TutorPup

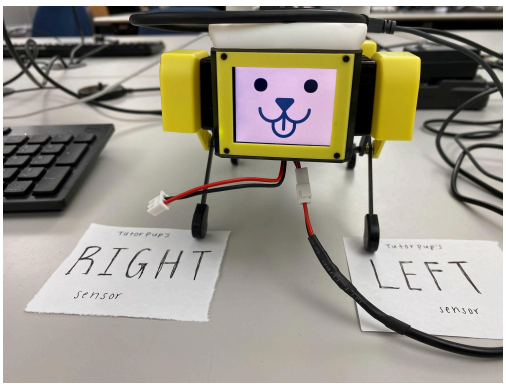
TutorPup is an educational robot that makes use of the Mini Pupper 2 quadruped robot. It aims to improve students' learning experience through the use of personalized question decks and enforce retrieval practice when reviewing each question.

Team Members:

Anna Niu, Role: GUI Design and GUI Implementation (Section 3.1.1, 3.1.2)

Tracy Truong, Role: Audio Controller, Touch Sensor Management, and Idle Expressions (Section 3.1.5, 3.1.6, 3.1.7)

Yuancheng 'Kaleo' Cao, Role: Question Database and Question Management (Section 3.1.3, 3.1.4)



1. Executive Summary

TutorPup is an educational companion that makes use of interaction between a GUI display and the Mini Pupper 2 quadruped robot. It allows users to generate a custom question deck: for every question they input, they can provide three answer options—one correct option and two incorrect options. This is accomplished through user interaction with the GUI display, which stores these inputted questions/answers in a backend database. Once done, TutorPup will cycle through these questions until the user has mastered them (i.e. all questions have been answered correctly) before providing the option of replaying with the same question deck or creating a new one.

When the application administers questions, users indicate their answer option by tapping on the Mini Pupper's sensors, which correspond to randomized answer choices shown on the GUI display. Appropriate audio feedback—such as emitting a barking sound when a question is answered correctly, and emitting a growling sound when a question is answered incorrectly—is provided through the Mini Pupper's speaker system. Additionally, we make use of the Mini Pupper's LCD screen by displaying idle blinking expressions to create a sense of liveliness for TutorPup.

For our hardware components, we utilized a monitor to display the GUI, a mouse and keyboard for inputting question decks, and the Mini Pupper robot for playing audio and processing touch sensor inputs. For our software components, we used Tkinter to create the GUI display, and used Python to implement the database and audio/touch sensor logic. The resulting application is fully self-sufficient, including a "Help" page for instructions on how to use TutorPup, as well as a "Home" page that provides easy navigation.

After completing the technical components of TutorPup, we conducted a study to determine if participants using TutorPup to study new material would result in better retention of such material. To do so, we recruited ten participants and asked them to take a quiz, which contained a set of ten questions about a niche topic—late 20th century films—before having them use TutorPup to study the questions. Following this, we asked the participants to retake the quiz, as well as fill out a post-survey regarding their experience using TutorPup.

For the results of our study, we analyzed participants' quiz scores before and after they used TutorPup to study. We found that most participants experienced score improvements. We also looked at the results of our post-survey to conduct qualitative analysis regarding participants' perception of TutorPup (e.g. if they were likely to use it frequently, etc.).

At the conclusion of the study, we developed a comprehensive understanding of the positive and negative aspects of our design for TutorPup. Generally, most participants felt that the integration of the GUI display and the Mini Pupper was effective, indicating that the application was intuitive and user-friendly. However, many participants struggled with finding the touch sensors on TutorPup, indicating that the process of answering questions was somewhat awkward.

For future work, it would be interesting to explore different ways to improve the integration of touch sensor input. It would also be beneficial to expand on the GUI design, offering a settings option for the number of answer choices, as well as providing an option for users to view current questions in the deck. Because TutorPup is currently targeted toward English speakers, it would also be beneficial to implement different language options.

2. Introduction

The primary motivation behind developing TutorPup was to create an educational robot that could also act as a companion for holding the user accountable for learning new material. We were inspired by Quizlet's flashcard and multiple choice activities that help enforce retrieval practice, and wanted to develop a tool for students to practice recalling information and receiving immediate feedback (correct, incorrect) to boost long-term retention of study materials.

We were also inspired by the theory of social facilitation, which states that people often perform low-complexity tasks more efficiently in the presence of others who generate "enhanced drive" and evoke "dominant responses" (O'Connell, 2024). This ties into the interactivity component of TutorPup, which allows users to answer questions by tapping on the Mini Pupper's touch sensors and receive immediate audio feedback—barking, growling—as well as visual feedback on the GUI display. We also created idle blinking expressions for TutorPup, which contributes to the liveliness and presence of the Mini Pupper. Due to its repetitive and consistent behavior, TutorPup serves the role of a leader, furthering the aspect of accountability that our application provides (Sebo et al., 2020)

Because we wanted to test the effectiveness of TutorPup's ability to help users learn new material, our study involves measuring participants' quiz scores for a set of ten questions before and after using TutorPup to study. We also wanted to gain some insight into the interactivity of TutorPup. To do so, we designed a survey that draws inspiration from the System Usability Scale detailed in Lewis and Sauro's paper, "Revisiting the Factor Structure of the System Usability Scale." We had all participants fill this survey out after they completed the quiz a second time.

3. Methodology

In this section, we will discuss how we accomplished the aims and goals of our project that we described in Section 2. First, we will discuss the technical approach of our project. Next, we will discuss a high level overview of the human-centric work for our project. We have divided this section of the report into different subsections which convey the key aspects of our project's technical approach and human-centric work.

3.1 Technical approach

In this section we will discuss each individual component of the technical tasks. We began the technical portion of this project by creating a diagram for the overall control architecture that illustrates the connections between hardware components, which we used to begin listing the required tasks (See Figure 1). After making this list of tasks, we created a Gantt chart that outlined each group member's role and responsibilities. This was decided by individuals and their preferences.

The overall control architecture diagram was also used to detail how the robot's system would perform sensing, perception, control, and interaction (See Figure 1). From the diagram, we can see raw data is captured from the environment through the robot's touch sensors, and text input from the connected external display. This input data is then used to make decisions regarding how the robot and overall application should respond to these inputs, by feeding the data collected to the various output controllers. Finally, the diagram also illustrates how the robot interacts with users, through the output of feedback from both an external display, and the Pupper robot's built-in speaker and display.

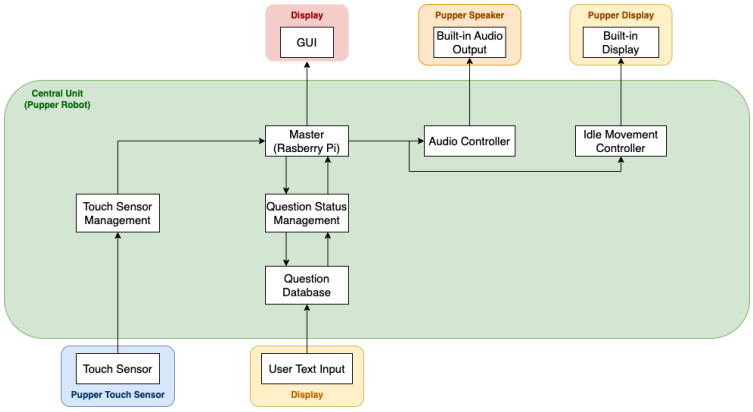


Figure 1: Overall Control Architecture

Following the division of tasks between group members, we then created a high-level diagram of the various software components our application would need, and planned how we thought it would fit together (See Figure 2). After we discussed what components and files we would need for our application, we created an FSM diagram to formally represent

the behavior of the application, and detail the interactions that would lead from one state to another for the application (See Figure 3).

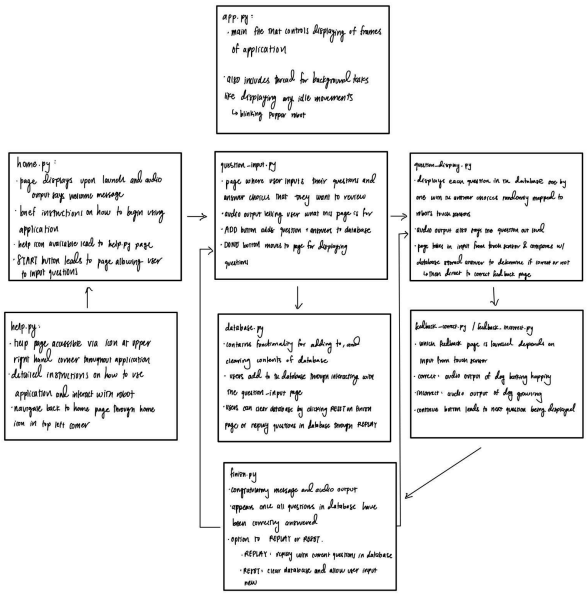


Figure 2: High-Level Diagram of Software Components and How They Fit Together

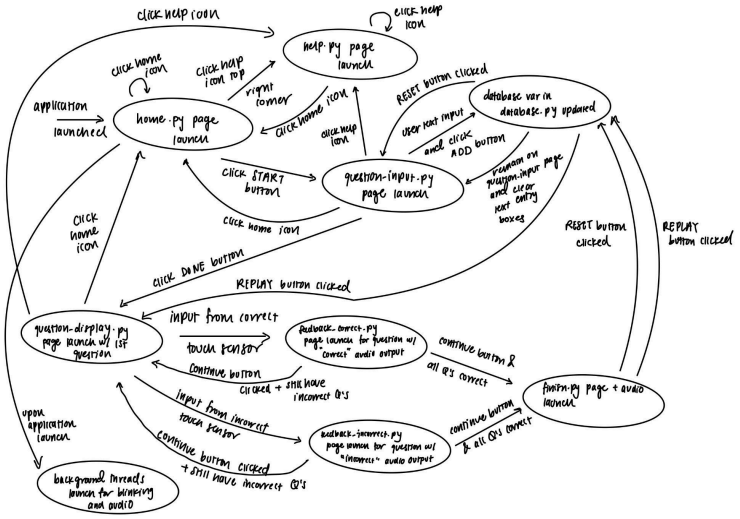


Figure 3: FSM Diagram for Formal Behavior Representation

We would work on our tasks individually, then commit our code to a shared Github repository and meet regularly to combine our code into a working prototype. Overall, collaboration and teamwork was a vital component of this project. We also used the feedback received from Professor Dr. Laurel Riek and teaching assistant Sandhya Jayaraman to help us iterate on our project design.

3.1.1 Technical Sub Task 1: GUI Design (Lead: Anna)

To design the GUI for the application, the first step we took was to create a low-fidelity UI wireframe using Figma (See Figure 4). Designing this wireframe gave us a basic idea of the application frames we would need to implement, how the different frames would interact with each other, when the user would provide input to the system, and when the application would provide audio and visual output. When creating this wireframe, we decided that the user interface would have six main frames. This included the home page, the help page, the question and answer input page, the question and answer display page, the feedback page, and the question deck complete page.

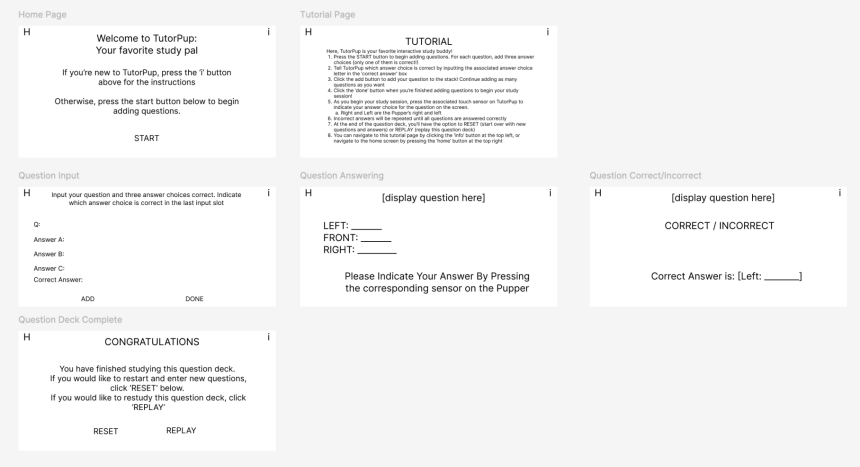


Figure 4: Low-fidelity User Interface Wireframe

The home page is the first page that is opened when the application is launched. It is meant to briefly give the user information on how to start using the application, and tell the user where to go to find more detailed instructions about TutorPup. The home page can be accessed from every frame in the application by clicking on the icon in the top right corner of the screen. Attached below is a close up of the initial design for this page created in Figma, and the eventual final home page design included in the final application (See Figure 5). As can be seen from the figure below, not much was changed about the design for the home page between the creation of the wireframe, and the final implementation of the application frame. The main change was adding color to the design.

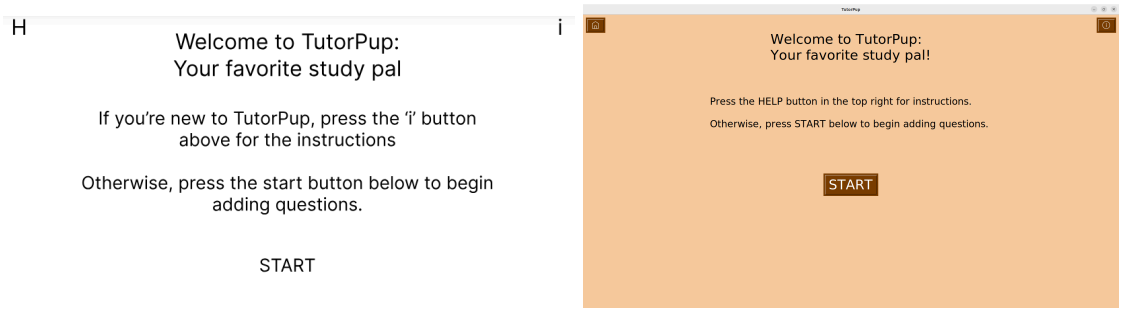


Figure 5: Initial (Left) and Final (Right) Home Page Design

The help page gives the user detailed instructions on how to use the TutorPup application, and what the different pages in the application all do. The user can navigate to the help page from every frame in the application by clicking on the 'help' icon in the top right corner of the screen. Attached below is a close up of the initial design for this page created in Figma, and the eventual final help page design included in the final application (See Figure 6). As can be seen from the figure below, the design remained the same overall, with the main change being the addition of a background color, and rewriting the instructions text to make it more clear.

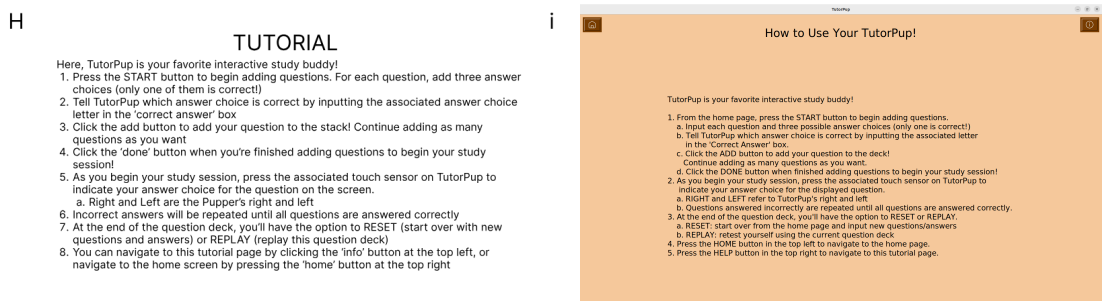


Figure 6: Initial (Left) and Final (Right) Help Page Design

From the home page, clicking the "`Start`" button causes the application to go to the question and answer input page. On this page, the user inputs their question, inputs three possible answer choices for the question, and indicates which answer choice is correct using the given user input text boxes. Clicking the "`Add`" button will add the given user input to the database of questions, and clear the text entry spaces. Clicking the "`Done`" button will signal to the application that the user is finished adding questions to study, and launch the question and answer display page. Attached below is a close up of the initial design for this page created in Figma, and the eventual final question and answer input page design included in the final application (See Figure 7). As can be seen from the figure below, the design largely remained the same overall.

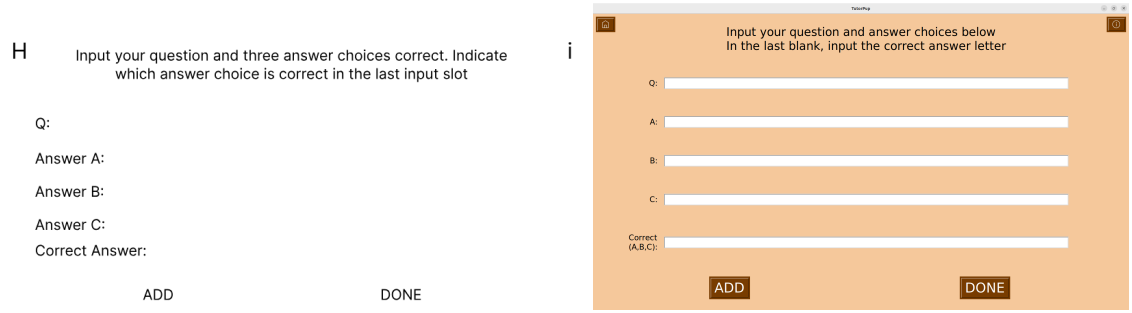


Figure 7: Initial (Left) and Final (Right) Question and Answer Input Page Design

The question and answer display page is launched once the user has signaled that they are finished adding questions to the database by clicking the "`Done`" button on the input page. This page displays the current question at the top of the screen, and also displays the three answer choices for the question and labels them with the three TutorPup touch sensors (left, right, and front). At the bottom of the screen, the page also tells the user how to answer the question by interacting with the TutorPup robot's touch sensors. This page is repeated for every question in the database that has not yet been answered correctly. Attached below is a close up of the initial design for this page created in Figma, and the eventual final question and answer display page design included in the final application (See Figure 8). As can be seen from the figure below, the design largely remained the same overall.

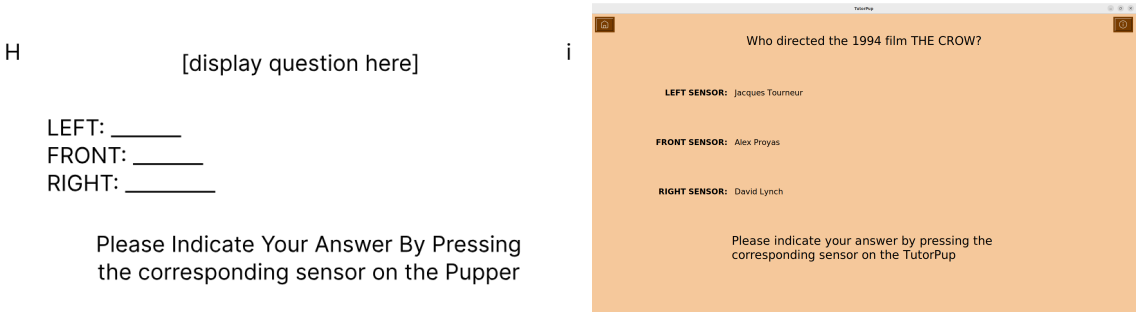


Figure 8: Initial (Left) and Final (Right) Question and Answer Display Page Design

From the question and answer display page, the user answers the displayed question by pressing one of the touch sensors on the TutorPup robot. Whether or not the user presses the correct sensor will influence which feedback page the application navigates to. If the user presses the correct sensor, the application will launch the feedback correct page, which tells the user that they answered the question correctly. If the user presses the incorrect sensor, the application will launch the feedback incorrect page, which tells the user that they answered the question incorrectly. From both feedback pages, pressing the "`Continue`" button will lead back to the question and answer display page and display the next question in the database that the user has not answered correctly yet. In the initial

Figma design, we had the feedback page as one frame in the application where the text in the frame would change depending on if the user chose the correct answer. In the final implementation, we made two different feedback frames, and the application would choose which frame to go to based on the user's answer. Attached below is a close up of the initial design for this page created in Figma, and the eventual final feedback page design included in the final application (See Figure 9). As can be seen from the figure below, the design largely remained the same overall.

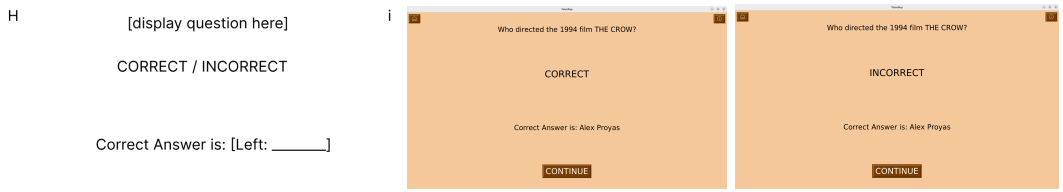


Figure 9: Initial (Left) and Final (Middle and Right) Feedback Page Design

When the user has correctly answered every question in the database, then the application launches the question deck complete page. This page displays a congratulatory message, telling the user that they have completed studying this question deck. It then also gives the user the option to either reset or replay the current question deck. Clicking the "`Reset`" button clears the question database, and navigates back to the question and answer input page for the user to input new questions to study. Clicking the "`Replay`" button allows the user to restudy the current questions in the database, and launches the question and answer display page for the user to re-study the question deck. Attached below is a close up of the initial design for this page created in Figma, and the eventual final question deck complete page design included in the final application (See Figure 10). As can be seen from the figure below, the design largely remained the same overall.

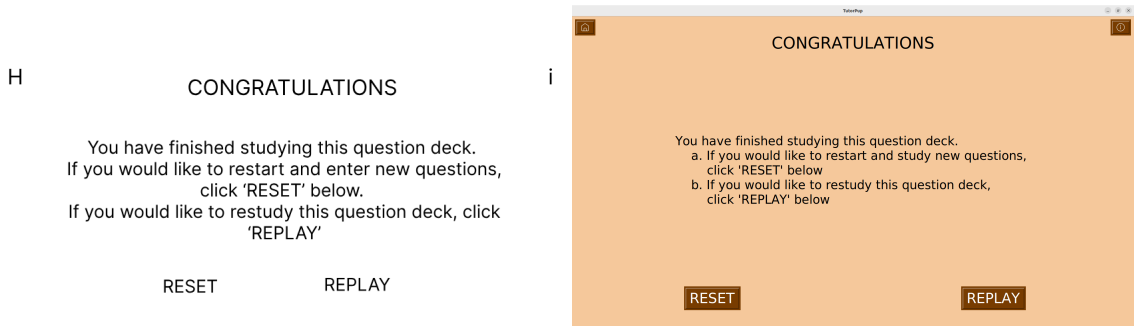


Figure 10: Initial (Left) and Final (Right) Question Deck Complete Page Design

3.1.2 Technical Sub Task 2: GUI Implementation (Lead: Anna)

The GUI was implemented using the Tkinter library. We used the `Tk` object defined in the Tkinter library to create the application which loads an interactive window when the program is run. The implementation of the GUI for the application follows the Figma

wireframe. The file structure of the program and a summary of what each file in the program does can be seen below (See Figure 11).

The file `app.py` is the main driver file for the application, and is where the frame objects for the GUI screens in the application are initialized. Running `app.py` launches the application in a new window. The files `feedback_correct.py`, `feedback_incorrect.py`, `finish.py`, `help.py`, `home.py`, `question_display.py`, and `question_input.py` each handle the implementation of a frame in the application. Each file defines the GUI details and backend functionalities for its frame.

TutorPup

→ `audio`	This folder stores all the application audio files.
→ `images`	This folder stores all the application image files.
> `README.md`	This file gives a brief summary of the project.
> `app.py`	This file initializes frame objects for the different GUI screens in the TutorPup application, utilizing the Tkinter library. On start up, the `homePage` frame will be displayed. The background thread for idle blinking is initialized in this file. Audio and Touch Sensor Management threads are also initialized here.
> `database.py`	This file initializes the question and answer database for the TutorPup application. It contains functions to add, remove, get, and reset items. The database uses a list structure.
> `feedback_correct.py`	This file creates the GUI for correct answer feedback. The `audio/barking.mp3` audio will play when this screen appears.
> `feedback_incorrect.py`	This file creates the GUI for incorrect answer feedback. The `audio/growling.mp3` audio will play when this screen appears.
> `finish.py`	This file creates the GUI for the finish screen, which is shown once a question deck is complete (i.e. all questions have been answered correctly). The `audio/congratualtory.mp3` audio will play when this screen appears.

`> help.py`	This file creates the GUI for the `helpPage` screen, which is shown when users select the 'help' icon. This page lists the instructions for how to use TutorPup.
`> home.py`	This file creates the GUI for the `homePage` screen, which is shown at application startup. The `audio/welcome.mp3` audio will play when this screen appears for the first time.
`> json_utils.py`	This file is no longer used in the final version of the application, but was originally where the database elements were accessed.
`> question_display.py`	This file creates the GUI for the `displayPage` screen. The question will be read out loud through the Pupper's speaker system.
`> question_input.py`	This file creates the GUI for the `inputPage` screen. The `audio/questionInputAudio.mp3` audio will play when this screen appears.

Figure 11: File Structure and Summary for TutorPup Application

All the GUI components are created with the `ttk` object in the Tkinter library. Objects that we used for this application include 1) images, to use self-uploaded images as part of the UI; 2) labels, to display text; 3) buttons, to link to different functionalities upon request of the user; and 4) entry, to collect text input from the user. We used the `ttk` object's `Style()` function to create customized appearances for the above objects.

When implementing the GUI, we began by implementing based on the low-fidelity wireframe, and then waited to refine UI details once the backend functionality was better implemented and connected. By implementing the UI in this way, it allowed us to iterate on the design and implement features one at a time, which further helped with the debugging process.

3.1.3 Technical Sub Task 3: Question Database (Lead: Kaleo)

For the question database, we implemented a local solution using the list data structure in the Python file `database.py`. Each item in the list is represented as a dictionary, which allows us to store multiple pieces of information related to each question. The dictionary format for each question is as follows: `{"question": "", "option_a": "", "option_b": "", "option_c": "", "correct_answer": "", "status": False}`

The `question` key stores the actual question text, while `option_a`, `option_b`, and `option_c` store the corresponding answer options. The `correct_answer` key holds the content of the correct answer. The `status` key is a boolean value that indicates whether the question has been answered correctly or not. By default, the value of `status` is set to 'False'.

To manage the question database effectively, we created two essential functions in the local Python file:

1. `add_item()`: This function is responsible for adding new questions to the database. It takes the question text, answer options, and the correct answer as input parameters and creates a new dictionary item with the provided information. The newly created dictionary item is then appended to the list, effectively storing the question in the database.
2. `get_list()`: This function is used to retrieve the list of questions from the database. It returns the entire list of dictionaries, allowing other parts of the application to access and utilize the stored questions and their associated information.

By including the `correct_answer` key in the dictionary format, we can store the content of the correct answer along with the question and answer options. This allows us to randomly assign the answer options to different touch sensors while still being able to verify if the user's selected answer matches the correct answer. Overall, the local question database enables the TutorPup application to randomly assign answer options to touch sensors, evaluate user responses, and track the progress of questions answered correctly.

3.1.4 Technical Sub Task 4: Question Management (Lead: Kaleo)

The question management component of the TutorPup application is responsible for handling the interaction between the question database and the GUI, as well as implementing the logic for displaying questions, tracking answer choices, updating question statuses, and managing the overall flow of the learning experience.

To establish a connection between the question database and the GUI component, we import the `database.py` file, which contains the necessary functions to access the list of questions. This allows the GUI component to retrieve questions and answer choices from the database and display them to the user.

The question management component ensures that only one question is displayed at a time. It retrieves the list of questions from the database and identifies the next question to

be displayed based on the `status` key. Only questions with a status of 'False' are eligible for display, indicating that they have not been answered correctly yet.

To track which answer choice is correct for each question, the question management component utilizes the `correct_answer` key in the question dictionary. It stores the content of the correct answer, allowing the application to compare the user's selected answer with the correct answer and determine if the question has been answered correctly.

The question management component includes a mechanism to randomly assign the answer choices to the touch sensors. It shuffles the order of the answer choices (`option_a`, `option_b`, and `option_c`) and associates each choice with a specific touch sensor. This ensures that the correct answer is not always associated with the same touch sensor, promoting a more engaging and challenging learning experience.

When a user selects an answer by touching a sensor, the question management component compares the selected answer with the correct answer stored in the `correct_answer` key. If the selected answer matches the correct answer, the `status` key of the corresponding question dictionary is updated to 'True', indicating that the question has been answered correctly. This update is then reflected in the question database.

The question management component sends signals to the GUI and audio output components when certain conditions are met. If the user's selected answer matches the correct answer stored in the `correct_answer` key of the question dictionary, the question management component sends a signal to the GUI to navigate to the feedback page. The feedback page is designed to display the result of the user's answer and provide additional information. In addition to updating the GUI, the question management component sends signals to the audio output component to provide auditory feedback to the user.

If a question is answered incorrectly, the question management component rotates the question to the back of the queue and continues presenting questions until all questions have been answered correctly. This ensures that users have the opportunity to revisit and practice the questions they struggled with until they achieve mastery.

The question management component includes functionality to reset the database, removing all questions and answers, and to replay the database, resetting the question statuses and allowing users to revisit all questions in the database. These features provide

flexibility and control over the learning content, enabling users to start fresh or review previously answered questions.

3.1.5 Technical Sub Task 5: Audio Controller to Audio Output (Lead: Tracy)
Audio output is utilized in the following files: `home.py`, `question_input.py`, `feedback_correct.py`, `feedback_incorrect.py`, `finish.py`, and `question_display.py`.

In the `home.py` file, we first created an audio thread. The target for this thread was the `textToAudio()` function, which took in a string, "Welcome to TutorPup, your favorite study pal!", and used the gTTS library to convert the string to the `audio/welcomeAudio.mp3` file. Afterward, the Playsound library was used to play the audio.

With the exception of the home screen audio output, we first had to add conditions to the `show_frame()` function in the `app.py` file checking whether or not the current GUI screen on display was one of the above files. If so, then calls to each file's corresponding `playAudioThread()` function were made. We didn't add in a condition for `home.py` because we only wanted the `audio/welcomeAudio.mp3` file to play once upon application startup.

1. If a call was made to the function `playAudioThread()` in file `question_input.py`, then it first checked if an audio thread was already created from this file and not yet closed. If so, then it would wait for that audio thread to finish running. If not, then it would create a new audio thread with a target to the function `textToAudio()`. This function would take in a string, "Input your questions and answer options!", and use the gTTS library to convert the string to the `audio/questionInputAudio.mp3` file. Afterward, the Playsound library was used to play the audio.
2. If a call was made to the function `playAudioThread()` in file `feedback_correct.py`, then it first checked if an `audioThread` was already created from this file and not yet closed. If so, then it would wait for that audio thread to finish running. If not, then it would create a new `audioThread` and play the `audio/barking.mp3` file using the Playsound library.
3. If a call was made to the function `playAudioThread()` in file `feedback_incorrect.py`, then it first checked if an `audioThread` was already created from this file and not yet closed. If so, then it would wait for that audio thread to finish running. If not, then it would create a new `audioThread` and play the `audio/growling.mp3` file using the Playsound library.

4. If a call was made to the function `playAudioThread()` in file `finish.py`, then it first checked if an `audioThread` was already created from this file and not yet closed. If so, then it would wait for that audio thread to finish running. If not, then it would create a new `audioThread` and play the `audio/congratulations.mp3` file using the Playsound library.
5. If a call was made to the function `playAudioThread()` in file `question_display.py`, then it first checked if an `audioThread` was already created from this file and not yet closed. If so, then it would wait for that audio thread to finish running. If not, then it would create a new `audioThread` with a target to the function `textToAudio()` and pass in an argument called `displayPage.currentQuestion_`, a string representation of the current question that the user was attempting in the question deck. The gTTS library converted the question string to a temporary audio file called `audio/question.mp3`. The Playsound library was then used to play this audio file. The `displayPage.currentQuestion_` field and `audio/question.mp3` file are both updated every time a new question is shown on the display screen.

3.1.6 Technical Sub Task 6: Touch Sensor Management (Lead: Tracy)

To implement the touch sensor management, we first had to add a condition into the `show_frame()` function in the `app.py` file checking whether or not the current GUI screen on display was the question display page. If this condition was met, then we made a call to the function `waitForSensorInputThread()` in the `question_display.py` file. This function first checks whether or not a thread that waits for touch sensor input was already running. If so, then it would wait for that thread to finish running. If not, it would create a new one, indicating that a new question was being displayed.

The thread's target is the `receiveSensor()` function. Here, a continuous loop is run to check whether the Pupper's front, left, or right sensors have been tapped by the user. Once it detects input from the user, then a call is made to the `answer_question(user_answer)` method, where `user_answer` is the string representation of the answer corresponding to the sensor that was tapped. The user's answer option is compared to the correct answer. If they match, then the `feedbackCorrectPage` frame is shown. If they don't match, then the `feedbackIncorrectPage` frame is shown.

3.1.7 Technical Sub Task 7: Idle Expressions (Lead: Tracy)

To implement the idle expressions, we first created two images—an open-eye expression, a closed-eye expression—in Canva (See Figure 12). Then, we resized these two images so that their dimensions aligned with the dimensions of the Mini Pupper's LCD screen. The original images (`eyes_opened.png`, `eyes_closed.png`) and their resized versions (`image_eyes_openedRZ.png`, `image_eyes_closedRZ.png`) can be found in the `images` folder.

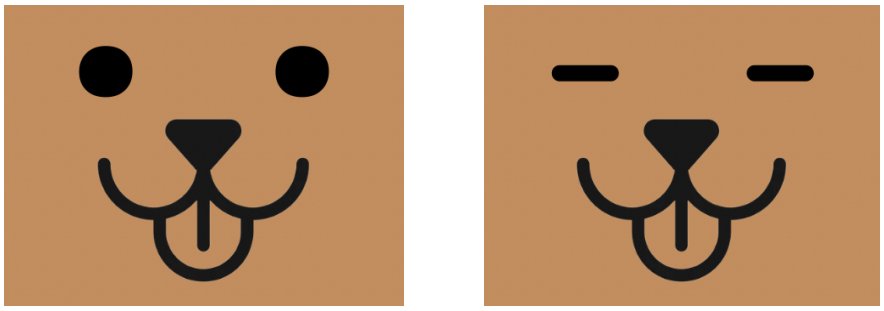


Figure 12: TutorPup's Open-Eye (Left) and Closed-Eye (Right) Expressions

In the `app.py` file, we then created a background task to have TutorPup "blink" on application startup. To do so, we created a thread that constantly looped through the following commands: display the open-eye expression, sleep for three seconds, display the closed-eye expression, sleep for half a second. Essentially, this provides the illusion of TutorPup blinking every three seconds. We assigned the background task thread to be a daemon thread; this allows the background task thread to be properly closed upon program termination.

3.1.8 Technical Sub Task 8: Application Testing (Lead: All)

In this subtask, we focused on testing the various functionalities of our application. During implementation, we focused on one feature at a time, which also allowed us to focus on testing each feature one by one rather than attempting to debug the entire program at once. The main way we tested our application was by running the program multiple times whenever a new feature was implemented and interacting with the program like we were users of the application.

The first page that was implemented on the application was the home page. For the home page, we tested that clicking on the 'home' and 'help' icons would bring the user to the correct page. Once we confirmed that the home page was functional, we began implementing the UI and features for the other pages of the application.

The second page that was implemented on the application was the help page, because it does not have as many dependencies as the other frames in the application. For the help

page, we tested that the instructions on the screen would not be cut off when the user navigates to the page. We also tested that clicking the ‘help’ icon would have the application stay on the help page. From the home page, we would test that clicking the ‘help’ icon would navigate to the help page, and that clicking the ‘home’ icon on the help page would navigate back to the home page.

The third page that was implemented on the application was the question and answer input page. We tested that upon application launch and the user being on the home page, clicking the “Start” button would navigate to this input page. We also tested that the page would accept user input via the four text entry boxes on the input page. At this point with the GUI implementation, the “Add” button functionality was not implemented yet, as we did not yet have the database implementation. Thus, we were not able to test the “Add” button functionality and check that it added the user input to the question database. However, we were able to test the “Done” button functionality, and check that it would navigate to the question and answer display page. We also tested that the ‘home’ and ‘help’ icons still worked from this page.

The fourth page that was implemented was the question and answer display page. Since the database and touch sensor functionality was not implemented yet, the first iteration of this page was a basic UI layout denoting where the question and answer choices should go on the screen. For navigation purposes, we added a dummy button that would navigate to the basic feedback page. This dummy button was later removed from the UI once the database and touch sensor functionality was implemented. For this page, we were able to test that it successfully launched, and we could navigate to the feedback page. We also tested that the ‘home’ and ‘help’ icons still worked from this page.

The last two pages that were implemented were the feedback page and the question deck complete page. The feedback page was initially implemented as a basic UI frame denoting what feedback should be shown depending on what the user answered. For easy navigation purposes, we had the “Continue” button immediately lead to the question deck complete page upon being clicked, since there was no question database yet. For this page, we were able to test that it successfully launched, and we could navigate to the question deck complete page. We also tested that the ‘home’ and ‘help’ icons still worked from this page.

The question deck complete page showed the congratulatory message, and the “Reset” and “Replay” buttons. We could not test that the buttons would clear or update the question database, since that had not been implemented yet. However, we were able to test that the “Reset” button would navigate back to the question and answer input page,

and the "Replay" button would navigate to the question and answer display page. We also tested that the 'home' and 'help' icons still worked from this page.

Upon the basic GUI being implemented for all the pages, we then implemented the question database. The database object itself and functions for interacting with it were implemented in the file `database.py`. Upon the database functionality being implemented, we first tested that the user's text input via the question and answer input page would be added to the database when the "Add" button was clicked. We then tested that the question and display page would display the questions in the database one at a time, and randomly match the question's answer choices to sensors and reflect this matching via the UI. Additionally, we tested that clicking the "Reset" button on the question deck complete page would clear the database, and that clicking the "Replay" button would change the statuses of all the questions in the database back to 'False'. Finally, we tested that the UI for the question and answer display page and feedback correct or incorrect page would correctly display the current question and answer choices via the UI, rather than displaying the default values we set when we implemented the basic UI.

Following the implementation of the database, we then implemented the touch sensor functionality. We tested this functionality by testing that the program received touch sensor input when the application was on the question and answer display page. We also tested that touching the correct sensor would cause the application to navigate to the feedback correct page and update the question's status to 'True' in the database, while touching an incorrect sensor would cause the application to navigate to the feedback incorrect page, and the question would later be asked again via the display page.

Finally, we implemented the audio output and idle movement features. For audio output, we tested that the correct audio would be output depending on the current frame we were on. For the idle movement feature, we implemented 'blinking' by using a background thread. We tested that the thread would be created upon launch of the application, and killed once the application was terminated. We also tested that this thread would run continuously throughout the application, and that any interaction with the application would not cause the blinking motion to stop or lag.

Once all the features were implemented, we began to test the application with different questions and answer choice inputs. For example, we would test with very long question and answer inputs to check if the text was ever cut off in the UI. We would also run multiple tests with dummy input to stress test the "Reset" and "Replay" functionality. For example, we would replay the current question deck, then reset it and enter new questions, then replay the new question deck, all without terminating the

application. By testing the application in this way, we were able to optimize the UI to ensure all the text was readable and nothing on the screen was cut off, and finetune the "`Reset`" and "`Replay`" database functionalities.

3.2 High Level Overview of Human-centric Work

In this section, we will summarize the human-centered aspects of TutorPup. Our goal in developing TutorPup was to provide an interactive way for users to study new material by letting them create personalized question decks. In the repetition of answering these questions, we seek to enforce retrieval practice.

In designing the interactivity aspect, we sought to make the most out of the Mini Pupper's unique hardware features, including the touch sensors, speaker system, and LCD screen. Because the Mini Pupper is a quadruped robot, its physiological features, inspired from zoomorphism, can shape the way users project their expectations regarding how the robot will behave (Teyssier, 2020). Due to this, we wanted to have the tapping of the touch sensors mimic the petting of a dog, the audio feedback to mimic a dog's barking/growling behavior, and the LCD screen to display a dog's blinking expressions. All of these features contribute to TutorPup's role and presence as a study companion.

To test the effectiveness of TutorPup, our study had participants take a knowledge quiz before and after using TutorPup to study. We then compared these scores to determine if there was a general trend of score improvements. We also had participants fill out a post-survey, which measured more qualitative items, such as if they found TutorPup easy to use. These results helped us analyze the effectiveness of TutorPup's interactive aspect.

3.2.1 Methodology

In this section, we will discuss the methodology that our study took on.

a. *Overview*

   Our study focuses on evaluating the effectiveness and usability of TutorPup, an educational robot designed to enhance learning through interactive study sessions. We aimed to assess how well TutorPup improves knowledge retention and user satisfaction among participants. The experiment involved a pre-test to measure initial knowledge, a study session using TutorPup, and a post-test to assess learning improvement. Additionally, we gathered feedback on user experience through the System Usability Scale (SUS) questionnaire.

b. *Participants*

   We recruited participants who had no prior knowledge of late 20th century films, ensuring that they would benefit from the learning experience provided by

TutorPup. The recruitment process involved reaching out to individuals both within and outside of the CSE 190/276B course. A total of 10 participants were recruited for the study. This sample size allowed us to gather diverse perspectives and obtain meaningful feedback on the TutorPup.

Among the 10 participants, there were 7 males and 3 females. The majority of the participants were students majoring in STEM (Science, Technology, Engineering, and Mathematics) fields. The age of the participants ranged from 20 to 24 years old. Overall, the mix of participants aligns well with the target audience of TutorPup, as it aims to provide an interactive and engaging learning experience for individuals with an interest in technology and learning.

c. *Protocol*

Before starting the experiment, our team asked participants to sign a consent form. We then provided them with a Google form containing ten questions, such as "Who directed the *YEAR* film *TITLE*?" to assess their initial knowledge of the subject. Next, we instructed the participants to use TutorPup to study the material until they could answer all the questions correctly. Once they completed their study session with TutorPup, we asked them to retake the same ten questions to evaluate any improvement in their knowledge.

Finally, we distributed the System Usability Scale (SUS) questionnaire as a post-survey to evaluate the usability of TutorPup. This allowed us to gather feedback on participants' satisfaction and the overall user experience.

d. *Measures*

To evaluate the effectiveness of and usability of TutorPup, we employed two primary measures. First, we assessed the impact of TutorPup on participants' learning outcomes by comparing the number of questions they answered correctly before and after using the TutorPup to study. This measure allowed us to quantify the improvement in participants' knowledge and understanding of the subject matter.

Next, to evaluate the usability and user satisfaction of TutorPup, we utilized the System Usability Scale (SUS) questionnaire to survey participants after they had interacted with the TutorPup. Figure 13 presents the SUS questionnaire consisting of ten statements that participants rate on a scale of 1 to 5, where 1 represents "Strongly Disagree" and 5 represents "Strongly Agree." The statements cover various aspects of usability, such as ease of use, learnability, consistency, and user satisfaction. By analyzing the individual responses to each statement, we can

identify specific usability issues and make informed decisions about enhancing the user experience of TutorPup.

	The System Usability Scale Standard Version	Strongly Disagree	1	2	3	4	Strongly Agree 5
1	I think that I would like to use this system frequently.		o	o	o	o	o
2	I found the system unnecessarily complex.		o	o	o	o	o
3	I thought the system was easy to use.		o	o	o	o	o
4	I think that I would need the support of a technical person to be able to use this system.		o	o	o	o	o
5	I found the various functions in this system were well integrated.		o	o	o	o	o
6	I thought there was too much inconsistency in this system.		o	o	o	o	o
7	I would imagine that most people would learn to use this system very quickly.		o	o	o	o	o
8	I found the system very awkward to use.		o	o	o	o	o
9	I felt very confident using the system.		o	o	o	o	o
10	I needed to learn a lot of things before I could get going with this system.		o	o	o	o	o

Figure 13: The System Usability Scale (Finstad, 2006; Sauro & Lewis, 2009)

4. Results

In this section, we will discuss the results from both our robot-focused evaluation, and our human-focused evaluation.

4.1 Robot-focused Evaluation Results

Our team has successfully developed a TutorPup prototype that meets the requirements and objectives outlined in the project proposal. We have implemented all the functions and features that were proposed. The evaluation results demonstrate the effectiveness and efficiency of TutorPup. We have successfully created a user-friendly interface that enables seamless interaction between user and the robot.

In the following sections, we will delve into the detailed evaluation results for each major component of TutorPup, including the GUI design implementation, question database and management, and other relevant aspects.

4.1.1 GUI Design Implementation

Our team conducted a series of tests to evaluate the implementation of the GUI design. We focused on five key areas: the home page, the question and answer input page, the question and answer display page, the feedback page, and the question deck complete page.

We began by testing the launch of the home page upon starting the program. Our tests showed that the home page consistently appeared as expected when the application was

initiated. The page loaded quickly and all elements, such as buttons and text, were displayed correctly.

Next, we evaluated the page for inputting questions and answers, which includes an "Add" button for submitting individual question and answer pairs and a "Done" button for completing the input process. The input fields for questions and answers were functional, allowing users to enter their desired text. The page's layout and design meet our specifications.

We then proceed to test the page responsible for displaying questions and answers. Our tests revealed that this display page launched successfully when the "Done" button was pressed on the input page. The question and answers were displayed accurately, matching the information entered in the input page. The formatting and presentation of the content aligned with our design requirements.

After the question and answer display page, we assessed the feedback page. This page is designed to inform the user whether their answer was correct or incorrect and display the correct answer. Our tests confirmed that the feedback page appeared as expected after the user submitted their answer, displaying the user's results and the correct answer accurately. Additionally, a "Continue" button was present, allowing the user to proceed to the next question.

Finally, we evaluated the question deck complete page, which launches when the user has answered all questions correctly. Our tests verified that the page appeared as intended only after the user successfully completed all questions in the deck. The page's content, including any messages or prompts, was displayed accurately, confirming the user's completion of the question deck. The "Reset" and "Replay" buttons were also present, and successfully navigated to the correct pages upon being clicked. The "Reset" button navigated to the question and answer input page, and the "Replay" button navigated to the question and answer display page.

4.1.2 Question Database and Management

We evaluated the question database, which stores the questions and answers for our application. Our tests showed that the database successfully updated when new questions and answers were added, ensuring that the latest questions were available. Furthermore, when the status of a question changed, such as being marked as answered correctly, the database accurately reflected this change. The database also effectively communicated the change in item status to the question management system.

Next, we assessed the question management system. Our tests confirmed that the system correctly displayed the questions and answers stored in the database and accurately tracked when a question was answered correctly, updating the question's status accordingly. The system randomly assigned question answer choices to the touch sensors, ensuring a fair and unbiased presentation of options. When a question was answered correctly, the question status was promptly updated, and the system sent a signal to the output component to provide feedback to the user.

We then evaluated the system's ability to continue rotating questions with a 'False' status and end the rotation when all questions had a 'True' status. Our tests verified that the system effectively presented questions marked as 'False' until they were answered correctly. Once all questions in the database had a 'True' status, the system successfully concluded the question rotation process.

Finally, we tested the reset and replay functionality of the question database. Our evaluation confirmed that the reset feature effectively removed all entries from the database, providing a clean slate for a new session. The replay feature, on the other hand, successfully launched the question-answer display page, utilizing the same entries present in the database from the previous session.

4.1.3 Audio Controller to Audio Output

We evaluated the audio playback functionality of our application through the built-in audio output. Our tests confirmed that the audio controller successfully read and played the welcome message, input instructions message, questions, and congratulatory audio as intended. The audio was clear, audible, and synchronized with the corresponding text displayed on the screen.

Next, we assessed the system's ability to provide audio feedback based on the user's answers. When a question was answered correctly, the audio controller successfully triggered the "Bark" audio output, providing a positive reinforcement sound. Conversely, when a question was answered incorrectly, the "Growl" audio output was accurately played, indicating an incorrect response. These audio outputs were distinct and easily distinguishable, enhancing the user's experience and understanding of their performance.

Finally, we tested the audio controller's ability to receive and process touch sensor input signals. Our evaluation showed that the controller effectively detected and responded to the signals generated by the touch sensors. When a touch sensor was activated, the audio controller promptly initiated the corresponding action, such as playing the associated audio feedback.

4.1.4 Touch Sensor Management

Our team conducted testing to evaluate the touch sensor management. We focused on assessing the random assignment of answer choices to right, left, or front sensors. Our tests confirmed that the system effectively distributed the answer options across the three sensors, ensuring an unbiased presentation of choices.

Next, we verified that the question status flag was accurately updated based on the touch sensor input. When a user selected an answer by touching the corresponding sensor, the system correctly identified the chosen option and updated the question's status accordingly. Additionally, we tested the audio output triggered by the touch sensor input. Our evaluation showed that the system generated different audio output based on the user's selection, providing appropriate feedback for correct and incorrect answers.

Finally, we assessed the system's ability to correctly track which sensor corresponded to the correct answer choice. Our test confirmed that the system accurately maintained the association between the sensors and the answer options, ensuring that the correct answer was consistently identified regardless of its assigned sensor. We can conclude that the touch sensor management in our application meets the desired requirements, providing an accurate and responsive user interaction experience.

4.1.5 Idle Movement

Our initial design included idle movements such as occasional barking audio and leg motions, which were intended to enhance the user experience and make TutorPup more engaging. However, feedback from our pilot study revealed that these features actually distracted participants from learning. As a result, we made the decision to remove these elements and focus on maintaining a simple idle blinking expression to minimize distractions and promote a more focused learning environment.

We tested that the background thread used to implement the blinking motion would be created upon launch of the application, and killed once the application was terminated. We also tested that this thread would run continuously throughout the application, and that any interaction with the application would not cause the blinking motion to stop or lag.

4.1.6 TutorPup Effectiveness

The experimental results demonstrate that TutorPup is highly effective in improving participants' scores. According to Figure 14, out of the 10 participants, 9 showed an increase in their scores after using TutorPup to study. For example, Participant 1's and Participant 9's score increased from 4 to 9 correct answers, indicating a significant

learning improvement. This 90% improvement rate indicates that TutorPup is a valuable learning tool that can significantly enhance users' knowledge and performance.

Despite the overall positive trend, Participant 6 showed a slight decrease in performance, with their score dropping from 6 to 5 correct answers. This outlier suggests that while TutorPup is generally effective, there may be individual differences in how participants respond to the tool. Further investigation is needed to understand why Participant 6 did not benefit and how the tool can be adapted to meet diverse learning needs. Overall, the data suggests that TutorPup significantly boosts learning outcomes for most participants.

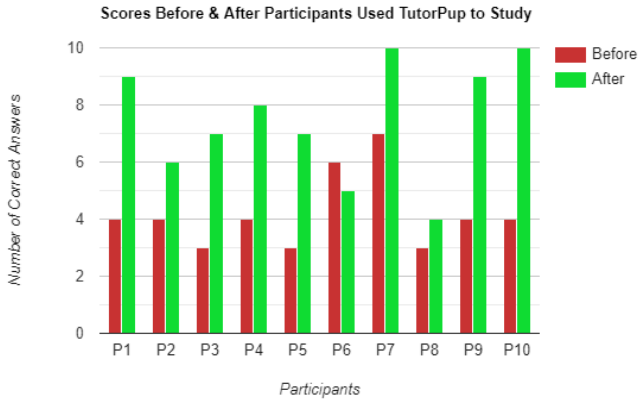


Figure 14: Improvement in Participants' Scores Before and After Using TutorPup

4.2 Human-focused Evaluation Results

To evaluate the usability of TutorPup, we used the System Usability Scale (SUS) survey to survey participants after they interacted with the system. The survey results indicated that TutorPup is highly user-friendly. The majority of participants found it easy to use, with 5 people strongly agreeing and 3 agreeing with this statement (See Figure 15). Additionally, most participants expressed that they would use TutorPup frequently, with 2 people strongly agreeing and 4 people agreeing (See Figure 16).

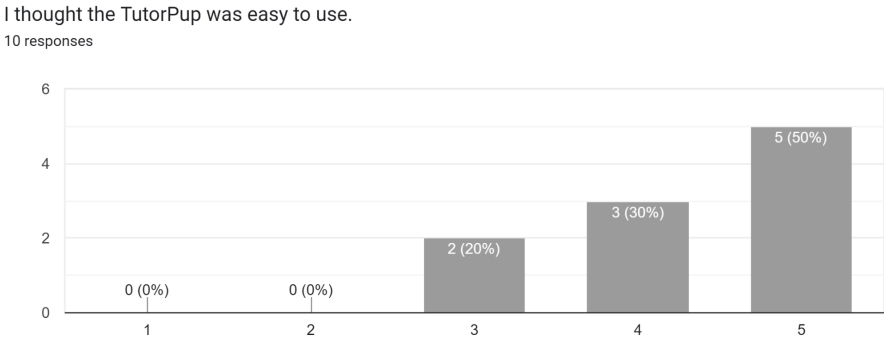


Figure 15: Participants' Opinions on Whether TutorPup is Easy to Use

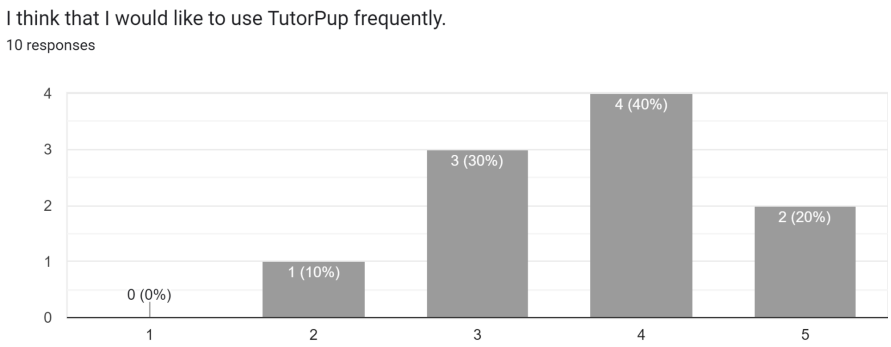


Figure 16: Participants' Opinions on Whether They Would Use TutorPup Frequently

Participants also appreciated the well-integrated functions of TutorPup, with 5 people strongly agreeing and 4 people agreeing that the functions were seamlessly integrated (See Figure 17). Moreover, the survey revealed that most participants believed that people would learn to use TutorPup quickly, with 6 people strongly agreeing and 3 people agreeing with this statement (See Figure 18). Many participants also verbally expressed that they didn't need to learn a lot of things before using TutorPup, indicating its intuitive design.

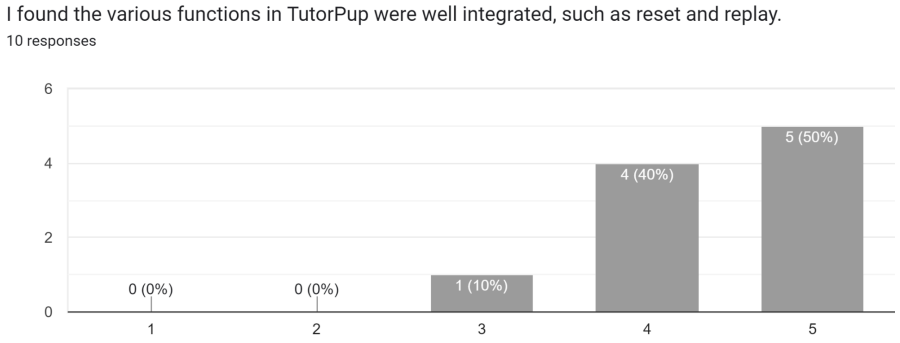


Figure 17: Participants' Opinions on Whether TutorPup's Functions Were Well Integrated

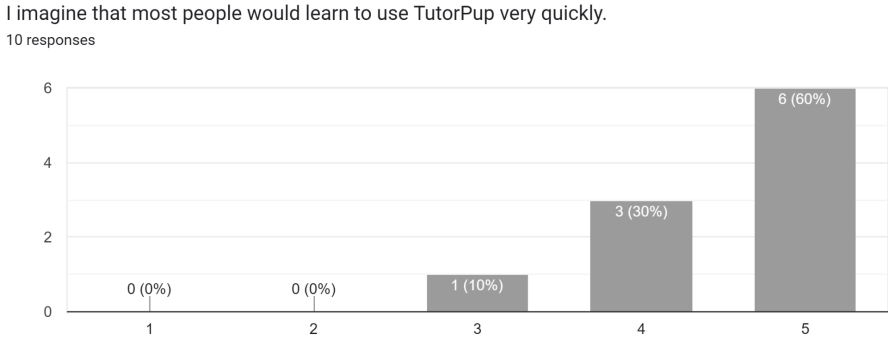


Figure 18: Participants' Opinions on Whether People Would Learn to Use TutorPup Quickly

The survey results also showed that participants felt confident using TutorPup, with 3 people strongly agreeing and 6 people agreeing with this statement (See Figure 19). Many participants also felt they didn't need support from a technical person to use TutorPup (See Figure 20).

There were mixed responses regarding the perceived awkwardness of using TutorPup. As shown in Figure 21, 2 people strongly disagreed, 4 people disagreed, 2 people were neutral, 1 person agreed, and 1 person strongly agreed with the statement that TutorPup was awkward to use. This variation in responses suggests that while the majority of participants found TutorPup easy to use, there may be some aspects of the system that could be improved to enhance the overall user experience. When asked about the usability, many participants verbally expressed that they found TutorPup's touch sensors difficult to touch and access. This feedback gives some possible insight into the reasoning behind the variation in responses seen in Figure 21.

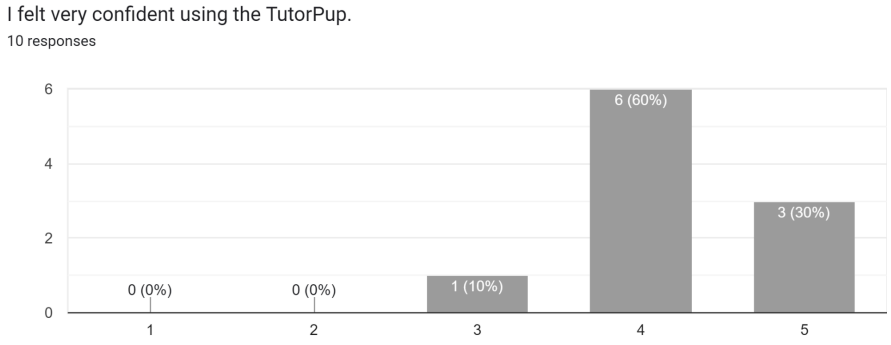


Figure 19: Participants' Opinions on Whether They Felt Confident Using TutorPup

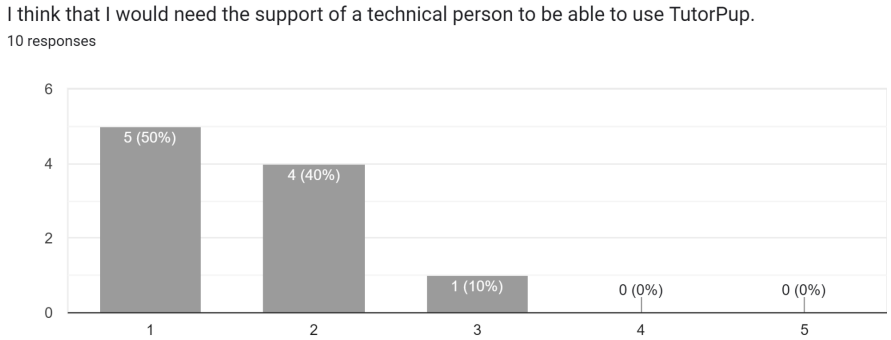


Figure 20: Participants' Opinions on Whether they Need Support From a Technical Person to Use TutorPup

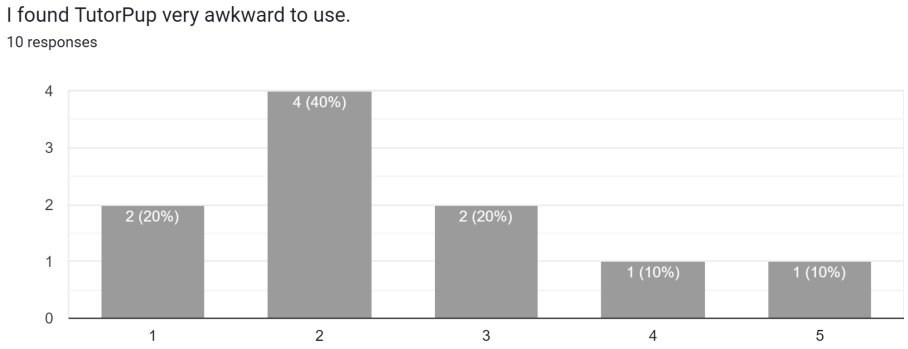


Figure 21: Participants' Opinions on Whether TutorPup was Awkward to Use

5. Discussion and Future Work

One of the main critiques we received after conducting our study was that the sensors on TutorPup were difficult to touch and see. Participants found it challenging to interact with the sensors, which contributed to the perceived awkwardness of using the system. To address this issue, we would like to remove the white case on top of TutorPup for future iterations of this project. By eliminating this physical barrier, users will have easier access to the sensors, making the interaction more intuitive and user-friendly.

In addition to improving the sensor accessibility, in the future we would also like to expand on the GUI design of TutorPup. Currently, the interface provides basic functionality for inputting questions and answers. One potential improvement is to include options for viewing already inputted questions, allowing users to review and revisit previous content. Furthermore, we would also like to add a setting that enables users to specify the number of answer choices, providing greater flexibility and adaptability to different learning scenarios.

Another important aspect we would want to focus on in future development is accessibility. Currently, TutorPup only supports the English language, limiting its usability for non-English speakers. A possible extension that future students could do, and future work that we would like to do if we could continue with this project, would be to incorporate multiple language options to make TutorPup more inclusive and accessible to a wider audience. By offering a variety of language settings, we can ensure that learners from different linguistic backgrounds can benefit from the interactive learning experience provided by TutorPup.

Moving forward, we will prioritize these identified areas for improvement and continue to gather feedback from users to guide our development efforts. By addressing the main critiques, expanding the GUI design, and enhancing accessibility, we aim to create more user-friendly and effective learning tools. We are excited about the future possibilities of TutorPup and look forward to further exploring its potential as a transformative educational tool.

References

Lewis, J. J. R., & Sauro, J. (2017). Revisiting the factor structure of the system usability scale. Journal of Usability Studies, 12(4), 183-192.

Marc Teyssier. Anthropomorphic devices for affective touch communication. Human-Computer Interaction [cs.HC]. Institut Polytechnique de Paris, 2020. English. ffNNT : 2020IPPAT023ff. fftel-02881894

O'Connell, Amy, et al. "Design and Evaluation of a Socially Assistive Robot Schoolwork Companion for College Students with ADHD." *Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction*. 2024.

Sarah Sebo, Brett Stoll, Brian Scassellati, and Malte F. Jung. 2020. Robots in Groups and Teams: A Literature Review. Proc. ACM Hum.-Comput. Interact. 4, CSCW2, Article 176 (October 2020), 36 pages. https://doi.org/10. 1145/3415247

Code

LINK TO CODE (zip file hosted on Google Drive)