Boston University
Electrical & Computer
Engineering

Boston University
Electrical & Computer Engineering
EC 464 Senior Design Project

Final Project Testing Plan

BUtLAR

By
Team 12
Digital Human - Yobe

Team Members

Noa Margolin noam@bu.edu
Suhani Mitra suhanim@bu.edu
Jackie Salamy jesalamy@bu.edu
Andrew Sasamori sasamori@bu.edu

Required Materials:

Hardware:

- Raspberry Pi V5
- Two Røde Microphones
- Laptop

Software:

- Python Virtual Environment
- Live Audio Processing
    - *miniaudio_stream.c*
        - Utilizes C++ miniaudio library to capture audio in real-time
- Yobe SDK (GrandE) → Audio Generation
    - *IDListener_demo.cpp*
- Context-Specific Database
    - *school.db* (database with information about course logistics — professors, time, location, etc.)
- Google ASR Speech-To-Text API, LLM API
    - *main.py*
        - Call the files below
    - *voiceAssistant.py*
        - Processes streamed audio and prepares for LLM processing
        - Handles user experience edge cases
        - Performs API calls
    - *txtToLLM.py, callLlm.py, manualCheck.py*
        - The txtToLLM.py script reads a user's question, corrects professor's name spellings using lastNames() from callLlm.py, and corrects any other words calling using obviousMispellings() from manualCheck.py.
        - Using OpenAI, generateSql() (using corrected professor names version) creates a SQL query based on the school.db course table.
        - Output is turned into a natural sentence in respondToUser()
- Django Framework (Notable Files)
    - *consumers.py*
        - Text-to-speech (TTS) and socket handling
    - *butlar_interface.html*
        - Creates web interface

Final Project Testing Goal: A use-case specific Voice Chatbot that supports a full conversation, employing speech to text, data processing, and LLM-generated relevant responses with low latency. We are targeting the hardware and full-stack functionality to provide our client, Yobe Inc, with a Voice Chatbot framework that their Software Development Kit (SDK) can be applied to and demonstrate its societal and commercial value.

Setup:

Our system setup begins with the hardware components: a Raspberry Pi connected via Ethernet to host the software on a Linux machine and two Rode Microphones for capturing audio input. The microphones are set at 9 inches apart, placed on the top of the laptop in our custom 3D-printed housing, which displays our web-based interface. The pipeline is driven by a Python script that manages audio capture, processing, and response generation, with Django serving as the web framework connecting our backend to the front-end interface. The components are summarized in *Figure 1* and *Figure 2*.

The backend workflow captures audio and processes it in real-time. Once audio is captured, Google's Speech-to-Text API generates a transcription with automatic punctuation enabled. The system uses a flag-based mechanism to manage conversation flow, pausing audio capture during system responses. When a question is transcribed, it's first processed for obvious misspelling corrections from the automatic speech recognition. Next, it loads instructor names from a CSV file and uses an LLM to detect and correct potential name errors through phonetic matching.

The corrected query is then transformed into SQL by the LLM, which is executed on our SQLite database containing faculty and course information. The system retrieves the results and generates a natural language response through our respondToUser function. Finally, the text response is synthesized using Google's Text-to-Speech API and transmitted through the speaker, with a web interface displaying both the query and response. The system includes timeout monitoring to handle periods of inactivity, ending the session after 45 seconds of silence. The flowchart is summarized in *Figure 3*.

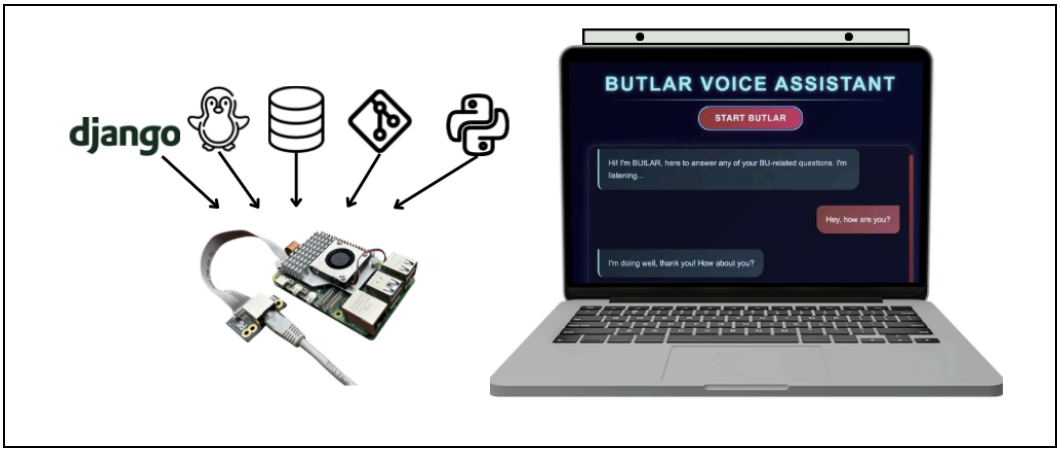

Figure 1: Hardware Set-Up Overview

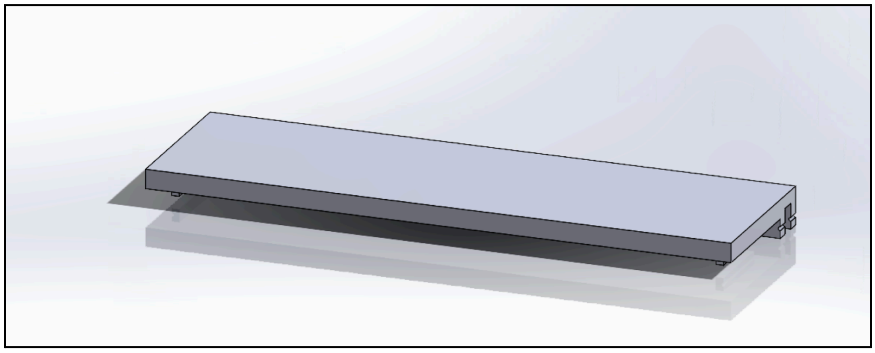

Figure 2: Microphone Housing CAD (SolidWorks)

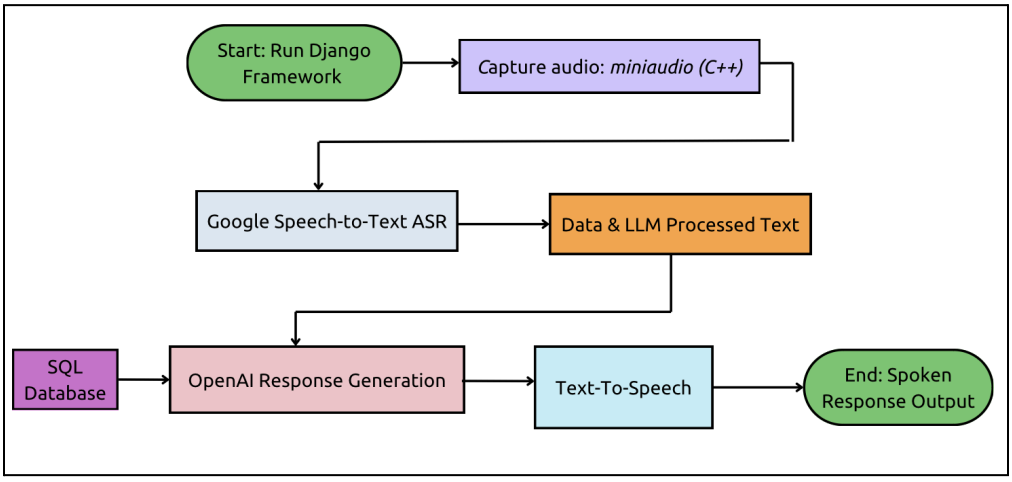

Figure 3: Software Pipeline Flowchart

Pre-Testing Setup Procedure:

Raspberry Pi Connection:

1. 2 AI-Micro Rode Dual Speakers are connected to Raspberry Pi.
2. Raspberry Pi is connected to the network via Ethernet.
3. Run and pipe the live audio streaming files on the Raspberry Pi.

Server-Side Connection:

1. Establish SSH connectivity with the Raspberry Pi (remote access) using the following command: `ssh yobe@128.197.180.176`
2. Navigate to the appropriate directory:
    a. `cd BUtLAR_Voice-Powered-Digital_Human_Assistant/Audio/testing_audio/django_top`

Running the Session

1. Start a GCloud virtual environment:
    a. `source ~/gcloudenv/bin/activate`
2. Start running server
    a. `daphne -b 127.0.0.1 -p 8000 django_top.asgi:application`
3. Access the server here and press
    a. `http://127.0.0.1:8000/butlar/interface/`

Testing Procedure:

3 specific tests must be evaluated as either “Pass” or “Fail.” To achieve a “Pass,” each test must meet its unique criteria, ensure a latency of less than 4 seconds from the end of the audio recording to transcript generation, and produce a transcript that accurately conveys the intended message.

1. Specific (Location-based Query, Class Type-based) Queries
    a. Prompt BUtLAR using a Class ID & name
    b. Prompt BUtLAR using a course type (lab/discussion/lecture)
2. Name Correction
    a. Mispronounced names with inaccurate transcriptions are matched and output the correct last name
    b. Last name is properly detected from the database list
3. Multiple Queries (Miscellaneous Prompts)
    a. Users can ask multiple questions until they terminate the session
    b. Prompting questions with a less apparent/straightforward answer

Measurable Criteria:

Specific Test Case Requirements:

I. **Specific (Location-based Query, Class Type-based) Queries**
   A. Prompt BUtLAR using a Class ID & name
      1. "When is EC 414?"
      2. "When is machine learning?"
   B. Prompt BUtLAR using a course type (lab/discussion/lecture)
      1. "When is EC 414 discussion?"
      2. "Where is EC 311 lab?"

II. **Name Detection**
   A. The transcript after Speech-To-Text will be checked for conveying the Professor's correct last name.
   B. We will say, "What does Professor Eagle Teach?" → Correct Name: Egele
   C. We will say, "What does Tally Moret teach?" → Correct Name: Tali Moreshet

III. **Multiple Queries (Miscellaneous Prompts)**
   A. We will first ask, "How many discussion sections does EC414 have?"
   B. We will then ask, "What class is on Tuesdays and Thursdays from 1:30 to 3:15 PM in CAS 227?"

   *Conversation will terminate when the user says, "Goodbye, BUtLAR."

General Requirements:
In addition to satisfying the criteria above, the system must meet the following overarching requirements for every test case:

- **Latency:** The time from the end of the audio recording to the generation of the LLM-generated response must be less than 5 seconds.
- **Message Accuracy:** The transcript must accurately convey the intended message query.
- **Response Relevancy:** All answers provided must be accurate and relevant to the user's question.
- **Conversation Termination:** Conversation terminates when the user says, "Goodbye, BUtLAR."
- **Timeout:** A session will timeout if no user speaking is detected for 45 seconds.

Score Sheet:

Requirement	Transcript is correct (Y/N)	TTS output is correct (Y/N)	Latency (< 4 sec)	Pass/Fail
Specific Query Test 1 & 2	N/A	Y	Q1: 1.78 Q2: 1.68	PASS
Specific Query Test 3 & 4	N/A	Y	Q1: 1.69 Q2: 1.82	PASS
Name Correction Test 1	(Print both wrong and corrected) Y	Y	1.53	PASS
Name Correction Test 2	(Print both wrong and corrected) Y	Y	1.38	PASS
Multiple Queries Test 1	N/A	Y	1.47	PASS
Multiple Queries Test 2	N/A	Y	2.30	PASS
Result →			6/6	

Test Data Conclusions

Our final prototype test consisted of three distinct test brackets. The primary metric we focused on was end-to-end completion time, particularly in comparison to our second prototype. For this round, we aimed for a response latency of under 4 seconds.

Overall, our final prototype passed all tests and successfully met the project's goals. We observed key performance improvements, particularly in handling specific queries that involved course IDs and names—for example, distinguishing between "When is EC 414?" and "When is machine learning?" The system also correctly identified and responded to course type distinctions, such as "When is EC 414 discussion?" versus "When is EC 311 lab?"

Another key success was our name detection and correction capability. For instance, the system correctly interpreted "Tally Moret" as "Tali Moreshet." In the third test bracket, which involved answering more complex and open-ended prompts (e.g., "How many discussion sections does EC414 have?"), the system still provided accurate and relevant answers. Among all test cases, the conversation properly terminated upon hearing the phrase, "Goodbye, BUtLAR."

Latency ranged from a minimum of 1.38 seconds to a maximum of 2.30 seconds—an improvement over earlier versions. While these results are promising, some refinements remain. Prior to ECE Day, we plan to implement UI enhancements such as a more intuitive interface with visual cues to indicate when the user should begin speaking, ensuring complete queries are captured. We are also considering a visual redesign, potentially incorporating a pulsing circle animation for a more futuristic aesthetic.

Finally, although our database and retrieval system have become more robust, we aim to further improve search accuracy by integrating VannaAI. This will help address inconsistencies we encountered—for example, when a query occasionally produced an incorrect response but succeeded upon being repeated.