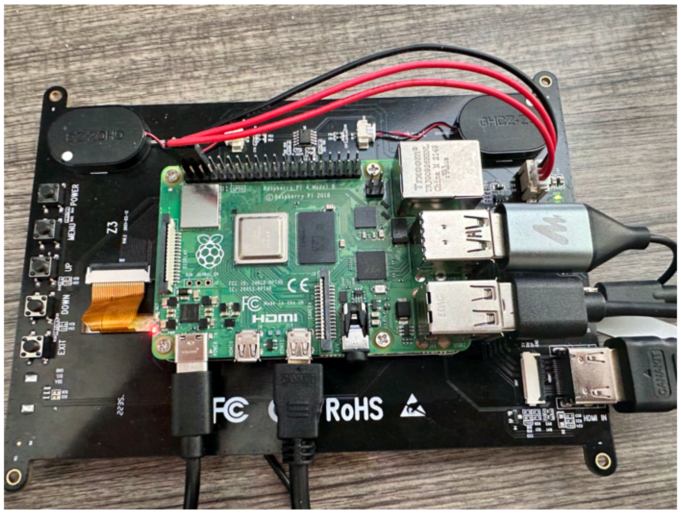


Fig. 3. RPi, touchscreen monitor, and USB-microphone connection

B. Speech Recognition

To convert the voice inputs given by the users to text strings that could be fed to NLP models, a speech recognition pipeline script is implemented using several Python libraries. An audio stream is initialized once a new recording session starts, and corresponding data is continuously written using the methods in PyAudio library. The recorded data is then passed on to the noise reduction module that is implemented via the noisereduce library.

After eliminating possible environmental noise, the modified data would be transferred to the speech recognition module built based on the SpeechRecognition library. This library supports different speech recognition engines and APIs that could work both offline and online. The output will be concatenated as strings that are sent to the NLP models once the conversion process is completed.

C. Text Parsing

The input of the NLP models would be the script from speech recognition. Our initial guide for customers to learn how to interact with our system would require them to use imperative sentences. Therefore, we would expect all the commands given by the customers to be imperative sentences.

Based on our requirements and the structure of imperative sentences, for most of the time, there should be only one verb in the sentence and that verb would be the key action users want our system to proceed. Therefore, we would load the existing spaCy model and use POS (Part-of-speech tagging) figures of spaCy to identify the verb.

It would be harder for us to extract item names and prices. The features we would use here is Named Entity Recognition. For this part, we would first train our own spaCy model. We would want our model to learn to assign ITEM label to items in our 6 categories. For example, “apple” (categorized in food), “laptop” (categorized in necessities), and “rent” (categorized in housing) should all have label ITEM in our model. For the price, we want our model to assign PRICE label to prices in different format. That is to say, we want both “xx dollars xx cents” and “$xx.xx” would have a label PRICE. For the time range, we want our model to assign DATE label to both a time range and a single month. For instance, both “month1, year1 to month2, year2” and “month1, year1” would be assigned a label DATE.

D. Item Classification

After parsing the required components from the text, we use a pre-trained GloVe model for the item categorization process. The model is downloaded and customized with the NumPy library in Python. An initial dataset is created mapping item names into categories. Given a new item name, our algorithm will calculate the similarity of the new item name and each of the item names in the dataset, obtaining a normalized similarity score for each category. The category with the highest score is assigned to the new item name. The user has chances to modify the category of a spending, and once confirmed, the dataset will be updated with the item name and the new category for future comparisons.

E. Web App UI & Text-to-Speech

The web application user interface is split into a GUI section and a VUI section that satisfies user requirements for different customer groups. HTML pages could be rendered via both voice commands and button clicking. Customers could submit new entries through the Entry form and a new instance of the Entry model will be created and saved to the database if confirmed. Similarly, entries could be modified by extracting the model instance data from the database and saving the updated instance back. To display the monthly entry lists and financial reports, the application would retrieve and filter Entry data based on the time field and feed the data to specific command handler functions.

The major component of the VUI section is a large button on all pages that enables and disables the voice input functionality. Customers interact with the web application through voice commands in this section, and the system will respond accordingly. The user could start giving voice commands after pressing the button and hearing a "start recording" notification.

Action "Enter", "Get entries", and "Generate report" could be achieved by simply sending an HTTP GET or POST request with no extra information despite those in the current command. Therefore, the corresponding action functions that render the pages could collect the required request information and render the page. However, other actions such as "Confirm" and "Modify" would require additional information from the previous page, which could not be achieved through the voice command. Since Django framework does not allow direct access from views.py (the file where most backend algorithms are) to the currently displayed static HTML page and the data posted on the page, sessions are introduced to store the necessary information from the last page to the page rendering by the new request. For example, the session could store the newly entered item information in the request after an "Enter" command which only parsed the item name and price temporarily. In this case, after the voice command "confirm", which is equivalent to a manually submitted new entry creation form, is given, the request could still keep the entry information and save it successfully to the database.

Each voice input is assisted with a corresponding audio response, so after parsing the key parameters from the audio input and processing the command, the response information strings will fill the corresponding pre-generated message templates that are stored in static text files. The completed response texts are then generated based on the requested information and fed to a text-to-speech converter implemented using gTTS. A new thread will be created to generate the audio file, deliver the output audio to the customer via the built-in speaker of the touchscreen monitor, and remove the audio file while the page is rendering in the main thread.

The GUI section is mainly constructed using HTML elements and JavaScript components. Customers could manually create or update entries through HTML forms and could view the entry lists and responsive financial reports that include a pie chart.

F. Data Storage & Management

The database used in the design is the SQLite database in the Django framework and the key data that need to be stored in the database are Expense, Date, and Category.

The Django model for Expense is defined in models.py and consists of an item name character field, a price number decimal field, a category foreign key field, and a date field that indicates the creation time of the entry. Item name and price number are parsed from user input, while category and date are determined or assigned by scripts. Once a new entry is created, the instance will be stored in the database table for the Expense model. To update specific fields of an existing Expense, an ID number that corresponds to the ID of the entry (which is assigned automatically by Django) should be entered and the entry could be found in the database. Modifiable fields for the Expense model are item name, price number, and category. When a financial report is requested, expenses created in the given month are extracted via the date field. These entries are distributed into 6 arrays by categories, and the total price spent for each category is calculated. The statistics are fed to chart components and the complete report page is generated.

The Django model for Category contains the category name. In the migration file, 6 Category objects are automatically created in the database, serving as the 6 predefined categories. In the ExpenseForm, the Category field is a dropdown list of those 6 categories. There is also a DateSelectionForm that can be used to select the time range to view entry records and to generate the report.

VII. Test, Verification and Validation

We evaluate the functionality and performance of the system through both unit tests and integration tests. These tests are conducted during the implementation or after the integration of all parts.

A. Results for Speech Recognition Accuracy

As the primary module that collects the user input and translates to text, the audio-to-text accuracy plays a crucial role in the whole system performance. The speech-to-text conversion done by the SpeechRecognition library is expected to reach an accuracy of 90%.

To conduct unit testing, we spoke the common phrases used in the product to the microphone and counted the number of words that were translated incorrectly. Then the accuracy is calculated accordingly. 150 different voice commands that cover all available action verbs are tested with different item name, price number, id number, and date time information. The general speech recognition accuracy among these test cases is about 98.3%.

Table II. Speech Recognition Error Cases

Test command	*Translated Text*	*Accuracy*
"Spend 15 dollars on eggs"	"Spend 15 dollars on X"	80%
"Enter bread for 5 dollars"	"Enterprise for 5 dollars"	60%
"Buy pie for 3 dollars"	"By Pi for 3 dollars"	60%
"Enter healthcare for 100 dollars"	"And her healthcare for 100 dollars"	80%
"Enter laptop for 1500 dollars"	"Enter laptop for 1500"	80%
"Enter music for 19 dollars"	"Enter music for 19"	80%

After analyzing the test cases, we figured out that the SpeechRecognition library may fail to convert money into the expected pattern with its unit attached i.e. ${price}, which would lead to further NLP edge case handlers. As for misunderstanding of action verbs and item names, reasons including pronunciation and accents should be taken into consideration, and the current error handling method is simply asking the user to repeat the command with a response of "Try again".

B. *Tests for text-to-command parsing*

The input for spaCy model is the text string of commands. spaCy would parse five different categories of words and phrases from these text strings. We fed 150 different commands into spaCy for testing.

spaCy can recognize the action verbs in every command, reaching 100% accuracy. It could also identify cardinal number, and date with 100% accuracy. However, it may encounter some edge cases when parsing items and price. For example, when we want it to parse "apple pie" from "enter apple pie for 5 dollars", spaCy would only parse "apple". Tests for Latency

We will test the total time spent waiting for the system to parse the command. From the time the user stops recording to when the parsing result is reflected on the page, the elapsed time should be less than 3 seconds for 90% of the trials. We will run 10 trials for this test.

TABLE III. TEXT-TO-COMMAND ACCURACY

Text Label	***Example***	***Accuracy***
ACTION / VERB	Enter/buy/spend, modify/edit, remove/delete, get entries/view entries, generate report	80%
ITEM	Change item name to apples (item), enter housing (item) for 1200 dollars	96%
CARDINAL	Modify/delete entry number **12** (cardinal), modify/delete entry **12** (cardinal)	100%
PRICE	"Enter laptop for 800 dollars (price), change price to $6 (price)	96%
DATE	Get entries for **May 2024** (date), generate report from **January 2023 to March 2023.**	100%

C. *Results for Text Input Functionalities*

The app is expected to function correctly as a normal money tracking tool supporting text inputs. It should serve basic functions discussed in the Architecture and/or Principle of Operation section and behave as described in Table I. The tests include but are not limited to the following: the pages for entry input, expense list, and financial report contain all components as shown in the UI design (Figure 1(b)-(e)); upon the submission of an entry input, the information is stored into the database and can be viewed in the expense list; users can modify or delete any spending entry in the expense list, and the change will be made correspondingly in the database and reflected in the expense list; the financial report calculates the total amount spent and the amount spent in each category for the selected time period correctly.

These basic functionalities were tested immediately after their implementation so that later audio input functionalities could be built upon these basic functions.

D. *Results for Item Classification Accuracy*

If the voice command is to enter a new entry, we will then classify the item name into one of the categories using the GloVe model. The item name is limited to a single word and is converted to all lower-case letters to match words in the GloVe model. The classification process should reach an accuracy of 90%. Among the 20 randomly selected and labeled item names, 18 were correctly classified, reaching the expected accuracy of 90%. The incorrectly classified words belong to the expected category of transportation, probably due to insufficiency of words under the transportation category in the training dataset.

TABLE IV. ITEM CLASSIFICATION ACCURACY

Item Name	***Expected Category***	***Actual Category***
taco	Food	Food
soup	Food	Food
sushi	Food	Food
avocado	Food	Food
lease	Housing	Housing
electricity	Housing	Housing
tv	Entertainment	Entertainment
laundry	Necessities	Necessities
shampoo	Necessities	Necessities
tissue	Necessities	Necessities
cosmetics	Necessities	Necessities
music	Entertainment	Entertainment
shoes	Necessities	Necessities
concert	Entertainment	Entertainment
flight	*Transportation*	*Entertainment*
parking	Housing	Housing
car	*Transportation*	*Housing*
train	Transportation	Transportation
haircut	Others	Others
decoration	Others	Others
Total number of items: 20		
Number of correctly classified items: 18		

E. *Results for End-to-End Accuracy*

For the audio command processing as an integral part, we are aiming to achieve an overall accuracy of 90%. That is, 9 out of 10 audio commands should be interpreted and executed without mistakes in order for the product to be useful and labor-saving for our users. Among the 10 audio commands we tested, only 1 failed to generate the expected output: audio command "Enter cake for 12 dollars" was converted to text "Enter a cake for $12" and the parsed item name was "a cake", classified to "Others" category instead of the expected "Food" category. The remaining 9 commands triggered correct responses, resulting in an end-to-end accuracy of 90%.

F. *Results for Latency*

We tested the total time spent waiting for the system to parse the command. From the time the user stops recording to when the parsing result is reflected on the page, the elapsed time should be less than 4 seconds on average. We ran 10 trials for this test. The actual average latency is 4.52 seconds, 13% greater than the expected value. This is because we ended up fixing the recording time to 6 seconds and the time spent waiting for the recording to stop counts most for the latency.

G. *Results for Noise Reduction*

Since our product should function well in noisy environments such as supermarkets and restaurants, we repeated the test for accuracy with surrounding noise of 70dB. We conducted the end-to-end accuracy test in Giant Eagle supermarket, which has a constant ambient noise of between 70dB and 80dB. The test is expected to meet the same requirements as the test for accuracy, and the result is indeed the same as that of the end-to-end accuracy test (90% accuracy).

H. *Results for Battery Life*

We power the battery with the power bank only and keep using the app for 1 hour to test if the battery can support the system for 1 hour with the monitor on. We will also shut down the monitor for sleep mode and see if the battery can last for 24 hours without charging.

It turned out that the power only dropped from 100% to 77% after one hour of intense usage, so the battery life is approximately 4 hours, which is much better than our expectation. The power further dropped from 77% to 72% with the monitor off for 24 hours. Our initial calculation of the required battery capacity took efficiency loss into consideration, leading to an overestimation of the capacity needed. We could replace the power bank with a 6,000mAh one to reduce the weight of our device, but this is not a big concern for now.

I. *Results for Portability*

The portability of the whole device is a key feature because the expected use cases of this product include shopping in grocery stores and malls. The system should be lightweight enough and small in size to be carried around by the user. The expected weight of our product is 500 grams. The key contributors to the total weight of the device are the touchscreen monitor, Raspberry Pi, and the power bank. While we did not have many choices for the monitor and the RPi, we managed to choose the most lightweight 10,000mAh power bank without sacrificing the battery life. The resulting weight of the device is exactly 500 grams including the power bank, satisfying our requirement.

J. *Results for User Experience*

We invited 5 volunteers to use our product without our excessive interference. After they thoroughly explored our product, they were invited to provide any feedback regarding the conciseness of the UI, the complexity level of their interaction with the app, and the overall experience. This guided us to make potential improvements or adjustments before the final demonstration.

During the test, we found several bugs in voice command parsing where we did not catch or report the error upon failure. For example, if the user did not say "dollars" when talking about price, the number would not be parsed as a monetary value and would cause the web page to crash. We fixed several edge cases causing similar issues.

Based on the volunteers' feedback, we identified some situations where more guidance was needed by the user. For example, we initially introduced all our commands in the response to the "help" command, but the user found it too long to remember those commands at once. We then kept three basic commands in the "help" menu and incorporated the remaining commands to the pages where they could be triggered. The user will be instructed to "modify" or "remove" a specific entry after they visit the record page by saying "get entries", for example. By spreading the instructions to the corresponding pages, the user can get a more coherent experience when exploring those pages.

Moreover, we made modifications to our UI by displaying bigger font size and unifying the voice command and button name (like changing "delete" buttons to "remove" to match the command "remove entry number x").

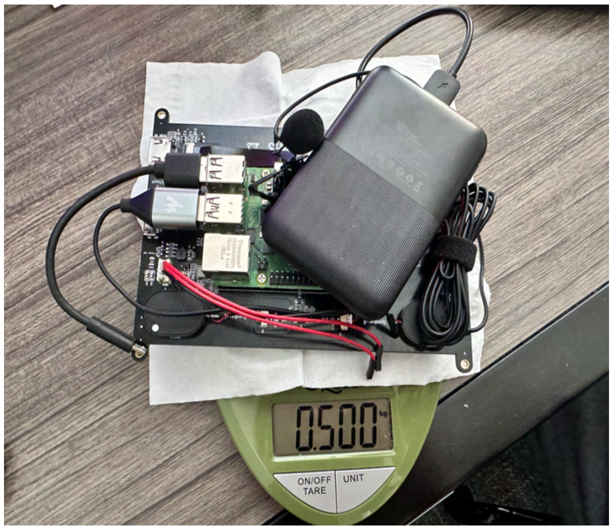


Fig. 4. Device weight measurement.

VIII. Project Management

A. *Schedule*

Please see the Gantt Chart (Figure 5) for our detailed schedule.

B. *Team Member Responsibilities*

The project is roughly divided into three parts and distributed to team members.

Lynn is responsible for UI design, graphic works, speech recognition, and text-to-speech conversion. Yixin researches and trains the spaCy model for parsing commands and parameters from the text. Yuxuan develops the web application and customizes the GloVe model for word classification. Everyone in the team participates in hardware setup, component integrations, and testing process. We worked together on certain tasks that require extra effort.

C. *Bill of Materials and Budget*

See Table V for a complete list of materials and costs.

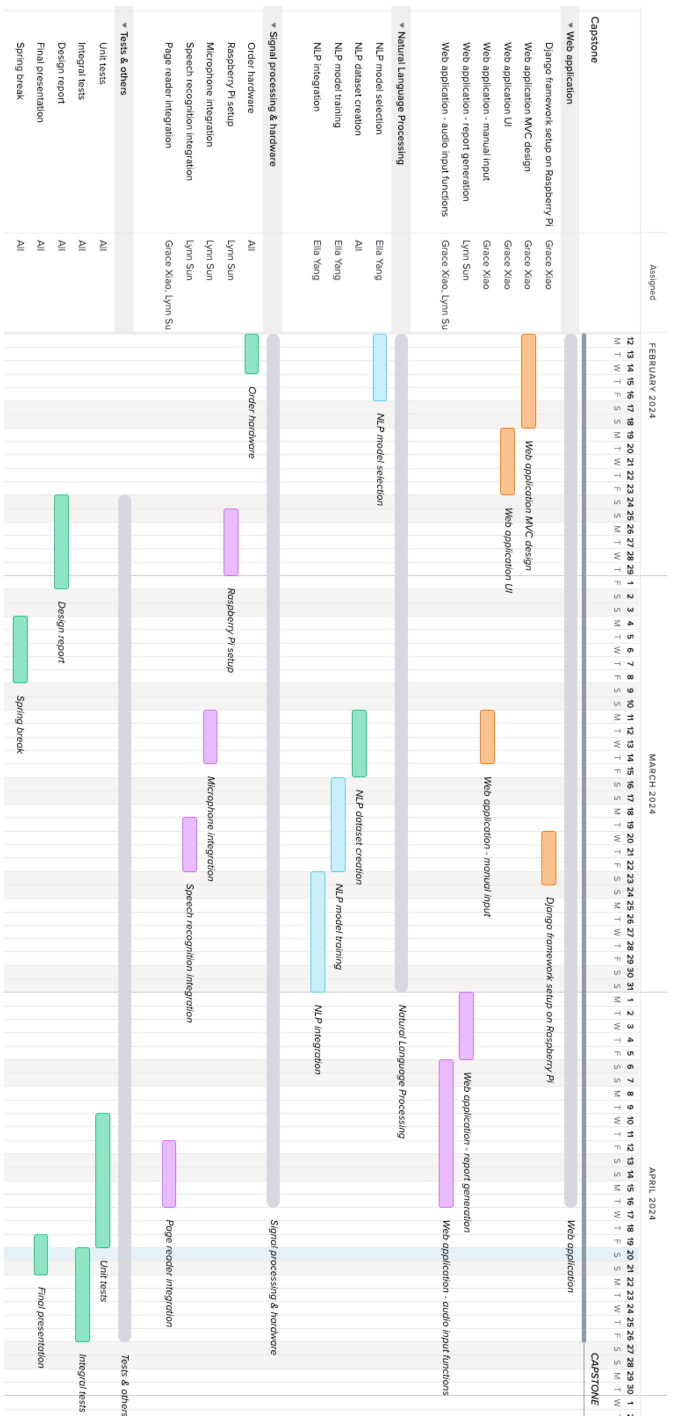


Fig. 5. Schedule Gantt Chart

D. *Risk Management*

Accessibility: It might be hard for visually impaired people to press accurately on the button. To assist them in pressing on the recording button, we will notify the user upon the start of the recording with a voice output of "start recording". We also implemented a "help" command to guide our user to follow templates of commands that are guaranteed to be recognized by our models. We respond to the "help" command by telling the user that the recording button is on the right half of the screen and that there are three basic commands they can start exploring the system with: enter a new expense, get entries, and generate a report. We also prompt the user about the available commands on each page after rendering that page so that the user will not get confused about the next steps. For example, after reading out the list of entries on the record page, the voice assistant will tell the user how to modify and remove a specific entry.

Latency: The Google News word2vec model failed to be loaded on Raspberry Pi probably due to the large size (1.5 GB). Instead of spending time reducing the dimensions of the model and truncating the unnecessary words (non-nouns), we decided to work around with a smaller pretrained GloVe model which is only 347.1 MB in size. The change also reduced the latency in item classification because the words in the new model are only 100-dimension compared to the 300-dimensional Google News model. The GloVe model met the accuracy requirement of item classification, but it also comes with a drawback of accepting only single words but not phrases.

Error handling: Our NLP models would not support all variations of voice commands and could only parse imperative sentences for the verb to identify the requested action. Errors in audio-to-text and text-to-command conversions could cause a failure to parse the command. We caught all these errors by displaying an error message on the page and playing an audio output such as "no such category" or "action not supported". We thoroughly tested all edge cases and made sure that the web page would not crash under any scenario.

IX. Ethical Issues

Ethical concerns of our product can be divided into two main parts: privacy and accessibility.

Since all the information of the user is stored in the raspberry pi, if the user loses the product, their financial information may be seen by other people. This would be a severe privacy issue. To tackle this problem, we may add some user identification in the future to make sure only the owner of this product would be able to open it.

For accessibility, there is a risk of bias in voice recognition systems, which might not work equally well for all accents or dialects, potentially leading to unequal access or quality of service based on demographic factors. This bias could potentially lead to frustration and financial errors. We would address this problem in several ways. First, in addition to voice commands, our system also supports typing. If our system cannot get several words correctly, users could choose to type it. In addition, we also have an "edit" action for users to modify existing entries, making error fixing very easy.

X. Related Work

There is currently no money tracking app with built-in voice control feature in the market. Apple does have comprehensive accessibility support on iPhone, iPad, and iPod. There are voice controls that help users interact with the screen, display accommodations for the visually impaired, and Siri to convert voice into very basic commands. However, the accessibility support cannot describe UI components for the user, and most apps do not support internal interaction with Siri. Hence, Apple's accessibility support does not provide a practical or efficient way to help the visually impaired, especially fully blind, group to interact with a money tracking app.

XI. Summary

Our project is a money tracking application integrated on Raspberry Pi with innovative features of audio input and webpage reader. It's a highly interactive, easy-to-use, and inclusive money tracking app that helps visually impaired and elderly people log, review, and visualize their spendings with our powerful speech recognition and NLP models. The main challenge is to correctly parse elements from a text with NLP models, and our team will carefully create datasets and tune the models for satisfactory outputs. Our team is confident that this product provides a promising solution to catering the need for financial management and budget keeping of the visually impaired and elderly people.

A. Future work

While our current system demonstrates a complete product of our original design, further improvements can still be made to enhance user experience and serve more customers.

The current solution only supports approximately 7 voice commands and can only accept imperative sentences, parsing a command based on the key word, which needs to be a verb. We could train the SpaCy models with more comprehensive datasets and incorporate more SpaCy models into the NLP pipeline to support commands with various sentence structures, giving the user a greater degree of freedom to speak out their requests.

Additionally, there are only 6 pre-defined categories to assign to the spendings, but the user might have the need to create their own categories. Future versions of our product could allow the user to customize their category list while still supporting the automatic classification of item names. One potential method to implement this functionality is by checking the similarity between new item names and the category names while constantly updating the words under each category to be compared to future item names. Therefore, the user can create their own categories while still enjoying an intelligent and self-updating item classification system.

Another improvement we can make is to support more languages to benefit non-English speakers. The internationalization process includes designing the web page layout based on text sizes and positions, handling dates, currencies, and numbers as dynamic content, and testing the product in different languages and regions. The localization process includes translating the contents, training the NLP models, and using the text-to-speech libraries for different languages. By supporting more languages, our product can serve the need for more people, especially the visually impaired group.

B. Lessons Learned

Creating a project from scratch is very different from implementing a system given the requirements. We need to think from the user's perspective, come up with detailed requirements, and evaluate the trade-offs for each design decision we make. Our group gained valuable experience in gathering information online, choosing the best tools based on the requirements, and learning new tools by ourselves.

We also recognized the importance of user testing to the finalization of our product. The quantitative tests have limitations because we tend to ignore the special cases while designing those tests. By inviting volunteers to play around with our product, we fixed many bugs caused by edge cases and improved the user interface based on their feedback.

Glossary of Acronyms

FPC - Flexible Printed Circuit
GPIO - General Purpose Input/Output
GUI - Graphical User Interface
HDMI - High-Definition Multimedia Interface
NER - Named Entity Recognition
NLP - Natural Language Processing
RPi - Raspberry Pi
UI - User Interface
VUI - Voice User Interface

References

[1] "Gensim: Topic Modelling for Humans." Modelsl.word2vec embeddings – genism, December 21, 2022[Online]. Available: https://radimrehurek.com/gensim/models/word2vec.html.
[2] Great Learning Team. "What Is Word Embedding: Word2vec: Glove." Great Learning Blog, February 23, 2024 [Online]. Available: https://www.mygreatlearning.com/blog/word-embedding/.
[3] S., Edward. "SQLite vs Mysql – What's the Difference." Hostinger Tutorials, December 21, 2022 [Online]. Available: https://www.hostinger.com/tutorials/sqlite-vs-mysql-whats-the-difference/.
[4] "Support." Official Apple Support. Accessed March 1, 2024 [Online]. Available: https://support.apple.com/accessibility.
[5] J., Lindner. "Statistics About The Average Arm Length." Gitnux, February 7, 2024 [Online]. Available: https://gitnux.org/average-arm-length/#:~:text=An%20average%20NBA%20player's%20arm,humans)%20is%20about%2082%20cm..
[6] What Noises Cause Hearing Loss?" Centers for Disease Control and Prevention, November 8, 2022 [Online]. Available: https://www.cdc.gov/nceh/hearing_loss/what_noises_cause_hearing_loss.html#print.
[7] E., Ryan. "Website Load Time Statistics: Why Speed Matters in 2024." WebsiteBuilderExpert, November 27, 2023 [Online]. Available: https://www.websitebuilderexpert.com/building-websites/website-load-time-statistics
[8] "Speech-to-Text transcript accuracy rate among leading companies worldwide in 2021." Statista, May 2021 [Online]. Available: https://www.statista.com/statistics/1133833/speech-to-text-transcript-accuracy-rate-among-leading-companies/.
[9] "Apple iPhone 15 Pro Max Battery Test." DXOMARK, December 8, 2023 [Online]. Available: https://www.dxomark.com/apple-iphone-15-pro-max-battery-test/
[10] Abraham CH, Boadi-Kusi B, Morny EKA, Agyekum P. Smartphone usage among people living with severe visual impairment and blindness. Assist Technol. 2022 Sep 3;34(5):611-618. doi: 10.1080/10400435.2021.1907485. Epub 2021 May 3. PMID: 33760680.

TABLE V. BILL OF MATERIALS

Description	**Model #**	**Manufacturer**	**Quantity**	**Cost @**	**Total**
Raspberry Pi 4 4GB	Model B	Raspberry Pi	1	ECE inventory	$0
Raspberry Pi Starter Kit	SD card, cables, etc.	CanaKit	1	ECE inventory	$0
7-inch Touch Screen	IPS HD 1024*600	NORSMIC	1	$69.99	$69.99
Microphone	AU-UL10 USB	MAONO	1	$22.99	$22.99
Portable Power Supply	10000mAh 5V/3A	Charmast	1	$29.99	$29.99
				Grand Total	**$122.97**