



**Boston University**  
**Electrical & Computer Engineering**  
EC464 Capstone Senior Design Project

User's Manual

**BUtLAR: Boston University Large Language Model**  
**Auditory Responder**

Submitted to

Yobe Inc.  
77 Franklin St, Boston, MA 02110  
(617) 848 8922  
Email: [contact.us@yobeinc.com](mailto:contact.us@yobeinc.com)

by

Team 12  
BUtLAR



Noa Margolin [noam@bu.edu](mailto:noam@bu.edu)  
Suhani Mitra [suhanim@bu.edu](mailto:suhanim@bu.edu)  
Jackie Salamy [jesalamy@bu.edu](mailto:jesalamy@bu.edu)  
Andrew Sasamori [sasamori@bu.edu](mailto:sasamori@bu.edu)

Submitted: 4/18/25

# **BUtLAR User Manual**

## **Table of Contents**

<b>Executive Summary</b>	<b>3</b>
<b>1 Introduction</b>	<b>4</b>
<b>2 System Overview and Installation</b>	<b>5</b>
2.1 Overview Block Diagram	5
2.2 User Interface	5
2.3 Physical Description	6
2.4 Installation, Setup, and Support	6
<b>3 Operation of the Project</b>	<b>8</b>
3.1 Operating Mode 1: Normal Operation	8
3.2 Operating Mode 2: Abnormal Operations	9
3.3 Safety Issues	9
<b>4 Technical Background</b>	<b>10</b>
<b>5 Relevant Engineering Standards</b>	<b>15</b>
<b>6 Cost Breakdown</b>	<b>17</b>
<b>7 Appendices</b>	<b>18</b>
7.1 Appendix A - Specifications	18
7.2 Appendix B – Team Information	18

## Executive Summary

Existing chatbots present several limitations to human-machine interactions. They often require manual input, lack personalization, and struggle to provide accurate responses to complex queries. To address these challenges, we introduce the Boston University Large-Language-Model Auditory Responder (BUtLAR), a voice-powered digital human assistant designed to offer an interactive and personalized conversational experience.

BUtLAR has capabilities to seamlessly integrate features from Yobe Inc. that personalize audio processing through speaker identification, noise immunity, and answer generation in real-time. Starting with the Yobe Software Development Kit (SDK), this tool preserves voice data, such as denoising speech in a loud environment and identifying who the speaker is via biometric tracking. Speech-to-text transcription is then performed through API calls and passed to a Large-Language Model (LLM), which makes SQL queries to BUtLAR's database.

For our demonstration, a sample database contains structured information about Boston University, with an emphasis on ECE-related topics. The software is deployed on a Raspberry Pi 5 with a 2-way microphone, offering an intuitive interface and feedback loop. The demonstration runs on a locally hosted web page, enabling users to query the database and receive real-time responses directly through a search browser. By combining AI-driven speech processing, language modeling, and database interaction, BUtLAR sets up a framework to enhance voice-based human-computer interactions.

## 1 Introduction (Noa)

In the rapidly expanding field of human-machine interactions, many digital assistants still fail to simulate a natural conversation feel. They suffer from delayed responses, lack of personality, or overlook seemingly obvious contextual details. These limitations leave businesses and consumers wary of their adoption, despite the potential of voice-powered AI to facilitate natural conversations between humans and machines.

Our customer, Yobe Inc., is an audio processing start-up specializing in extracting speech from noisy environments to remove background noise through denoising and detection of who the speaker is via biometric markers. Through these advanced signal processing capabilities, Yobe intends to revolutionize voice AI technology. They requested that we develop a Voice Chatbot framework that seamlessly integrates their SDK to highlight its commercial and societal value.

To address this, we developed BUtLAR (Boston University Large-Language-Model Auditory Responder), an end-to-end pipeline integrating audio processing, response generation, and an intuitive user interface. The architecture is designed to function without Yobe's SDK, while enabling seamless integration to call their API—and future versions—demonstrating a “before” baseline and an “after” with enhanced naturalness and accuracy.

BUtLAR is an advanced Linux-based audio processing stack on the Raspberry Pi V5, utilizing two Røde Microphones for high-quality audio capture and a web interface for scalability and portability. BUtLAR activates with a single button click and processes conversations in real-time. It captures audio using C++'s miniaudio library, converts speech to text, retrieves data from a tailored database using SQL queries generated by Open AI's large-language-model (LLM), and responds with natural text-to-speech output, continuing until the user says “Goodbye BUtLAR” or is silent for too long.

Designed for flexibility, BUtLAR operates independently of Yobe's SDK but can integrate with it to enhance clarity and accuracy in noisy environments. Without this integration, BUtLAR can struggle with noisy settings, misinterpreting questions, and the inability to identify speakers. Additionally, BUtLAR's response quality depends entirely on the database and LLM prompting, limiting its effectiveness for diverse use cases—like university navigation, grocery stores, malls, or medical appointments—to the quality of human-engineered inputs. The absence of reinforcement learning capabilities further restricts its adaptability, requiring ongoing human development. Security risks emerge from vulnerabilities to malicious inputs, such as SQL injections, which could compromise the system. Moreover, like all LLMs, BUtLAR is prone to errors and should not be relied upon for critical applications, such as medical or emergency response, where inaccuracies could lead to serious safety issues.

Overall, BUtLAR serves as a framework to demonstrate Yobe's SDK potential to advance natural, accurate voice AI conversations, while highlighting its real-world applicability.

## 2 System Overview and Installation (Noa)

### 2.1 Overview Block Diagram

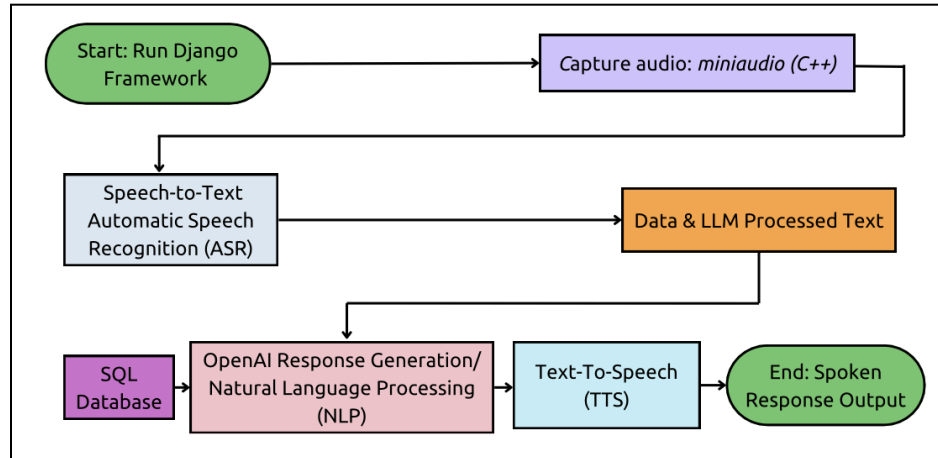


Figure 1: Software Block Diagram

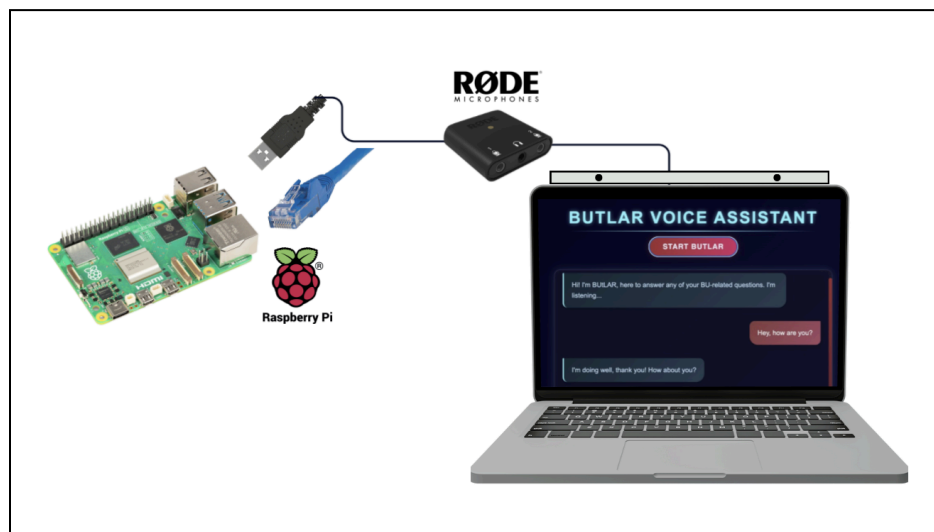


Figure 2: Hardware Block Diagram

### 2.2 User Interface

The user interface is web-based. *Figure 3* illustrates BUtLAR’s three interaction states: start, speaking, and listening/processing. To begin, the user clicks the “Start” button, triggering a “Ready to start...” notification. While BUtLAR speaks, a “BUtLAR is speaking” display advises the user to listen rather than speak, with a real-time chat transcription shown for reference. When BUtLAR is ready to receive input, a “BUtLAR is listening” display appears. The interface toggles between “BUtLAR is speaking” and “BUtLAR is listening” states during the conversation. The interaction ends when the user says “Goodbye BUtLAR,” terminating the session.

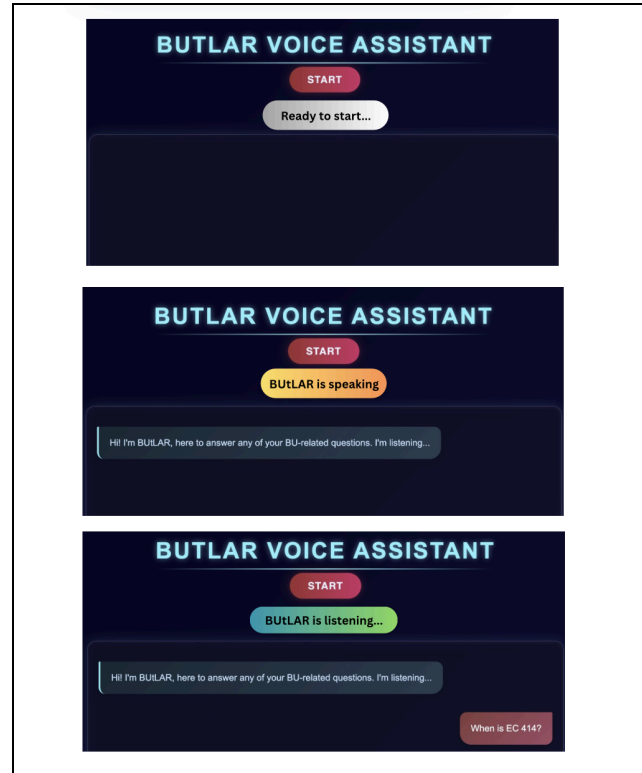


Figure 3: User-Interface, with a start button, a state display, and transcription

### 2.3 Physical Description

BUTLAR's hardware is designed for portability and high-quality audio interaction. It features two Rode microphones that enable precise signal processing. These microphones are mounted on a 3D-printed attachment that easily slides onto the top of a laptop or LCD screen/monitor, ensuring compatibility across various devices. Users speak directly into the microphones when interacting with BUTLAR. The system is powered by a Raspberry Pi, since Yobe's SDK only operates on a Linux device.

### 2.4 Installation, Setup, and Support

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 9 inches apart on the 3D-printed housing, and attached on top of the laptop, 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 above in *Figure 1* and *Figure 2*.

The backend workflow captures audio and processes it in real-time. Once audio is captured, Speechify'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 from the prompt-engineered 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 the Web Speech 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 1*.

Users should change the SQL database and LLM prompt based on their use-case-specific needs. BUtLAR is initialized with a BU database, but it can easily be changed for other applications like a grocery store, mall, or doctor's office. Once the user's database and LLM prompt are established, follow these steps to set up the Voice-Powered Digital Human Assistant:

### Raspberry Pi Connection

1. Connect two AI-Micro Rode Dual Speakers to the Raspberry Pi.
2. Connect the Raspberry Pi to the network using an Ethernet cable.
3. Run and pipe the live audio streaming files on the Raspberry Pi to enable audio processing.

### Server-Side Connection

1. Establish SSH connectivity to the Raspberry Pi for remote access using the command:  
`ssh yobe@128.197.180.176`
2. Navigate to the appropriate directory:  
`cd`  
`BUtLAR_Voice-Powered-Digital_Human_Assistant/Audio/testing`  
`_audio/django_top`

### Running the Session

1. Activate the GCloud virtual environment:  
`source ~/gcloudenv/bin/activate`
2. Start the server:  
`daphne -b 127.0.0.1 -p 8000 django_top.asgi:application`
3. Access the BUtLAR interface by navigating to:  
`http://127.0.0.1:8000/butlar/interface/`
4. Press Enter to begin the session.

Ensure all connections are secure and the Raspberry Pi is properly configured to run the Yobe SDK on a Linux device for optimal performance.

### 3 Operation of the Project (Suhani)

#### 3.1 Operating Mode 1: Normal Operation

##### Activation

1. To initiate a conversation with BUtLAR, the user must click “Start BUtLAR.”
  - a. This action triggers an activation message, prompting the user to introduce themselves. Multiple users may introduce themselves and carry on individual, independent conversation threads.
  - b. The user introduction period is intended to support voice recognition on Yobe’s SDK and to provide the Large Language Model (LLM) with contextual user information for personalized interactions.<sup>1</sup>
2. Once activated, a chat-style interface appears. This interface displays both the user’s transcript and BUtLAR’s responses.

##### In Session

1. The system supports simultaneous participation from multiple users within a single session, distinguishing between them by using voice profiles established during the activation stage.<sup>1</sup>
2. Users will know when to speak based on the visual status indicator. *Figure 3* from Section 2.2 illustrates this.
3. When a user asks a question, BUtLAR processes the input and retrieves a response.
  - a. If the query pertains to content in the pre-uploaded database, BUtLAR provides a relevant response. If the question is unrelated to the database, BUtLAR still attempts a coherent response using the LLM.
4. Regardless of whether the prompt relates to the data, BUtLAR will listen for the next query, enabling a seamless and natural back-and-forth conversation with the user.
5. Normal consequences include mistranscription of user input. Normal consequences also include a no-match response indicating that there was no content in the database that relates to the user's query.

##### Termination

A user can end a conversation with BUtLAR in one of two ways:

1. A user can say “Goodbye, BUtLAR!” and a session will automatically terminate.
2. If no audio is detected for more than 45 seconds, the session automatically ends.

---

<sup>1</sup> This feature is yet to be implemented.



### 3.2 *Operating Mode 2: Abnormal Operations*

BUtLAR has not been designed with explicit diagnostic or self-test modes. However, there are a few potential abnormal conditions and corresponding recovery steps.

Noncritical cases included a no-match query. When a user query does not match the database, BUtLAR will provide a general fallback response. This is expected behavior and not considered a failure state.

If the system becomes unresponsive or unstable, restart the backend server to restore normal functionality. No specialized user intervention is required beyond restarting the local server. A simple refresh or relaunch is usually sufficient.

### 3.3 *Safety Issues*

BUtLAR operates in a closed environment and answers only case-specific queries, minimizing exposure to external threats or malicious data inputs. The system is designed to ignore inappropriate content—specifically, it does not process or respond to curse words, ensuring a baseline of safe and respectful interaction.

It is important to note that responses are generated by an LLM. When using an LLM, there is always a possibility that it may occasionally provide inaccurate or misleading information and hallucinate data not present in the source database. As such, we advise users to exercise critical judgement and validate important information independently. Users should also be cautious when adding data to the database. Queries that are crafted to mimic SQL injection patterns could potentially retrieve sensitive data.

Finally, users must secure their API keys. An exposed key may result in unauthorized access or usage charges. (In *git*, you will not be allowed to commit private keys to a remote repository, so you would need to hide them.)

## 4 Technical Background (Andrew)

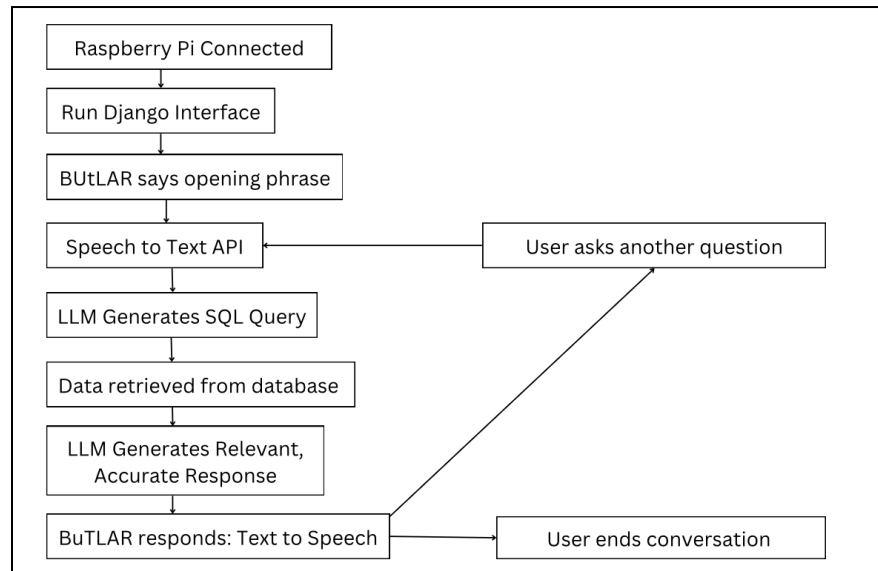


Figure 4: Software Flow

## Audio Pipeline

### Audio Capture:

BUTLAR begins operation when a user speaks into a stereo microphone system. The voice input is captured using a C++ library of Mini audio, set to a 16-bit PCM format at a 16 kHz sampling rate with a two-channel stereo configuration. This setup ensures that the system captures audio with sufficient clarity while maintaining a balance between processing efficiency and fidelity. Developers can choose to route the microphone input directly into the system through a pre-configured Bash script or use the provided C++ wrapper that interfaces with the microphone stream at a lower level. This wrapper enables real-time buffering, chunk-level control, and future expandability for audio-based triggers or keyword pre-processing.

### Noise Suppression & Speaker Verification (Optional):

If denoising and speaker identification are required, developers can integrate the Yobe SDK into the pipeline. The SDK supports audio enhancement by removing ambient interference and isolating voice features from registered users. It uses a template-matching architecture, where each registered user's voice pattern is saved and stored during an enrollment phase. During system operation, Yobe compares live audio input to the stored voiceprints to determine whether the input should be accepted or rejected. This step enables BUTLAR to function securely in shared spaces, preventing unauthorized users from prompting the assistant.

## Transcription & Voice Recognition

### Streaming Transcription:

After preprocessing, the enhanced or raw audio stream is transcribed using Speechmatics' Real-Time Speech-to-Text API. The system uses a streaming service to send continuous audio chunks and receive live interim and final transcription hypotheses. This streaming-based approach minimizes lag and allows the user to see their spoken words transcribed in near real time. The ASR engine also includes punctuation and formatting support to structure the transcript in a way that is optimal for parsing by the LLM downstream.

**Timeout & Latency Monitoring:**

To ensure responsiveness, the system runs an internal timeout thread that monitors idle time from the last transcript received. If the system detects a prolonged period of silence, it gracefully exits the session, resets any necessary flags, and provides an exit prompt to the user. All end-to-end latency is designed to remain under a *user-defined* number of seconds, with lightweight queries completing substantially faster.

**Retrieval Pipeline****Lightweight Retrieval Option:**

BUtLAR supports a fast-response path for frequently asked questions. This method loads a static knowledge base into memory and maps transcribed queries to predetermined responses. This approach is optimized for deployment in settings where network bandwidth is constrained or rapid query-response time is essential. Developers can update or expand a database through a simple .db (SQLite) file.

```

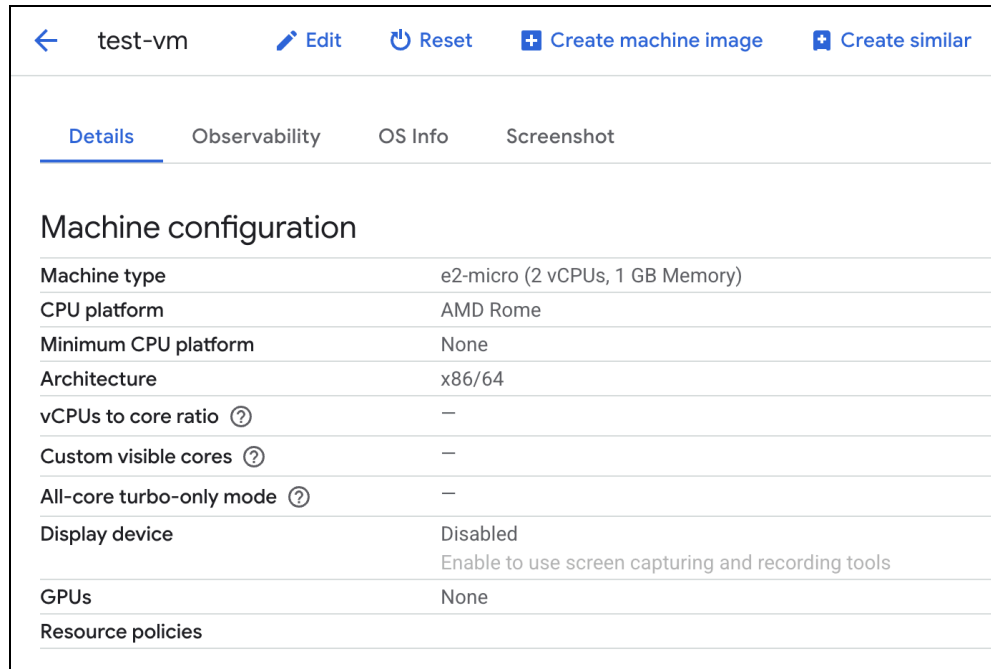
1  {
2    "New item - 2": {
3      "pageCount": 4,
4      "classes": [
5        {
6          "index": 1,
7          "crse_id": "124985",
8          "crse_offer_nbr": 1,
9          "class_section": "A1",
10         "start_dt": "01/21/2025",
11         "end_dt": "05/01/2025",
12         "campus_descr": "Boston University",
13         "class_nbr": 9634,
14         "component": "LEC",
15         "subject": "ENGEC",
16         "subject_descr": "ENGEC Electrical & Comp Engr",
17         "catalog_nbr": "311",
18         "class_type": "E",
19         "schedule_print": "Y",
20         "acad_group": "ENG",
21         "acad_org": "ENG",
22         "descr": "Introduction to Logic Design",
23         "instructors": [
24           {
25             "name": "Tali Moreshet",
26             "email": ""
27           }
28         ],
29         "section_type": "Lecture",
30         "meetings": [
31           {
32             "days": "TuTh",
33             "start_time": "09.00.00.000000",
34             "end_time": "10.45.00.000000",
35             "start_dt": "01/21/2025",
36             "end_dt": "05/01/2025",
37             "bldg_cd": "PHO",
38             "bldg_has_coordinates": false,
39             "facility_descr": "8 St. Mary's St PHO 210",
40             "room": "210",
41             "facility_id": "PHO 210",
42             "instructor": "Tali Moreshet"
43           }
44         ]
45       ]
46     }
47   }

```

Figure 5: Sample JSON snippet outlining a potential SQLite formatting for ECE course embeddings

### RAG with Vanna and OpenAI (Scalable Option):

For more complex or dynamic queries, BUtLAR leverages Retrieval-Augmented Generation (RAG) by interfacing with Vanna AI. In our implementation, a PostgreSQL database is deployed on a Google Cloud E2 micro virtual machine, enabling a free, persistent, cloud-hosted data backend. Vanna connects to the database, converts the user's natural-language question into SQL, and retrieves relevant rows. A semantic search layer powered by ChromaDB identifies data relevance even when exact keyword matches are absent.



The screenshot shows the Google Cloud console interface for a VM instance named 'test-vm'. At the top, there are navigation links: 'test-vm', 'Edit', 'Reset', 'Create machine image', and 'Create similar'. Below this is a tabbed interface with 'Details', 'Observability', 'OS Info', and 'Screenshot'. The 'Details' tab is active, displaying the 'Machine configuration' section. This section contains a table with various VM settings.

Machine configuration	
Machine type	e2-micro (2 vCPUs, 1 GB Memory)
CPU platform	AMD Rome
Minimum CPU platform	None
Architecture	x86/64
vCPUs to core ratio <sup>?</sup>	—
Custom visible cores <sup>?</sup>	—
All-core turbo-only mode <sup>?</sup>	—
Display device	Disabled Enable to use screen capturing and recording tools
GPUs	None
Resource policies	

Figure 6: VM details via Google Cloud

The SQL results are then interpreted by OpenAI's GPT-4o-mini through an API call. The prompt includes both the user's question and the tabular data extracted from the database, allowing the model to compose an intelligible and accurate response. This architecture allows BUtLAR to respond with institution-specific information while leveraging the LLM's broader linguistic and reasoning capabilities.

## Web Hosting & Frontend

### Django Server and Local Hosting:

BUtLAR is deployed through a Django web application hosted on the local machine. This server exposes an IP endpoint (e.g., `http://172.20.10.4:8000`) that allows any device on the same network to access the assistant. The frontend dynamically updates based on transcript data and system state, including waiting for queries, active listening, and response playback. Django's internal routing framework is used to maintain clean endpoint separation between the UI, backend scripts, and system flags.

### System Integration:

All backend services—transcription, audio handling, query processing, and response generation—are tightly integrated into the Django ecosystem. Developers can choose to host the server locally or expose it to a broader network using tools like ngrok or standard port forwarding. Static assets and audio interfaces are managed within Django's templating engine.

## Developer Integration

### Repository Inheritance:

Developers looking to build their own assistant systems can inherit the full BUtLAR repository. All key logic is modularized into reusable services, including real-time transcription handling, Vanna-based query processing, OpenAI integration, and web deployment. While the code is written primarily in Python, a C++ module is included to interface directly with audio input devices for low-latency streaming use cases.

### Customization Options:

- Developers may plug in their own database system and schema.
- OpenAI API keys are stored via environment variables and can be swapped for other LLM providers like Anthropic or Cohere.
- Prompt formatting, retrieval logic, and fallback paths are customizable through provided templates and scripts.
- Yobe SDK usage is optional, allowing flexibility for environments that do not require speaker verification.

### Core Technical Principles

- **Asynchronous Streaming:** The system supports live, non-blocking transcription to ensure minimal user latency.
- **Modular API Access:** All external services—Google ASR, OpenAI, Vanna—are wrapped in modular interfaces for easy mocking, replacement, or extension.
- **Voice-Based Access Control:** When enabled, Yobe provides robust speaker verification based on voiceprints instead of logins or biometrics.
- **Semantic Retrieval:** ChromaDB embeddings enable semantic alignment between the query and the database schema, even with vague or paraphrased language.
- **Scalable Deployment:** Designed for plug-and-play scalability, BUtLAR can operate on a Raspberry Pi or scale to a full server network depending on the deployment need.

## 5 Relevant Engineering Standards (Jackie)

The hardware components of BUtLAR align with several engineering standards. The Rode AI-Micro USB interfaces follow USB Audio Class specifications, which are derived from standards defined by the IEC (IEC 61938 for signal levels and IEC 62368-1 for device safety). The ISO/IEC 60958 standard also supports digital audio transmission used in USB audio interfaces. The Raspberry Pi's Ethernet connectivity follows IEEE 802.3, ensuring compliant network communication. Overall, the hardware system is built using components that adhere to standards set by ISO, IEC, and IEEE, ensuring compatibility, safety, and interoperability.

The system also operates in a POSIX-compliant environment, as the Raspberry Pi runs a Linux-based OS. POSIX (Portable Operating System Interface for Unix) is a standard defined by IEEE for maintaining compatibility across Unix-like systems. Through POSIX compliance, the system manages tasks like file I/O, real-time audio data processing, and device communication via ALSA (Advanced Linux Sound Architecture). These are all essential for capturing audio, processing user input, and generating responses. Following this compliance ensures that the system remains portable and compatible.

While BUtLAR currently uses a standard laptop for demonstration purposes, this setup still conforms to engineering standards. The laptop's display, speaker output, USB interfaces, and power supply are all manufactured to meet IEC and ISO requirements, such as IEC 60950 and ISO 9241. IEC 60950 is a safety standard that applies to information technology equipment such as laptops, ensuring they are designed and tested for safe use. ISO 9241 is a collection of ergonomic and usability standards that offer guidance on creating interactive systems (laptops) to improve user experience. With the usage of a laptop for demonstrating BUtLAR, these engineering standards are met. In future applications, the laptop could be replaced with an LCD screen and speakers, which would still fall under relevant IEC and ISO audio-visual standards, including IEC 60268 pertaining to audio signal processing and IEC 62368-1 for device safety.

Additionally, BUtLAR follows engineering standards on a software level. The Django web framework, which runs in a POSIX-compliant Linux environment (Raspberry Pi), ensures that the software adheres to IEEE POSIX standards for system calls and file I/O. Django provides a secure and modular backend structure and provides the frontend using HTML templates that strictly follow HTML syntax standards (i.e., always declare the document type, use lowercase element names, close all HTML elements, etc). These templates, along with supporting CSS, ensure that the user interface is accessible, responsive, and consistent across browsers.

BUtLAR utilizes REST APIs at multiple points in the system. Calls to third-party services like OpenAI and Speechmatics use RESTful endpoints with POST requests, Bearer token authentication, and structured data exchanges using JSON. These REST APIs are stateless and conform to W3C REST architectural constraints, enabling secure communication between the system components to build a smooth process in the BUtLAR connections.

Audio input is captured using the C++ miniaudio library, which interfaces with ALSA, the standard Linux audio subsystem. ALSA adheres to POSIX I/O and driver interface standards, serving as the interface between BUtLAR and using miniaudio and the connected USB Audio Class. Also, with

the context of audio, for the ASR, the system uses a secure Speech-to-Text API from Speechmatics. The audio is transmitted over TLS-encrypted HTTPS connections (IETF TLS 1.2/1.3 standards).

After transcription, the text is processed and through Open AI, it prompts the user's query to match the context of the question to an SQLite database using dynamic SQL queries. These queries conform to ISO/IEC 9075 (SQL) standards. The system outputs spoken responses using browser-based speech synthesis (via the W3C Web Speech API).

From a security standpoint, the Raspberry Pi uses SSH (Secure Shell) to manage system access. This follows IETF RFC 4251 and related specifications for encrypted remote communication.

BUTLAR also uses WebSocket communication (RFC 6455) between the browser and the Django backend for real-time interactivity, including transcript delivery and responses. This ensures low-latency communication alongside the stateless nature of the REST calls used throughout the system.

In addition to following software engineering standards, BUTLAR follows coding standards to maintain readability, consistency, and maintainability. Python scripts are written following PEP 8, the official Python style guide, as well as clear variable naming, proper indentation, and function definitions. All API interactions and logic are grouped into separate modules with clear names for each to follow an understanding of what each module accomplishes. Comments are included to clarify function behavior, and Git is used for version control with meaningful commit messages. This consistent coding approach improves the clarity and maintainability of the system.



## 6 Cost Breakdown (Jackie)

Project Costs for Production of Beta Version (Next Unit after Prototype)				
Item	Quantity	Description	Unit Cost	Extended Cost
1	1	Raspberry Pi 5 128 GB*	\$159.99	\$159.99
2	1	2 AI-Micro Rode Dual Speakers *	\$79	\$79
3	1	3D Printing PETG Filament	\$11	\$11
4	1	Macbook Laptop	\$1,599	\$1,599
5	per hour	Speech to Text Engine	\$0.30	N/A
Beta Version-Total Cost				\$1848.99

*Table 1: Items provided by our customer, Yobe.\**

The budget for BUtLAR consists of hardware and software costs. For the hardware, the costs for the Raspberry Pi 5 and Rode microphones are included. It is key to note that for the project, the client, Yobe, provided the Raspberry Pi 5 and two Rode microphones. Beyond these costs, for the microphone housing, PETG filament was used. Additionally, to display BUtLAR, a MacBook laptop from a team member was used for demonstration purposes; therefore, no extra costs for an LCD screen and speaker were needed. However, for future units, an LCD screen and speaker could be used as a more cost-efficient option. These alternative costs are around \$45.99 for the LCD screen and \$8.68 for the speakers.

For the software aspect of BUtLAR, the main component includes a speech-to-text engine. For demo purposes, the cost of the speech-to-text engine is not included in the final Beta version cost since the free subscription (with limited minutes allowed) was used. With Speechmatics, users under the free version get 8 hours free per month (4 hours batch + 4 hours real-time). However, for finalized product purposes, the cost would be a subscription per month. Speechmatics offers another plan of 8 hours free/month plus \$0.30 per hour for extra needed hours.

The overall cost of BUtLAR is relatively inexpensive when excluding the laptop cost, as that was simply for demonstration purposes. The final cost without the added laptop cost is approximately \$304.66 (with the LCD and speaker alternative to the laptop). However, if a modern display of BUtLAR is preferred, then the approximate cost is \$1848.99 (with the laptop). This cost is adjustable based on the different LCD screens or speakers used, as well as the subscription plan used for the speech-to-text engine, as the cost varies based on the plan.

## 7 Appendices

### 7.1 *Appendix A - Specifications*

Requirement	Value, range, tolerance, units
Microphone Stand Dimensions	9 in x 2.6 in x 0.29 in
Two-Microphone Distance	9 in.
Single Query Latency	< 5 sec
Duration of Silence for Timeout	45 sec
Cost of each BUtLAR product	< \$50

### 7.2 *Appendix B – Team Information*

Noa Margolin is graduating with a B.S. in Electrical Engineering. She is passionate about empowering people through consumer technology and digital health.

Suhani Mitra is graduating with a dual degree in Computer Engineering and Computer Science/Mathematics. She is interested in developing embedded systems and advancing autonomous technologies. Upon graduation, she will join Cisco as an ASIC Engineer II.

Jacqueline Salamy is graduating with a B.S. in Electrical Engineering. She has a passion for energy and sustainability. After graduation, she will join GE Vernova as an Electrical Engineer.

Andrew Sasamori is graduating with a degree in Computer Engineering and a concentration in Machine Learning.