GOYO

A Spatial Active Noise Control System for Smart Home Environments

Taerim Kim
Dept. Information System
Hanyang University
Seoul, Republic of Korea
trnara5375@gmail.com

Wongyu Lee
Dept. Information System
Hanyang University
Seoul, Republic of Korea
onew2370@hanyang.ac.kr

Junill Jang
Dept. Information System
Hanyang University
Seoul, Republic of Korea
jang1161@hanyang.ac.kr

Hoyoung Chung
Dept. Information System
Hanyang University
Seoul, Republic of Korea
sydney010716@gmail.com

***Abstract*—This study introduces GOYO, an AI-based spatial active noise control device designed for IoT-enabled smart homes. Modern living environments expose users to continuous low-frequency noise from appliances such as air conditioners, refrigerators, and air purifiers, which can impair concentration during studying, remote working, or reading.**

In the proposed framework, each household appliance is equipped with a reference microphone and an on-device AI model that classifies and broadcasts its noise characteristics over the IoT network. GOYO, positioned near the user, integrates these distributed noise signatures with the sound measured in the user's immediate vicinity to perform localized active noise cancellation.

By combining edge-level acoustic intelligence in home appliances with user-centered noise control provided by GOYO, the system offers an effective and privacy-preserving approach to reducing household noise. This integrated spatial noise management framework demonstrates the potential to enhance concentration and auditory comfort in everyday living spaces.

***Index Terms*—Real-Time Noise Analysis, Active Noise Control, Internet of Things, Smart Home, AI**

Roles	Name	Task description and etc.
User/Customer	Wongyu Lee	The User/Customer seeks to enhance daily life through intuitive smart home technology. They interact via voice or text commands, with the system recognizing emotions and context. They want a service that simplifies device control and creates a responsive environment that improves comfort and efficiency with minimal effort.
AI Developer	Taerim Kim	An AI Developer builds a multimodal service that generates optimal smart home routines. They design and implement the AI architecture, including text generation and speech recognition with emotion and context analysis. They apply transfer learning, ensure library integration, and monitor the system to continuously improve recommendation accuracy and adaptability.
Software developer	Junill Jang	A Software Developer transforms project requirements into functional, high-quality applications. Working with the team, they design scalable architectures, write maintainable code, and handle testing, debugging, and quality assurance. They also document processes and functionalities to support future updates and ensure project continuity.
Development manager	Hoyoung Chung	A Development Manager oversees the software development lifecycle, setting goals and methodologies to ensure timely, high-quality deliverables. They coordinate communication between clients and developers and align cross-functional efforts. Through oversight of design, development, and testing, they ensure products meet user requirements and follow best practices.

I. Introduction

A. Motivation

In modern smart home environments, activities that demand sustained concentration—such as remote work, studying, reading, and immersive entertainment—have become increasingly common. However, continuous background noise generated by household appliances like air conditioners, refrigerators, and water purifiers can significantly reduce users' comfort and focus, often leading to auditory fatigue over time. Conventional passive noise reduction methods, such as soundproofing materials or ear-covering devices, are often impractical for open or shared living spaces.

To address this challenge, active noise control (ANC) technologies offer a promising alternative. Similar techniques have already been successfully applied in automotive systems, where adaptive filtering algorithms and real-time acoustic analysis are used to suppress engine and road noise within confined spaces. These vehicle-based ANC solutions demonstrate how dynamic and environment-aware control can effectively mitigate unwanted sound.

Inspired by these developments, this study aims to explore how similar ANC principles—augmented with adaptive and AI-based algorithms—can be applied to indoor environments. Unlike vehicle cabins, residential spaces exhibit diverse acoustic reflections, complex sound propagation paths, and structural vibrations, making noise control inherently more challenging. By extending ANC methodologies from the automotive domain to everyday living environments, this research seeks to develop an intelligent, spatially adaptive system for real-time household noise suppression.

B. Problem Statement (Client's Needs)

Despite the growing demand for quiet and focused environments in modern homes, current noise management solutions remain limited. Passive soundproofing materials are often costly, space-inefficient, and ineffective against low-frequency noise generated by household appliances. Moreover, personal noise-cancelling devices such as headphones or earphones isolate the user from their surroundings, making them unsuitable for long-term use or shared spaces.

There is therefore a clear need for an intelligent, space-oriented noise control system capable of automatically detecting, analyzing, and reducing unwanted noise without physically isolating the user. Such a system should adapt to varying acoustic conditions, preserve natural sound cues (e.g., speech or media playback), and seamlessly integrate into daily life through IoT connectivity and AI-driven control.

C. Definition of Key Terms

- **Noise Cancellation:** Pass noise through a filter that tends to suppress the noise while leaving the signal relatively unchanged. [1]
- **Active Noise Control (ANC):** A technique that reduces unwanted sound by generating a secondary sound field that destructively interferes with the primary noise. It exploits the long wavelengths of low-frequency noise by electronically controlling secondary sources such as loudspeakers to produce anti-noise signals. [2]
- **Digital Signal Processing (DSP):** The manipulation of digital or digitized signals using digital technologies. DSP systems are implemented through mathematical abstractions, software, or hardware, often under strict real-time and precision requirements, demanding integrated knowledge of signal theory and technology. [3]
- **Second Path:** Entire transfer path between the secondary loudspeaker and the error microphone, including electronic and acoustic components such as the D/A converter, power amplifier, acoustic propagation in air, the microphone, and the A/D converter. Modeling this path accurately is essential, as it determines how the anti-noise signal is physically transformed before reaching the listener. [4]
- **Least Mean Squares (LMS) Algoritm:** An adaptive filtering method that iteratively updates filter coefficients to minimize the mean square error between the desired and actual signals. It uses a gradient-descent approach to adjust weights based on the instantaneous error, making it simple, efficient, and widely applicable in real-time signal processing and noise cancellation systems. [5]
- **FxLMS Algorithm:** A gradient-based algorithm used to identify an unknown system, such as an ANC controller, in the presence of a secondary path. It differs from the standard LMS algorithm in that the reference signal is filtered through an estimated secondary path model before adaptation, allowing compensation for the secondary path effect. [6]

D. Research on Related Software

Several existing software and systems demonstrate the application of AI and IoT integration for noise management and environmental optimization.

1) **Hyundai Motor's In-Vehicle ANC System**
   Hyundai Motor's In-Vehicle ANC system actively reduces unwanted cabin noise by generating phase-inverted sound waves through the car's audio speakers. Unlike passive insulation, it uses DSP and adaptive filtering algorithms—particularly FxLMS method—to analyze and counteract noise in real time. Microphones installed throughout the cabin capture sounds generated by the engine, road surface, and wind, which are then processed by a central controller. The system continuously adjusts its output based on environmental factors such as vehicle speed, road type, and cabin conditions. Hyundai has further advanced this technology through its Road Noise Active Noise Control (RANC) system, which integrates accelerometers to predict and cancel vibration-induced noise before it reaches the cabin. This approach demonstrates how AI-assisted adaptive control can enhance in-vehicle acoustic comfort and serves as a foundation for extending ANC principles to smart home environments. [7]
2) **Apple AirPods ANC System**
   Apple's AirPods employ an ANC system that minimizes unwanted ambient sounds through real-time signal processing. The system uses outward-facing microphones to detect environmental noise and inward-facing microphones to monitor the sound inside the ear canal. Based on this information, the internal processor generates phase-inverted sound waves that effectively cancel the incoming noise through destructive interference. The algorithm continuously adapts to changes in the acoustic environment—such as movement, wind, or variations in ear fit—ensuring stable noise reduction performance across various conditions. This approach demonstrates how compact wearable devices can achieve efficient, adaptive noise control by integrating real-time sound analysis and feedback-based signal adjustment.
3) **LG ThinQ**
   LG ThinQ is LG's AI-powered smart home platform that connects, monitors, and controls a wide range of household appliances through a unified app interface. The platform allows users not only to view the real-time status of devices but also to remotely operate them, receive alerts, and manage multiple products within an integrated ecosystem. It supports automated routines—such as scheduling air conditioner operation, adjusting refrigerator modes, or initiating laundry cycles—that optimize daily tasks and reduce manual effort. In addition, LG ThinQ leverages user behavior patterns, environmental data, and energy consumption insights to recommend more efficient operating modes and help reduce overall power usage. By incorporating AI features such as pre-

dictive maintenance, anomaly detection, and contextual suggestions, the platform ensures that appliances function reliably and adapt to the user's lifestyle. Through the seamless combination of AI and IoT technologies, LG ThinQ enhances convenience, operational efficiency, and personalized home management across diverse smart home environments.

II. Requirements

A. Common Features

1) **Sign Up**
The registration process requires users to provide an email address (serving as the user ID), phone number, password, and desired nickname. Passwords must contain at least 8 characters, incorporating a combination of letters, numbers, and special characters. The chosen nickname becomes the user's default display name within the application. Upon registration, a verification email is sent to activate the account before login access is granted.
2) **Log in**
The system supports standard authentication using email and password credentials. Invalid login attempts trigger clear error messages to guide users. Successful authentication redirects users to the Main page.
3) **Account Recovery**
The system provides an account recovery feature that allows users to retrieve their ID and reset their password. To find their ID, users verify their phone number, after which the registered email address is displayed. For password recovery, if the entered email exists in the system, a verification code is sent to that email, and upon successful verification, users can proceed to reset their password.

B. In-Application Features

1) **Splash Screen**
Upon application launch, a splash screen displaying the system logo against a minimalist background appears while the system initializes device connections and loads user configurations.
2) **Main Page**
The application opens with the Home page as the default view. This page includes a Start/Stop button for controlling the main functionality and provides an overview of the current sound environment detected by connected external microphones. The interface displays a live frequency spectrum and noise suppression graph, allowing users to visualize ambient sound levels and system performance. Users can enable or disable noise reduction for each detected sound and adjust the suppression intensity. Each noise type can also be deleted, and all changes are reflected in real time. A menu bar provides quick access to three main sections of the application.
   - **Home:** This button allows users to navigate from other pages to the main page.
   - **Device Management:** Lists connected GOYO smart chair with connection controls.
   - **Profile:** Manages user settings.
3) **Device Management**
This page enables users to register, monitor, and manage IoT devices. In GOYO, the smart chair functions as the main IoT device and is detected automatically via Wi-Fi. Users can access its status, run basic diagnostics, and disconnect the device if required.
4) **Profile Page**
Users can view and update personal information, including their name. The application stores user-specific ANC configurations and maintains data logs of noise patterns, suppression statistics, and device usage history.

C. AI Features

1) **Appliance Sound Source Identification**
Each household appliance is equipped with a reference microphone that continuously captures local acoustic signals. The AI module analyzes these signals to distinguish whether the incoming sound originates from the appliance itself or from external environmental noise. This classification is essential for ensuring that only valid appliance-generated noise is forwarded to the central ANC system.
2) **Distributed Noise Event Aggregation**
After a sound is identified as originating from the appliance itself, the AI module determines whether it constitutes a meaningful noise event that contributes to the user's acoustic environment. Valid noise events detected by each appliance-side AI module are then transmitted to a central processing unit, where the signals from multiple appliances are integrated to construct a real-time acoustic map of the home.

D. ANC Features

The system shall include a real-time DSP module responsible for noise analysis, anti-noise signal generation, and adaptive control. Incoming audio from connected microphones is processed with minimal latency to identify dominant noise frequencies and generate corresponding anti-phase signals for ANC. The DSP module continuously monitors residual noise through feedback and dynamically adjusts signal parameters to maintain optimal suppression performance.

III. Development environment

A. Choice of Software Development Platform

Development Platform:

1) **macOS**
macOS is Apple's operating system providing a robust development environment centered around Xcode for Apple ecosystem development. Its Unix foundation offers powerful command-line capabilities, while package managers like Homebrew facilitate tool management. The platform excels in native development through Swift and Objective-C support, and includes comprehensive debugging and performance optimization tools. macOS

integrates smoothly with Apple services like iCloud and TestFlight for deployment, while maintaining security through features like Gatekeeper. Its UNIX certification and virtual machine support enable versatility across different development scenarios.

2) **Raspberry Pi OS**
Raspberry Pi OS is a lightweight Linux-based operating system optimized for the Raspberry Pi's ARM architecture, providing a flexible environment for hardware-software integration and embedded development. Its Debian foundation offers powerful command-line tools and access to vast open-source packages, while GPIO libraries enable direct interaction with sensors, motors, and other peripherals. The platform supports programming languages like Python and C/C++, making it ideal for IoT, robotics, and educational projects. Raspberry Pi OS includes built-in tools for remote access, system configuration, and network setup, and its low-cost hardware ecosystem encourages rapid prototyping and experimentation across a wide range of development scenarios.
3) **AWS EC2**
AWS EC2 (Elastic Compute Cloud) is a scalable virtual computing platform that provides resizable compute capacity in the cloud. It allows developers to deploy and run applications on customizable virtual machines with full control over the operating system, networking, and storage configurations. EC2 supports a wide range of instance types optimized for compute, memory, or storage-intensive workloads, enabling flexible resource allocation based on application requirements. It offers features such as auto-scaling, load balancing, security groups, and VPC integration, making it suitable for building reliable and secure cloud-based systems. With its pay-as-you-go pricing model and high availability across multiple regions and availability zones, EC2 serves as an ideal platform for hosting backend services, microservices, distributed applications, and development environments in modern cloud architectures.

Language and Framework:

1) **Python**
Python is a versatile and widely used high-level programming language, praised for its simplicity and readability. This makes it particularly attractive for both beginners and experienced developers. Python's extensive standard library and rich ecosystem of third-party libraries provide powerful tools for various tasks, including web development, data analysis, and artificial intelligence. The language's strong support for object-oriented, imperative, and functional programming paradigms allows developers to choose the style that best fits their needs. Furthermore, Python is heavily utilized in the AI community due to its robust frameworks and libraries that facilitate tasks such as data preprocessing, model building, and evaluation.
2) **Flutter**
Flutter serves as a robust and versatile framework for mobile application development, enabling the creation of high-performance applications on both iOS and Android platforms from a single codebase. Its rich set of pre-designed widgets allows developers to implement highly customizable and visually appealing user interfaces efficiently. Flutter's hot-reload feature accelerates the development cycle by instantly reflecting code changes, thereby enhancing productivity. Moreover, Flutter integrates seamlessly with native APIs and third-party packages, providing flexibility to incorporate diverse functionalities. Extensive community support and comprehensive documentation ensure that development challenges can be quickly addressed, making Flutter an ideal choice for cross-platform mobile development.
3) **FastAPI**
FastAPI is a modern, high-performance web framework for building APIs with Python, designed to enable fast development and high efficiency. It leverages Python's type hints to provide automatic data validation, serialization, and comprehensive API documentation through OpenAPI and Swagger. FastAPI supports asynchronous programming using Python's async and await syntax, allowing for scalable and non-blocking request handling. Its simplicity, combined with robust performance comparable to Node.js and Go, accelerates backend development while maintaining reliability. Furthermore, FastAPI integrates seamlessly with popular Python libraries, database ORMs, and authentication systems, offering flexibility and extensive community support to quickly resolve development challenges.
4) **Flask**
Flask is a lightweight and flexible Python web framework designed for building web applications and APIs with minimal overhead. It follows a micro-framework philosophy, providing only the core essentials such as routing, request handling, and template rendering, while allowing developers to extend functionality through a wide ecosystem of third-party extensions. Flask's simplicity and unopinionated structure make it easy to learn and highly customizable, enabling developers to design architectures tailored to their specific use cases. Despite its minimal core, Flask supports integration with ORMs, authentication modules, and modern tooling, making it suitable for projects ranging from small prototypes to production-grade services. Its strong community, clear documentation, and flexible design continue to make Flask a popular choice for Python-based backend development.

Cost estimation:

Item	**Price**
Laptop	$1,199
Raspberry Pi	$100
Speaker	$9
Microphone	$9
EC2 Instances (t3.small x2)	$30 (per month)

Development Environment:

1) On Local Machine

Name	Computer Resource
Taerim Kim	Apple M2 8GB RAM
Wongyu Lee	Apple M4 16GB RAM
Junill Jang	Apple M2 8GB RAM
Hoyoung Chung	Apple M3 8GM RAM Ubuntu 24.04 / AMD 5600x, rtx3070 8gb, 32GB RAM

2) Cloud Platform

Purpose	Computer Resource
Back-end server deployment	AWS EC2 (t3.small) 2 vCPU 2GB RAM Ubuntu 20.04
AI server deployment	AWS EC2 (t3.small) 2 vCPU 2GB RAM Ubuntu 20.04

B. Software in use

1) **Visual Studio Code**
Visual Studio Code (VS Code) is a powerful, open-source code editor developed by Microsoft. It supports a wide range of programming languages and is highly extensible through its rich marketplace of plugins and extensions. VS Code provides an integrated terminal, debugging tools, and Git version control, making it an ideal environment for collaborative development. Features like IntelliSense offer smart code completion and context-aware suggestions, enhancing developer productivity. Its user-friendly interface and customizable settings allow developers to tailor their workspace, facilitating efficient and streamlined coding practices.
2) **Figma**
Figma serves as the core tool for our project's UI/UX design. The latest web-based version provides real-time collaboration, enabling multiple team members to work simultaneously. Figma allows us to manage the entire design process on a single platform, from wireframe creation to detailed prototype production. Its design system management features help maintain consistent UI elements and facilitate easy extraction of CSS values and assets during the transition from design to development. Moreover, Figma's component-based approach aligns seamlessly with React Native's component structure, ensuring an efficient workflow from design to implementation.
3) **PostgreSQL**
PostgreSQL is a widely used open-source relational database management system renowned for standards compliance and extensibility. It delivers robust data management with advanced SQL, full ACID transactions, and stored procedures/functions. PostgreSQL supports built-in replication and high availability options, along with strong security features such as SSL/TLS encryption, role-based access control, and row-level security. Its rich indexing, cost-based optimizer, and effective caching enable high-performance workloads at scale. Native JSONB, arrays, and user-defined types/extensions make PostgreSQL a versatile choice for applications requiring reliable, persistent, and flexible data storage.
4) **Docker**
Docker is an open-source containerization platform that enables developers to package applications and their dependencies into lightweight, portable containers. These containers provide a consistent runtime environment across different machines, ensuring reliable deployment from development to production. Docker uses isolated filesystem and resource environments, allowing multiple services to run securely on the same host without conflicts. It supports features such as image versioning, layered file systems, container orchestration, and automated builds, making it highly efficient for developing microservices and distributed systems. Its portability, scalability, and reproducibility make Docker an ideal choice for modern application development, CI/CD pipelines, and managing complex multi-service architectures.
5) **TablePlus**
TablePlus is a modern, native database management tool that provides a streamlined interface for interacting with multiple relational and non-relational databases. It supports databases such as MySQL, PostgreSQL, SQLite, Redis, and more. TablePlus offers features like syntax highlighting, query editor, table browsing, and secure connections, enabling developers to efficiently manage database structures and perform data operations. Its intuitive interface and fast performance make it a convenient choice for both development and database administration tasks.
6) **GitHub**
GitHub is a widely-used web-based platform for version control and collaborative software development. It provides Git repository hosting, enabling developers to track changes, manage code versions, and coordinate work across teams efficiently. Features such as pull requests, code reviews, and issue tracking facilitate seamless collaboration, while GitHub Actions allows automated workflows for testing, building, and deployment. The platform's extensive community support and integration with numerous development tools make it an essential resource for modern software projects.
7) **Notion**
Notion is an all-in-one workspace platform that combines note-taking, task management, and team collaboration tools. It enables teams to organize projects, documentation, and workflows within a single, flexible interface. Users can create databases, kanban boards, calendars, and wikis, facilitating both personal and team productivity. Notion's real-time collaboration features allow multiple team members to edit and comment simultaneously, ensuring smooth communication and coordination. Its customizable templates and integration with other tools make it a versatile solution for project management and knowledge sharing.
8) **Overleaf**

Overleaf is a cloud-based LaTeX editor that facilitates collaborative document creation and editing. It provides real-time collaboration, version control, and seamless compilation of LaTeX documents without the need for local installation. Overleaf offers a rich set of templates for academic papers, reports, and presentations, streamlining the document preparation process. Its integrated PDF preview and error highlighting features help users quickly identify and correct issues, while collaboration tools allow multiple contributors to work simultaneously, making it ideal for team-based research and technical writing projects.

C. Task distribution

Roles	Name	Task description and etc.
Front-end Developer	Wongyu Lee	Responsible for implementing the user interface and user experience of applications. They work with technologies like HTML, CSS, JavaScript, and frameworks such as React or Flutter to create responsive, interactive, and visually appealing designs. Frontend developers ensure seamless integration with backend services and optimize performance for various devices.
Back-end Developer	Junill Jang	Handles server-side logic, databases, and application infrastructure. They design and implement APIs, manage data storage, ensure security and scalability, and maintain server performance. Backend developers work with frameworks like FastAPI, Spring Boot, or Node.js to provide reliable and efficient services for frontend applications.
AI Developer	Taerim Kim	Develops and deploys artificial intelligence and machine learning models to solve specific problems or enhance application functionality. They preprocess data, train models, optimize algorithms, and integrate AI solutions into applications. AI engineers often work with frameworks like TensorFlow, PyTorch, or scikit-learn.
Data Engineer	Hoyoung Chung	Designs, builds, and maintains data pipelines and infrastructure for collecting, processing, and storing large volumes of data. They ensure data quality, accessibility, and scalability, supporting analytics and AI workflows. Data engineers work with databases, ETL tools, and cloud platforms to enable reliable data-driven decision-making.

IV. Specifications

A. Common Features

1) Sign Up

Fig. 1: Sign-up

ID	Name	Description
01	Signup-Page	GOYO requires four user information to sigh up for membership: E-mail, phone number, password, and name.
02	Signup-Email	The Email field serves as the unique identifier (primary key) for user login. The entered value must follow a valid email format (e.g., contain '@' and domain). Upon submission, the system checks the PostgreSQL database for duplicate entries. If found, an error message such as "This email is already in use" is displayed. Invalid formats prompt "Please enter a valid email address."
03	Signup-Password	The Password field requires at least 8 characters, combining two or more of the following: letters, numbers, and special symbols. During input, the password should be displayed on the screen in the format of asterisks '****'.
04	Signup-Name	The Name field is a mandatory input representing the user's real name. Upon successful registration, this value is stored in the users table under the name column and used as the user's default nickname within the system. Input validation ensures that the field cannot be left empty.
05	Signup-PhoneNum	The mobile phone number is mandatory for user verification. In case user forget their login information, they can restore account access through mobile identity authentication.

2) Login

ID	Name	Description
06	Login-Page	The Login page provides access control for registered users of the GOYO system. It includes two primary input fields (Email and Password), one action button (Sign In), and a navigation link to the Sign-Up page. Two account recovery buttons are also provided: Find ID and Reset Password.
07	Login-Success	When both ID and password are correctly entered and match an existing entry in the user DB, the login is successful and the backend returns an HTTP 200 response. The user is then redirected to the main homepage.

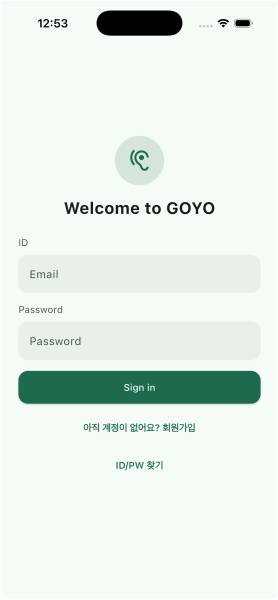
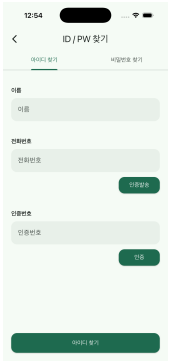


Fig. 2: Login

ID	Name	Description
08	Login-Failure	If the entered login information does not match any existing record in the user DB, the system will deny access and display a "Enter your ID/PW again" message.

3) Account Recovery



(a) Recovery ID

(b) Recovery Password

Fig. 3: Account recovery screens of GOYO app

ID	Name	Description
09	RecoverID-Page	The ID Recovery page allows users to retrieve their registered account ID by verifying their personal information. It includes two primary input fields (Name and Phone Number) and an action button (Send Verification Code). Once a valid user is found, an additional input field (Verification Code) appears for SMS verification. After successful verification, the user's registered ID (email address) is displayed on the screen. If the information does not match any record, an error message "User information not found" is shown.
10	ResetPswd-Page	The Password Reset page allows users to verify their identity before resetting their account password. It includes two primary input fields (Email and Phone Number) and an action button (Send Verification Code). Once a valid user is identified, an additional input field (Verification Code) appears for SMS verification. After successful verification, the user is redirected to the password reset page to create a new password. If the provided information does not match any record, an error message "User information not found" is displayed.
11	ResetPswd-Reset	The Password Reset (New Password) page allows verified users to create a new account password. It includes two input fields (New Password and Confirm Password) to ensure accuracy. The password creation rules are identical to those on the Sign-Up page but additionally, users cannot reuse their previous password for security reasons. If the confirmation does not match or the new password violates the rules, an appropriate error message is displayed.

B. In-Application Features

1) Main Page

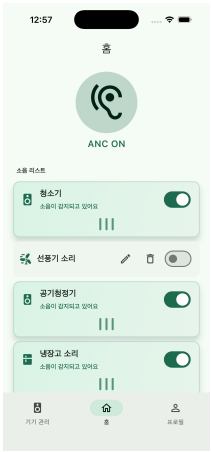


Fig. 4: Home

- Active Noise Control
  The Active Noise Control witch allows users to globally enable or disable AI-based real-time noise suppression. When toggled ON, the system activates the embedded model that classifies ambient sound from connected microphones and generates anti-phase signals to reduce low-frequency noise. When OFF, all suppression processes are paused to conserve resources. The switch's label dynamically displays the current ANC status ("ON" / "OFF").
- Noise Rules
  The Noise Rules section displays a list of IoT devices (e.g., refrigerator, air conditioner, fan), rather than generic noise types. Each device contains a built-in reference microphone that monitors its operational sound. When the AI engine detects that the noise level from a specific device exceeds a defined threshold, that device appears in the Noise Rules list as a detectable noise source. These baseline devices are included by default

to represent common household appliances within the environment..

- Add Noise Rule
  The Add (+) function is triggered when an IoT device detects a new noise pattern through its built-in microphone. Once ANC is activated, the system continuously monitors ambient sound at the device level, and when an unregistered noise is identified, the UI displays a dialog showing both the device name and the detected noise type. Users can choose to add this noise to their personal Noise Rules list by selecting the "Add" button. This mechanism allows the GOYO system to learn from each IoT device's environment while giving users direct control over which noises should be managed by ANC.
- Noise Rule Toggle
  Each noise rule includes an on/off toggle switch that determines whether the AI should recognize and suppress that particular noise type. When the switch is ON, the system monitors incoming audio frames for the rule's frequency band and activates adaptive filtering accordingly. When OFF, the AI model ignores that category in classification.
- Noise Rule Edit
  The Edit (pencil icon) function allows users to modify the rule's name. The updated configuration is applied immediately to the current ANC session.
- Noise Rule Delete
  The Delete (trash icon) function permanently removes a noise rule from the user's configuration. Upon deletion, the rule is excluded from future classification cycles and the local settings are updated to reflect the change. The system requests user confirmation before executing deletion.

2) **Device Manager**

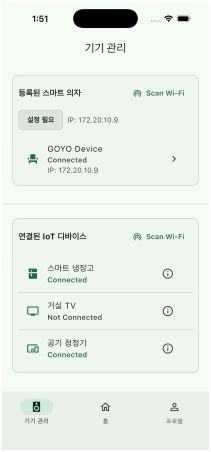


Fig. 5: DeviceManager

- Device manager page
  The Device Manager page provides an interface for registering, monitoring, and controlling all devices connected to the GOYO system. IoT devices are detected automatically via local Wi-Fi. Users can also pair devices manually by entering their unique device IDs. Real-time device status (Connected, Active, or Disconnected) is displayed for each entry.
- Scan device
  The Scan Devices button initiates a discovery process for nearby Smart Chairs using multicast DNS (mDNS) over Wi-Fi. Detected devices are listed with their connection information, and users can select a Smart Chair to establish a secure pairing connection. Once connected, the system continuously monitors the device's availability and connection status to ensure reliable operation.
- Device info
  Tapping a device item opens a detailed information panel showing device name, type connection protocol, and latency in milliseconds. From this view, users can perform actions such as testing audio input/output or checking signal stability.
- Rename device
  The Edit (pencil icon) function allows users to rename a device for easier identification (e.g., "GOYO Chair", "GOYO A/C"). Updated names are stored locally and synchronized with the PostgreSQL devices table, ensuring consistent labeling across sessions.
- Delete device
  The Delete (trash icon) function permanently removes a device from the paired list. Upon confirmation, the device entry is deleted from both the local cache and backend database. A confirmation dialog prevents accidental deletions.

3) **Profile**

Fig. 6: Profile

- Profile page
  The Profile Page allows users to view and modify personal information and configure their preferred sound environments. Each user's ANC (Active Noise Control) settings and usage history are managed indepen-

dently, providing a personalized listening experience optimized for two activity modes: *Focus Mode* and *Normal Mode*.

- User info
  The User Information section displays the user's registered name and offers the ability to edit it if needed. When a new name is entered and saved, the updated value is stored in the PostgreSQL users table and immediately reflected throughout the application interface.

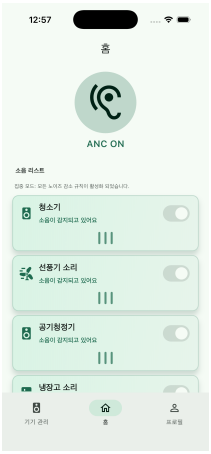


Fig. 7: FocusMode

- Sound mode
  The Preferred Sound Mode section lets users choose among three predefined environments: *Focus Mode*, *Normal Mode*, and *Favorites Mode*. In *Focus Mode*, the system turns **ON** suppression for **all** noise rules in the list, enabling maximum coverage across all detected IoT devices. In *Normal Mode*, only the **user-selected** noise rules remain enabled, following each rule's individual toggle state.

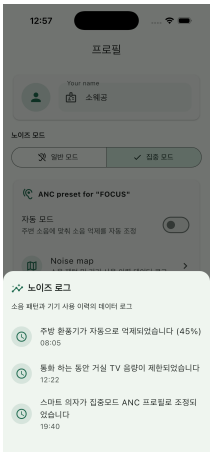


Fig. 8: NoiseLog

- User stats
  Usage & Noise Stats section summarizes past noise analysis data, including detected noise patterns, suppression performance, and device usage duration. This information is stored in the usage_logs table and can be displayed as visual graphs to help users understand how ANC settings performed over time.
- Save changes
  The Save Changes button confirms and applies all modifications made within the Profile Page. Successful updates trigger a confirmation toast message ("Profile updated successfully").

C. AI Features

1) **Core Model Construction**
   The existing YAMNet model, while robust, is not optimized for the specific, granular distinctions required by our ANC control logic. We utilize Transfer Learning to strategically re-purpose its feature extraction capabilities, allowing us to tailor the model to our custom classification ontology. Specifically, we focuse the classification set on various household appliance noises, emphasizing the need to accurately identify and categorize localized acoustic events (e.g., distinguishing refrigerator hum from general ambient hum).
   - **Feature Extractor:** The build_finetuned_model function defines a custom YAMNetLayer. This layer loads YAMNet's original TF function via hub.load() and sets trainable=False, freezing YAMNet's existing knowledge (millions of parameters) to reduce unnecessary training time.
   - **Batch Processing:** The YAMNetLayer's call function uses tf.map_fn to map the batch data (Batch Data) passed by Keras into 'single' data instances that the YAMNet function can process.
   - **Embedding Extraction:** Of the three outputs returned by YAMNet ([scores, embeddings, spectrogram]), only the second item, embeddings (1024 vectors), is extracted.
   - **Classifier Attachment:** The build_finetuned_model function uses tf.keras.Model to take the embeddings output from the frozen YAMNetLayer, flattens it with a Flatten layer, and connects it to a new Dense() layer that we defined. This layer is set to trainable=True and is the only part trained on our custom dataset.

2) **Dataset Construction and Model Training**
   The training protocol is designed to mitigate inherent data quality issues and address severe class imbalance, maximizing the utility of the YAMNet feature extractor.
   - **Data Acquisition and Cleaning:** The custom dataset is collected primarily from Freesound.org using the Freesound API. All files are strictly filtered to adhere to Creative Commons Zero (CC0) or Attribution (CC-BY) licensing requirements.
     – *Target Classes:* The dataset includes eight specific target appliance/event classes (e.g., Microwave, Refrigerator, Vacuum, Construction, etc.).

– *The Rejection Class (Others):* To prevent False Positives in the deployed system, the highly-imbalanced Others class was reduced via Under-sampling to diverse samples (e.g., speeching, water running). This cleaned class serves as the essential rejection boundary for non-target ambient noise.

- **Imbalance Management and Robustness:** A dual compensation strategy is applied to handle class imbalance between major and minor classes:
  – *Aggressive Augmentation:* Minority classes are oversampled through high–probability, strong augmentations such as Noise Injection, Pitch Shift, Time Masking, and Frequency Masking. This prevents Overfitting by increasing sample diversity.
  – *Class Weighting:* A balanced weight matrix is computed using the original unique sample counts and applied to the CrossEntropyLoss. This increases the penalty for errors on under–represented classes, improving minority-class learning.
- **Training Protocol:** The model is trained using Adam over 100 epochs following a two-phase fine-tuning schedule:
  – *Phase 1 (Warm-up, Epochs 1–20):* Only the classifier head is trained while the YAMNet backbone remains frozen, using an initial learning rate of 1×10^{-5}.
  – *Phase 2 (Fine-tuning, Epochs 21–100):* The YAMNet backbone is unfrozen, and the learning rate is substantially decreased (e.g., maintained at 1×10^{-5}). Importantly, YAMNet's Batch Normalization layers are manually frozen using a TensorFlow control function to avoid Catastrophic Forgetting caused by updating BN statistics on the small dataset.

D. ANC Features

1) **Measuring Secondary Path**
The secondary path means how the sound changes when it travels from the speaker (that plays the anti-noise) to the error microphone (placed near the listener's ear). It includes how much the sound is delayed, weakened, or altered in the air. This measured result is stored as a list of numbers (a vector). This vector is later used as a reference for calculations.
2) **Generating Anti-noise Signal**
When noise is detected, the system starts with an empty filter (no adjustment yet). The system applies a filter to the noise signal and then passes it through the secondary path model to predict how the anti-noise will actually sound in the air. The predicted anti-noise is played through the speaker to cancel the noise.
3) **Checking result and update the filter**
The error microphone, placed near the listener's ear, listens to the actual sound and checks if the noise level has decreased. If there is still noise, the system adjusts the filter based on the difference between the target and the actual sound. This process repeats continuously and very quickly, allowing the system to gradually improve the noise cancellation performance.

Additional explanation:

- Secondary path vector: A set of numbers that describes how sound changes from the speaker to the microphone.
- Filter: A simple set of rules (multiplying and adding numbers) that transforms the incoming noise into an opposite sound wave
- Core idea: First, measure how sound travels (secondary path), then create an opposite sound, and finally keep adjusting it based on the microphone feedback.

V. Architecture Design & Implementation

A. Overall architecture

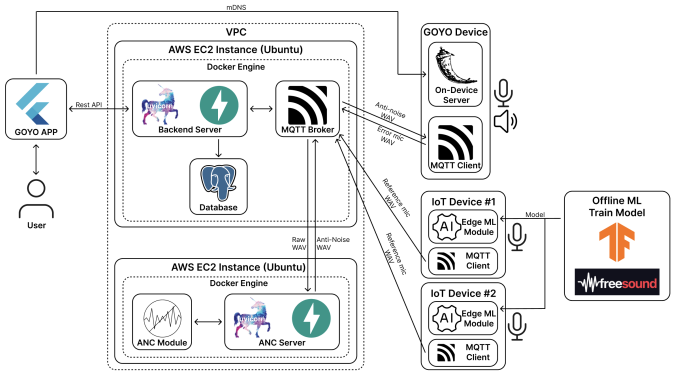


Fig. 9: System architecture

GOYO users interact with the system through a mobile application to monitor and control an AI-based Active Noise Control system for smart home environments. The system addresses continuous low-frequency noise from household appliances like air conditioners and refrigerators that reduce user focus during studying or remote work. To achieve this, the system consists of 6 main modules: Frontend, Backend, ANC Server, AI Module, GOYO Device, and IoT Device.

The first module is "Frontend," developed with Flutter and Dart. This module serves as the primary user interface for GOYO, providing a cross-platform mobile application for smart home noise control. The Frontend enables users to automatically discover and pair GOYO devices on the local network, visualize real-time ANC status and device states, manage user authentication and profile settings, and control per-appliance noise cancellation targeting. Through intuitive design, the module ensures seamless user experience across iOS and Android platforms while maintaining secure communication with backend services.

The second module is "Backend," built with FastAPI and Python on an AWS EC2 t3.small instance, serving as the central orchestrator of GOYO's functionality. This module manages all data flow between the mobile application, GOYO

devices, and AI services while ensuring data consistency through a PostgreSQL-based storage system. It provides secure user authentication, handles device registration and pairing, and manages user profiles and appliance configurations. The Backend also coordinates communication between devices and the ANC Server by operating as the system's MQTT broker and API gateway. It delivers necessary configuration data to devices, processes incoming device messages, and maintains centralized control over user-specific targeting options. With its RESTful architecture and structured service design, the Backend ensures stable, scalable, and secure operation of the entire GOYO system.

The third module is "ANC Server," built with FastAPI and Python on an AWS EC2 t3.small instance, serving as the core signal-processing engine for active noise cancellation. This module receives reference audio streams from IoT devices or the simulated GOYO Device and processes them in real time to generate anti-noise signals. In doing so, it accounts not only for phase-inversion but also for feedback microphone input, secondary-path characteristics, and other acoustic factors necessary for effective ANC. The server manages ANC activation based on targeted appliances and user preferences, synchronizes audio streams to maintain stable timing, and communicates with devices through MQTT. Through its low-latency processing pipeline, the ANC Server enables reliable and adaptive noise cancellation across different home environments.

The fourth module is "AI Module," the offline training and model development environment for GOYO's edge AI capabilities. This module is responsible for building and training the appliance noise classification models that run directly on IoT Devices or on the simulated GOYO Device during development. Using large collections of household audio data, it trains deep learning models capable of identifying sounds from appliances such as refrigerators, air conditioners, washing machines, and vacuum cleaners. The AI Module handles data preparation, model training, and model optimization for edge deployment. Once trained, the models are converted into lightweight formats suitable for Raspberry Pi–based devices and packaged for deployment. These models enable real-time classification on the device itself, allowing GOYO to detect which appliance is generating noise. The classification results—such as appliance type and confidence—are then sent via MQTT, enabling intelligent and targeted noise cancellation throughout the system.

The fifth module is "GOYO Device," the physical ANC hardware placed near the user—typically mounted on a smart chair implemented using a Raspberry Pi. This device captures real-time audio through microphones, plays anti-noise signals through speakers, and serves as the endpoint where active noise cancellation is physically applied. The GOYO Device receives anti-noise audio from the ANC Server, outputs it through speakers, and simultaneously measures the remaining noise at the user's position using an error microphone. It transmits these error signals back to the server, enabling continuous adaptive optimization of ANC performance. It supports automatic startup and stable operation through system-level service management, ensuring consistent and reliable ANC performance in everyday use.

The sixth module is "IoT Device," representing the smart home appliances in the GOYO ecosystem—such as air conditioners, refrigerators, or washing machines—that provide noise information to the system. In the envisioned architecture, each appliance is equipped with a reference microphone and an on-device AI model capable of identifying its own noise in real time. These devices send classified noise data and reference audio streams to the ANC Server, enabling accurate and appliance-specific noise cancellation. This distributed edge-AI setup allows appliances to process audio locally and transmit only essential information, reducing network usage and ensuring low-latency operation. In the current development environment, this IoT Device functionality is simulated using the Raspberry Pi–based GOYO Device, which emulates both sound capture and on-device noise classification to represent how a fully deployed smart home network would operate.

To ensure system reliability and performance, we implemented standardized API endpoints with robust error handling, monitoring, and logging capabilities across all modules - the Frontend mobile application in Flutter, Backend API service in FastAPI, ANC Server signal processing in FastAPI, AI Module offline training pipeline, GOYO Device edge processing on Raspberry Pi, and IoT Device distributed sensing. This comprehensive approach ensures seamless communication and optimal performance throughout the entire system.

B. Directory Organization

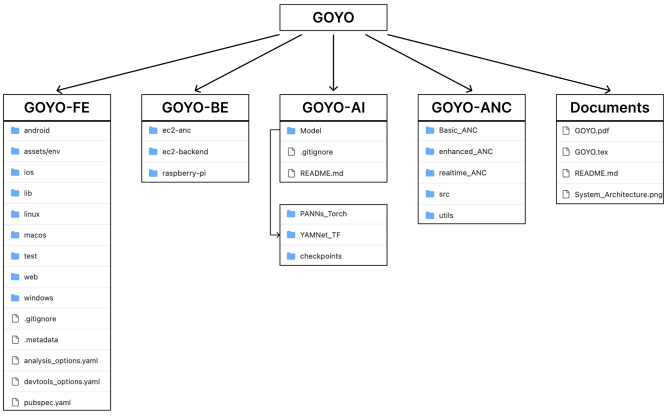


Fig. 10: Directory Organization

TABLE I: Directory Organization - Frontend

Directory	File names	Used Modules
GOYO-FE /lib	main.dart	flutter flutter-dotenv provider
GOYO-FE /lib/theme	app_theme.dart	flutter

Directory	File names	Used Modules
GOYO-FE /lib/core/auth	token_manager.dart	flutter_secure_storage
GOYO-FE /lib/core/config	env.dart	flutter_dotenv
GOYO-FE /lib/features/auth	auth_provider.dart	flutter/foundation
GOYO-FE /lib/features/anc	anc_store.dart	flutter/material
GOYO-FE /lib/data/services	api_service.dart	dio
GOYO-FE /lib/data/services	goyo_device_service .dart	http multicast_dns
GOYO-FE /lib/data/models	device_models.dart	
GOYO-FE /lib/data/models	noise_appliance.dart	
GOYO-FE /lib/ui/home	home_page.dart	flutter/material
GOYO-FE /lib/ui/home /widget	bottomnavigationbar .dart	flutter/material
GOYO-FE /lib/ui/home /widget	hometab.dart	flutter/material provider
GOYO-FE /lib/ui/device	devicemanager_page dart	flutter/material provider
GOYO-FE /lib/ui/device /widget	deviceinfo.dart	flutter/material provider
GOYO-FE /lib/ui/splash	splash_page.dart	flutter/material
GOYO-FE /lib/ui/login	login_page.dart	flutter/material provider
GOYO-FE /lib/ui/login	signup_page.dart	flutter/material provider
GOYO-FE /lib/ui/login	recovery_page.dart	flutter/material
GOYO-FE /lib/ui/profile	profile_page.dart	flutter/material provider

TABLE II: Directory Organization - Backend 1

Directory	File names	Used Modules
GOYO-BE /ec2-anc	Dockerfile docker-compose.yml setup.sh	docker docker-compose
GOYO-BE /ec2-anc /anc	Dockerfile anc_controller.py audio_processor.py config.py main.py mqtt_publisher.py mqtt_subscriber.py requirements.txt	fastapi pydantic-settings paho-mqtt numpy

TABLE III: Directory Organization - Backend 2

Directory	File names	Used Modules
GOYO-BE /ec2-backend	docker-compose.yml setup.sh	docker docker-compose
GOYO-BE /ec2-backend /backend	Dockerfile requirements.txt	
GOYO-BE /ec2-backend /backend /app	__init__.py config.py database.py main.py	fastapi pydantic-settings sqlalchemy
GOYO-BE /ec2-backend /backend /app /api	__init__.py appliances.py auth.py devices.py home.py profile.py	fastapi pydantic-settings sqlalchemy
GOYO-BE /ec2-backend /backend /app /models	__init__.py appliance.py device.py user.py	sqlalchemy
GOYO-BE /ec2-backend /backend /app /services	__init__.py appliance_service.py auth_service.py device_service.py matt_service.py profile_service.py	fastapi sqlalchemy paho-mqtt
GOYO-BE /ec2-backend /backend /app /utiles	__init__.py audio_device.py dependencies.py email.py security.py	fastapi python-jose passlib sqlalchemy numpy pyaudio
GOYO-BE /ec2-backend /mosquitto /config	mosquitto.conf	

TABLE IV: Directory Organization - Backend 3

Directory	File names	Used Modules
GOYO-BE /raspberry-pi	audio_client.py device_server.py goyo_config.json goyo-audio.service goyo-device-server.service mqtt_test.py README.md requirements.txt rpi_inference.py setup.sh test_alsaaudiomqtt.py test_audiostream.py test_latency.py testmqtt.py	flask scipy zeroconf alsaaudio numpy paho-mqtt
GOYO-BE /raspberry-pi /models	classifier.tflite	

TABLE V: Directory Organization - AI

Directory	File names	Used Modules
GOYO-AI /Model /PANNs_Torch	augmentutils.py PANNsdataloader.py PANNsmodel.py PANNstrain.py	scikit-learn torch tqdm librosa numpy
GOYO-AI /Model /YAMNET_TF	augmentutils.py YAMNetdataloader.py YAMNetinference.py YAMNetlayers.py YAMNettrain.py	scikit-learn tensorflow tensorflow-hub librosa numpy
GOYO-AI /Model /checkpoints	best_model_yamnet .keras	

TABLE VI: Directory Organization - ANC

Directory	File names	Used Modules
GOYO-ANC /Basic_ANC	ANC_test.py fxlms_controller.py reference_then_anc.py session_utils.py	numpy sounddevice
GOYO-ANC /enhanced_ANC	__init__.py config.py fxlms_controller_numba.py io_utils.py lowfreq_cli.py session_utils.py	numpy sounddevice numba
GOYO-ANC /src	sine_200Hz.wav sine_200Hz100s.wav sine_200Hz200s.wav	
GOYO-ANC /utils	inspect_secondary_path.py list_audio_devices.py make_sine_tone.py	numpy pyaudio soundfile

C. Module 1: Frontend

Purpose

To develop GOYO, Flutter was chosen as it supports both iOS and Android platforms as a cross-platform framework. It was selected for its ability to develop applications for both platforms with a single codebase using the Dart programming language. The main purpose of the frontend is to provide a user-friendly interface for smart home noise control management. It supports device discovery through mDNS, secure device pairing, and real-time ANC status monitoring, offering an intuitive experience for controlling GOYO devices and managing per-appliance noise cancellation settings.

Functionality

The main functionalities of the frontend include user authentication and account management (sign-up, login, password recovery), automatic GOYO device discovery via mDNS, secure device pairing with MQTT configuration delivery, real-time ANC status visualization and device state monitoring, appliance registration and per-appliance noise cancellation control, personal profile management, and support for cross-platform deployment on iOS and Android.

Location of source code

https://github.com/SWE-ITE/GOYO/tree/main/GOYO-FE

Class component

- **main.dart**: This file serves as the main entry point for the Flutter-based mobile application. It initializes the application with Provider for state management, loads environment variables from .env files, and sets up the MaterialApp with custom theme configuration and routing.
- **core folder**: Contains core configuration and authentication utilities.
  - **config/env.dart**: Manages environment variables loaded from .env files for different build configurations (development, production). Provides centralized access to API endpoints and configuration settings.
  - **auth/token_manager.dart**: Handles secure storage and management of authentication tokens using flutter_secure_storage. Manages JWT token persistence and retrieval for authenticated API requests.
- **features folder**: Contains feature-specific state management and business logic.
  - **auth/auth_provider.dart**: Manages authentication state using Provider pattern. Handles user sign-up, login, logout, and token refresh operations. Maintains user session state across the application.
  - **anc/anc_store.dart**: Manages ANC state and control logic. Handles ANC activation/deactivation, per-appliance targeting, and real-time status updates from GOYO devices.
- **data folder**: Contains data models and service layer implementations.
  - **models/device_models.dart**: Defines data models for GOYO devices, appliances, and device configurations. Implements JSON serialization/deserialization for API communication.
  - **services/api_service.dart**: Provides abstraction layer for backend API communication using dio HTTP client. Handles REST API requests with authentication headers, error handling, and response parsing.
  - **services/goyo_device_service.dart**: Implements mDNS device discovery using multicast_dns package. Handles device pairing flow, MQTT configuration delivery, and device registration with backend server.
- **ui folder**: Contains screen implementations and UI components.
  - **splash/splash_page.dart**: Displays the initial splash screen while loading application resources and checking authentication status.
  - **login/login_page.dart**: The login screen for GOYO users. Implements email/password authentication with input validation.
  - **login/signup_page.dart**: The sign-up screen for new GOYO users. Collects user information and creates new accounts through backend API.
  - **login/recovery_page.dart**: Provides password recovery functionality for users who forgot their credentials.
  - **home/home_page.dart**: The primary screen of the GOYO application. Displays connected GOYO devices, current ANC status, and quick controls for noise cancellation.
  - **home/widget/bottomnavigationbar.dart**: Implements bottom navigation bar for switching between main sections (Home, Device Manager, Profile).
  - **home/widget/hometab.dart**: Displays the home tab content with device status overview and ANC controls.
  - **device/devicemanager_page.dart**: Shows the list of discovered and paired GOYO devices. Provides device pairing options through mDNS discovery.
  - **device/widget/deviceinfo.dart**: Displays detailed information for individual GOYO devices, including

connection status, MQTT configuration, and device capabilities.
  - **profile/profile_page.dart**: The profile page section for GOYO users. Displays user information, registered appliances, and application settings.
- **theme folder**: Contains application theme configuration.
  - **app_theme.dart**: Defines the visual theme for the GOYO application, including color schemes, typography, and component styles. Ensures consistent design across all screens.
- **pubspec.yaml**: Flutter project configuration file that manages dependencies and project metadata. Specifies required packages including Provider for state management, dio for HTTP requests, flutter_secure_storage for secure token storage, and multicast_dns for device discovery.

D. Module 2: Backend

Purpose

The primary purpose of the backend is to enable smart home noise control through distributed IoT device coordination and real-time ANC management. It stores user data, GOYO device information, appliance configurations, and ANC status in a PostgreSQL database using SQLAlchemy ORM, while providing secure API endpoints for frontend interactions. Built using FastAPI and Python, it manages MQTT broker for IoT messaging, coordinates mDNS service advertisement for device discovery, and facilitates communication between GOYO devices, ANC server, and mobile clients.

Functionality

1) Authentication Management

Implements user registration and login with bcrypt password encryption and JWT token-based authentication. Provides secure session management with token validation and refresh mechanisms.

2) Device Management

Manages GOYO device pairing through mDNS discovery and MQTT credential distribution. Provides APIs for device registration, status updates, and real-time synchronization across the system. Handles device-to-backend communication via MQTT broker.

3) Appliance Management

Registers and manages user appliances for per-appliance ANC targeting. Provides APIs for appliance CRUD operations, configuration updates, and appliance-device associations. Enables selective noise cancellation based on appliance types.

4) MQTT Coordination

Operates as the central MQTT broker for device-to-server communication. Manages MQTT user authentication, topic-based message routing, and real-time data streaming between GOYO devices, IoT devices, and ANC server. Implements pub/sub messaging for audio streams and control signals.

5) Profile Management

Handles user profile information, preferences, and system settings. Provides APIs for profile updates, email verification, and account management.

Location of source code

https://github.com/SWE-ITE/GOYO/tree/main/GOYO-BE/ec2-backend

Class component

- **main.py**: FastAPI application entry point. Configures middleware including CORS, initializes database connections, registers API routers, and starts MQTT service. Implements application lifecycle management with startup and shutdown events.
- **config.py**: Manages application configuration using pydantic-settings. Loads environment variables for database connection, MQTT broker settings, JWT secrets, and API endpoints. Provides centralized configuration access across the application.
- **database.py**: Defines SQLAlchemy database engine and session management. Implements declarative base for ORM models and provides database connection dependency injection for FastAPI endpoints.
- **api folder**: REST API endpoint implementations
  - **auth.py**: Handles user authentication endpoints. Implements sign-up, login, token refresh, and logout operations. Uses JWT tokens for stateless authentication and bcrypt for password hashing.
  - **devices.py**: Manages GOYO device endpoints. Provides device pairing API that generates MQTT credentials, registers devices with mDNS information, and returns configuration data for device setup. Implements device listing, status updates, and deletion.
  - **appliances.py**: Controls appliance management endpoints. Handles appliance registration, updates, deletion, and listing for per-user appliance configurations. Implements appliance-device association logic.
  - **home.py**: Provides home page data aggregation endpoints. Returns dashboard information including connected devices, active appliances, and ANC status overview.
  - **profile.py**: Manages user profile endpoints. Handles profile information retrieval, updates, email verification, and account settings.
- **models folder**: Database schema definitions using SQLAlchemy ORM
  - **user.py**: Defines user schema with email validation and password encryption. Includes fields for authentication metadata, timestamps, and relationship definitions with devices and appliances.
  - **device.py**: Specifies GOYO device schema with MQTT configuration fields. Includes device identification, connection status, MQTT credentials, and mDNS service information.
  - **appliance.py**: Structures appliance schema for noise source registration. Contains fields for appliance type, targeting configuration, and user associations.
- **schemas folder**: Pydantic models for request/response validation

  - **user.py**: Defines user data transfer objects including UserCreate, UserLogin, UserResponse, and Token schemas. Implements validation rules for email format and password requirements.
  - **device.py**: Specifies device-related schemas including DeviceRegister, DeviceResponse, and DeviceConfig for API communication and validation.
  - **appliance.py**: Defines appliance data models including ApplianceCreate, ApplianceUpdate, and ApplianceResponse for request validation.
  - **profile.py**: Structures profile-related schemas for user information updates and profile data responses.
- **services folder**: Business logic implementations
  - **auth_service.py**: Implements authentication business logic. Handles user registration with password hashing, login validation, JWT token generation and verification, and session management.
  - **device_service.py**: Manages device pairing and registration logic. Generates unique MQTT credentials for devices, stores device configuration in database, and coordinates with MQTT broker for user provisioning.
  - **appliance_service.py**: Implements appliance management operations. Handles appliance CRUD logic, validates appliance-user associations, and manages per-appliance targeting configurations.
  - **mqtt_service.py**: Manages MQTT broker integration using paho-mqtt. Handles MQTT client initialization, connection management, topic subscription/publishing, and message routing for device communication.
  - **profile_service.py**: Implements user profile management logic. Handles profile data updates, email verification workflows, and account setting modifications.
- **utils folder**: Utility functions and helper modules
  - **security.py**: Provides security utilities including password hashing with bcrypt, JWT token creation and validation using python-jose, and token payload extraction for authentication.
  - **dependencies.py**: Defines FastAPI dependency injection functions. Implements database session management, current user extraction from JWT tokens, and authentication middleware.
  - **email.py**: Handles email-related utilities for verification and notifications. Implements email format validation and email sending functionality.
  - **audio_device.py**: Provides audio device utility functions for MQTT topic generation and device identification.

E. Module 3: ANC Server

Purpose

The primary purpose of the ANC Server is to serve as the core signal processing engine for active noise cancellation in the GOYO system. Built using FastAPI and Python on AWS EC2, it receives reference audio streams from IoT devices or simulated GOYO devices via MQTT broker, processes these signals through FxLMS adaptive filtering algorithms, and generates phase-inverted anti-noise waveforms in real-time. The server implements sophisticated audio processing pipelines utilizing numpy for numerical computations and numba for just-in-time compilation to achieve low-latency performance, while maintaining sub-100ms latency requirements for effective ANC performance. It acts as the central processing hub that transforms noise signatures into cancellation signals through MQTT-based communication.

Functionality

1) Audio Stream Reception

Receives reference audio streams from IoT devices or GOYO devices via MQTT broker. Processes incoming audio data with appliance classification metadata to identify noise sources. Implements buffering and synchronization mechanisms to handle real-time audio streams.

2) Signal Processing

Implements FxLMS adaptive filtering algorithm for real-time noise cancellation. Applies secondary path modeling to compensate for acoustic delay between anti-noise speaker and error microphone. Processes error microphone feedback for continuous filter coefficient adaptation using block-based audio processing. Utilizes numba JIT compilation for optimized performance with configurable step sizes and filter lengths. Provides performance metrics including mean squared error and convergence monitoring for algorithm optimization.

3) Anti-Noise Generation

Generates phase-inverted anti-noise waveforms using FxLMS-processed filter coefficients that destructively interfere with reference noise. Implements secondary path compensation to account for acoustic path between speaker and error microphone. Manages audio buffer synchronization with configurable block sizes to maintain timing alignment between reference and anti-noise signals. Applies vectorized numpy operations for efficient waveform generation and ensures continuous anti-noise output without gaps or artifacts through synchronized block processing.

4) MQTT Publishing

Transmits generated anti-noise audio to GOYO device speakers via MQTT broker. Broadcasts real-time ANC status updates including activation state, processing latency, and performance metrics. Implements topic-based message routing for efficient device-specific anti-noise delivery.

Location of source code

MQTT Integration: https://github.com/SWE-ITE/GOYO/tree/main/GOYO-BE/ec2-anc

ANC Algorithms: https://github.com/SWE-ITE/GOYO/tree/main/GOYO-ANC

Class component

GOYO-BE/ec2-anc - MQTT Integration Layer

- **main.py**: FastAPI application entry point for ANC server. Configures CORS middleware, initializes WebSocket endpoints for status monitoring, and manages application lifecycle. Starts MQTT subscriber and publisher services

for audio stream handling. Implements health check endpoints and error handling for production deployment.

- **config.py**: Manages ANC server configuration using pydantic-settings. Loads environment variables for MQTT broker connection, audio processing parameters (sample rate, channels, chunk size, buffer size), and server settings. Provides centralized access to configuration across ANC processing modules.
- **audio_processor.py**: Implements FxLMS adaptive filtering algorithm integration for ANC processing. Applies secondary path modeling and processes error microphone feedback for filter adaptation. Utilizes GOYO-ANC algorithms for anti-noise generation. Manages audio buffer synchronization and timing alignment between reference and anti-noise streams.
- **mqtt_subscriber.py**: Manages MQTT subscription for incoming audio streams using paho-mqtt. Connects to MQTT broker with authentication credentials and subscribes to reference audio topics. Receives and decodes audio data with appliance classification metadata. Implements message queuing and forwards audio streams to audio processor. Handles connection failures with automatic reconnection logic.
- **mqtt_publisher.py**: Manages MQTT publishing for outgoing anti-noise streams using paho-mqtt. Encodes processed anti-noise audio and publishes to device-specific topics. Broadcasts ANC status updates including activation state, processing latency, and buffer status. Implements message prioritization to ensure anti-noise delivery takes precedence. Handles connection management and publishing failures with retry mechanisms.

GOYO-ANC - Core ANC Algorithms

- **Basic_ANC folder**: Basic ANC implementation with FxLMS algorithm
  - **fxlms_controller.py**: Implements core FxLMS adaptive filtering algorithm. Defines FxLMSANC class with filter coefficient adaptation using error microphone feedback. Applies secondary path modeling to compensate for acoustic delay. Implements block-based audio processing with configurable step size and filter length.
  - **session_utils.py**: Provides session management utilities for ANC operations. Implements controller initialization with secondary path impulse response. Manages reference signal playback synchronized with anti-noise generation.
  - **reference_then_anc.py**: Implements reference-then-ANC operation mode for noise reduction demonstration.
  - **ANC_test.py**: Test script for ANC system validation with performance metrics logging.
- **enhanced_ANC folder**: Enhanced ANC implementation with numba optimization
  - **fxlms_controller_numba.py**: Numba-optimized FxLMS implementation with JIT-compiled signal processing for reduced computational latency.
  - **config.py**: Configuration management for enhanced ANC parameters including sample rate, block size, and filter settings.
  - **io_utils.py**: Audio I/O utilities for file reading, writing, and stream formatting using soundfile.
  - **lowfreq_cli.py**: Command-line interface for low-frequency noise cancellation with interactive ANC control.
- **utils folder**: System configuration and testing utilities
  - **inspect_secondary_path.py**: Implements secondary path acoustic modeling by measuring impulse response between speaker and microphone.
  - **list_audio_devices.py**: Enumerates available audio devices using pyaudio for system setup.
  - **make_sine_tone.py**: Generates sine wave test signals for ANC calibration and validation.

F. Module 4: AI Module

Purpose

The primary purpose of the AI Module is to serve as the offline training and model development environment for GOYO's edge AI capabilities. This module is responsible for training YAMNet-based and PANNs-based appliance noise classification models that run on IoT devices or simulated GOYO devices during development. Built using Python with TensorFlow and PyTorch frameworks, it processes large-scale audio datasets to train deep learning models that distinguish between different household appliance noises including refrigerator, air conditioner, washing machine, and vacuum cleaner sounds. The trained models are optimized for edge deployment using TensorFlow Lite quantization and exported in formats compatible with Raspberry Pi inference, enabling real-time on-device classification without cloud dependency.

Functionality

1) Dataset Management

Manages large-scale audio datasets stored on Google Drive due to size constraints. Implements data preprocessing including audio loading, resampling, and normalization. Applies data augmentation techniques including noise injection, pitch shifting, time masking, and frequency masking to improve model robustness. Implements train-test split using scikit-learn for proper evaluation.

2) Model Training

Trains YAMNet-based models using TensorFlow for appliance-specific noise classification. Implements PANNs-based models using PyTorch as alternative architecture for comparison. Fine-tunes pre-trained models on custom appliance noise datasets for transfer learning. Implements training loops with callbacks for model checkpointing, early stopping, and learning rate scheduling. Utilizes class weighting to handle imbalanced datasets across different appliance types.

3) Model Optimization

Converts trained models to TensorFlow Lite format for edge deployment on Raspberry Pi. Applies quantization techniques

to reduce model size and improve inference speed on resource-constrained devices. Implements model validation and performance testing on test datasets. Evaluates classification accuracy, precision, recall, and F1-scores for each appliance class.

4) Voice Activity Detection

Integrates VAD model training and tuning to prevent ANC interference during conversations. Implements speech detection algorithms to identify human voice presence in audio streams. Trains models to distinguish between appliance noise and speech for intelligent ANC control.

5) Model Deployment

Packages trained models for deployment to GOYO devices running on Raspberry Pi. Implements model export utilities that generate compatible formats for edge inference. Provides inference scripts for real-time classification on captured audio streams. Enables seamless model updates and versioning for deployed devices.

Location of source code

https://github.com/SWE-ITE/GOYO/tree/main/GOYO-AI

Class component

- **Model/PANNs_Torch folder**: PyTorch implementation of PANNs models
  - **PANNs_train.py**: Main training script for PANNs models using PyTorch. Implements training loop with model checkpointing and evaluation. Utilizes scikit-learn for train-test split and class weight calculation. Loads audio data using PANNs_dataloader and applies augmentation from augment_utils. Configures optimizer settings and learning rate scheduling for model convergence.
  - **PANNs_model.py**: Defines PANNs neural network architecture using PyTorch. Implements custom model layers and forward propagation logic for audio classification. Adapts pre-trained PANNs models for appliance noise classification task. Provides model instantiation and weight initialization functions.
  - **PANNs_dataloader.py**: Implements PyTorch Dataset and DataLoader for audio data loading. Handles audio file reading using librosa and preprocessing. Applies augmentation techniques during training for improved generalization. Implements batch generation and data shuffling for training efficiency.
  - **augment_utils.py**: Provides audio augmentation utilities for data preprocessing. Implements noise injection to simulate real-world acoustic conditions. Applies pitch shifting to increase pitch variance in training data. Implements time masking and frequency masking for robust feature learning.
- **Model/YAMNet_TF folder**: TensorFlow implementation of YAMNet models
  - **YAMNet_train.py**: Main training script for YAMNet models using TensorFlow. Implements transfer learning by fine-tuning pre-trained YAMNet from TensorFlow Hub. Utilizes keras callbacks including ModelCheckpoint for saving best models. Loads audio data using YAMNet_dataloader and applies augmentation. Configures training hyperparameters and evaluation metrics.
  - **YAMNet_layers.py**: Defines custom YAMNet layers using TensorFlow Keras. Implements YAMNetLayer that wraps pre-trained YAMNet model with custom classification head. Adapts YAMNet embeddings for appliance-specific classification task. Provides layer configuration for model architecture customization.
  - **YAMNet_dataloader.py**: Implements TensorFlow Keras Sequence for efficient data loading. Handles audio preprocessing and feature extraction compatible with YAMNet input requirements. Applies augmentation techniques during training batches. Implements generator-based data loading for memory efficiency with large datasets.
  - **YAMNet_inference.py**: Provides inference utilities for trained YAMNet models. Implements model loading from saved checkpoints in keras format. Applies preprocessing to input audio for classification. Returns classification results with confidence scores and predicted appliance types.
  - **augment_utils.py**: Provides audio augmentation utilities for data preprocessing. Implements noise injection, pitch shifting, time masking, and frequency masking techniques identical to PANNs augmentation for consistency.
- **Model/checkpoints folder**: Stores trained model checkpoints
  - **best_model_yamnet.keras**: Best performing YAMNet model saved in Keras format. Contains trained weights and architecture for appliance noise classification. Ready for deployment to Raspberry Pi after TensorFlow Lite conversion. Represents the production model for edge inference on GOYO devices.

G. Module 5: GOYO Device

Purpose

The primary purpose of the GOYO Device is to serve as the physical ANC hardware deployed near the user, typically mounted on a smart chair or desk. Implemented on Raspberry Pi, this device contains an error microphone that measures residual noise at the user's ear position after ANC is applied, speakers that output phase-inverted anti-noise signals for active noise cancellation, and an MQTT client that receives anti-noise audio from the ANC Server and transmits error signals for adaptive optimization. The device uses ALSA (Advanced Linux Sound Architecture) for low-latency audio capture and playback, with hardware configuration including multiple USB audio interfaces for reference microphone, error microphone, and speaker output. The device maintains persistent systemd services for automatic startup and recovery, ensuring reliable operation in production smart home environments. Additionally, it provides an mDNS discovery server that enables mobile

applications to automatically detect and pair GOYO devices on the local network.

Functionality

1) Error Microphone Monitoring

Captures residual noise at user's ear position using error microphone after ANC is applied. Provides real-time feedback to ANC Server for adaptive algorithm optimization. Implements low-latency audio capture using ALSA for effective ANC performance. Enables continuous monitoring of noise cancellation effectiveness.

2) Anti-Noise Playback

Receives anti-noise audio streams from ANC Server via MQTT broker. Plays phase-inverted signals through near-ear speakers for localized noise cancellation. Implements synchronized playback with minimal latency to maintain destructive interference. Manages audio buffer to ensure continuous anti-noise output without gaps.

3) Device Discovery

Implements mDNS service advertisement for automatic device discovery by mobile applications. Broadcasts device information including device name, capabilities, and connection endpoints on local network. Provides HTTP server for device pairing API that accepts MQTT configuration from paired users. Enables seamless device onboarding without manual network configuration.

4) MQTT Communication

Maintains persistent connection to MQTT broker with authenticated credentials. Subscribes to device-specific topics for receiving anti-noise streams and control commands. Publishes error microphone feedback and device status updates to ANC Server. Implements automatic reconnection logic for network resilience.

5) Edge AI Inference

In current development environment, simulates IoT Device functionality by running YAMNet-based appliance classification on reference audio. Performs real-time inference on captured audio streams to identify appliance noise types. Implements VAD for speech detection to prevent ANC interference during conversations. Transmits classified audio and metadata to ANC Server via MQTT.

Location of source code

https://github.com/SWE-ITE/GOYO/tree/main/GOYO-BE/raspberry-pi

Class component

- **audio_client.py**: Main audio processing client for GOYO device. Implements audio capture using ALSA (alsaaudio) for reference and error microphones. Integrates rpi_inference.py for edge AI classification of appliance noises. Implements VAD using signal processing to detect speech presence. Manages MQTT client connection using paho-mqtt for bidirectional communication with ANC Server. Publishes reference audio streams with classification metadata to ANC Server. Subscribes to anti-noise topics and plays received audio through speakers. Implements audio buffering and synchronization for continuous operation. Runs as systemd service (goyo-audio.service) for automatic startup and recovery.
- **device_server.py**: mDNS discovery server implemented using Flask and Zeroconf. Broadcasts GOYO device presence on local network using multicast DNS service advertisement. Provides HTTP API endpoints for device pairing and MQTT configuration delivery. Accepts MQTT credentials from paired users and stores in goyo_config.json. Returns device information including capabilities, hardware configuration, and connection status. Enables mobile applications to discover and pair devices without manual network configuration. Runs as systemd service (goyo-device-server.service) for persistent availability.
- **rpi_inference.py**: Edge AI inference engine for appliance noise classification on Raspberry Pi. Loads trained YAMNet model from Model/checkpoints directory for on-device inference. Implements audio preprocessing including resampling and feature extraction for model input. Performs real-time classification on audio chunks to identify appliance types. Returns classification results with confidence scores and predicted appliance labels. Optimized for Raspberry Pi performance with efficient numpy operations. In production, this functionality would run on individual IoT appliances rather than GOYO device.
- **goyo_config.json**: Configuration file storing MQTT broker connection settings. Contains MQTT broker host, port, username, and password for authenticated connection. Generated during device pairing process when user pairs device through mobile application. Provides persistent storage of device configuration across reboots.
- **setup.sh**: Device setup script for initial GOYO device configuration. Installs required system dependencies and Python packages from requirements.txt. Configures ALSA audio interfaces and sets up USB audio device permissions. Creates systemd service files and enables automatic startup. Implements configuration validation and network connectivity tests.
- **goyo-audio.service**: Systemd service unit for audio_client.py automatic startup. Configures service to start on boot and restart on failure for reliability. Sets environment variables and working directory for audio client execution. Implements dependency ordering to ensure MQTT broker connectivity before starting.
- **goyo-device-server.service**: Systemd service unit for device_server.py automatic startup. Ensures mDNS discovery server is always available for device pairing. Configures restart policy for network resilience. Manages service lifecycle for production deployment.
- **requirements.txt**: Python package dependencies for Raspberry Pi deployment. Includes alsaaudio for ALSA audio interface, paho-mqtt for MQTT communication, flask for HTTP server, zeroconf for mDNS service advertisement, numpy and scipy for signal processing, and librosa for audio preprocessing.

H. Module 6: IoT Device

Purpose

The IoT Device module represents the assumed smart home environment where each major household appliance such as air conditioner, refrigerator, or washing machine is equipped with edge AI capabilities for distributed noise sensing and classification. In this envisioned production architecture, each appliance contains a reference microphone that captures the noise signature at the source, an AI classification model running YAMNet-based sound classification to identify appliance noise types in real-time, a VAD system that detects human speech to prevent ANC interference during conversations, and an MQTT client that transmits classified noise data and audio streams to the ANC Server. This distributed edge AI approach reduces network bandwidth, enables real-time noise classification without cloud dependency, and provides localized noise sensing at each appliance source for optimal ANC performance. However, in the current development and testing environment, IoT Device functionality is simulated by the GOYO Device implemented on Raspberry Pi for convenience, where the single Raspberry Pi includes both reference microphone capabilities and AI inference to emulate the distributed IoT sensing network that would exist in a fully deployed smart home.

Functionality

1) Noise Capture

Captures appliance noise signatures at the source using reference microphones positioned near each appliance. Implements continuous audio streaming with configurable sample rates and channels. Provides high-quality audio capture to enable accurate noise classification and effective ANC processing.

2) Edge AI Classification

Performs on-device inference using YAMNet-based models to classify appliance noise types in real-time. Identifies specific appliance signatures including refrigerator hum, AC fan noise, washing machine vibration, and vacuum cleaner sounds. Generates classification confidence scores and appliance type metadata for each audio stream. Enables intelligent noise source identification without cloud dependency.

3) Voice Activity Detection

Implements VAD algorithms to detect human speech presence in audio streams. Distinguishes between appliance noise and conversational speech to prevent ANC interference during user conversations. Provides real-time speech detection flags to ANC system for intelligent activation control. Ensures ANC system does not interfere with voice communication in the smart home environment.

4) MQTT Communication

Transmits classified audio streams and metadata to ANC Server via MQTT broker. Publishes appliance identification, classification confidence, and VAD status along with reference audio data. Implements efficient topic-based messaging for device-specific audio routing. Maintains persistent connection to MQTT broker for reliable real-time communication.

5) Distributed Sensing

Operates as part of distributed IoT sensing network with multiple appliances providing simultaneous noise monitoring. Enables per-appliance noise source tracking for selective ANC targeting. Coordinates with backend server for appliance registration and targeting configuration. Provides scalable architecture for smart home environments with multiple noise sources.

Location of source code

https://github.com/SWE-ITE/GOYO/tree/main/GOYO-BE/raspberry-pi

Class component

In the production deployment, each IoT-enabled appliance would contain embedded hardware running the edge AI classification model with integrated microphone and MQTT client. The appliance would execute inference on captured audio locally and transmit results to the central ANC system.

In the current development environment, this functionality is implemented on Raspberry Pi as part of the GOYO Device module for testing and demonstration purposes. The relevant implementation files include:

- **rpi_inference.py** (in raspberry-pi folder): Implements YAMNet-based edge AI inference engine for appliance noise classification. Loads trained model from checkpoints and performs real-time classification on audio streams. This simulates the AI classification functionality that would run on each IoT appliance.
- **audio_client.py** (in raspberry-pi folder): Captures reference audio using ALSA and implements VAD for speech detection. Coordinates audio capture, AI inference, and MQTT publishing for classified audio streams. This simulates the complete IoT device functionality including noise capture, classification, and communication.
- **Model/checkpoints/best_model_yamnet.keras** (in goyo_ai folder): Trained YAMNet model deployed to Raspberry Pi for edge inference. Represents the AI model that would be embedded in each IoT appliance for on-device classification.

VI. Use Cases

A. Sign Up

Fig. 11: Sign-up page

Users can create an account by providing their basic information, including their name, email address, password, password confirmation, and phone number. After submitting the registration form, they are redirected to the login screen, where they can sign in and start using their newly created account.

B. Login

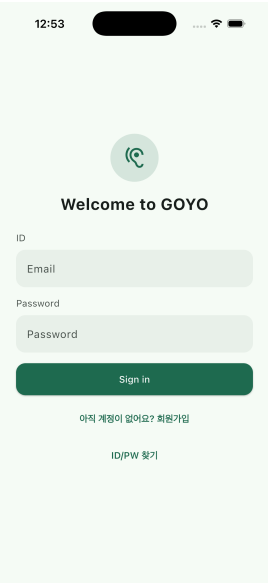


Fig. 12: Sign-up page

A Login Page allows users to access their accounts by verifying their credentials through ID and password authentication. Users simply enter their registered ID and password to sign in and begin using the service.

C. Loading

Fig. 13: Loading screen

A Loading Page is the first screen displayed when the app launches. It remains visible while essential components are initialized and automatically transitions to the next screen once the setup is complete.

D. Main Page

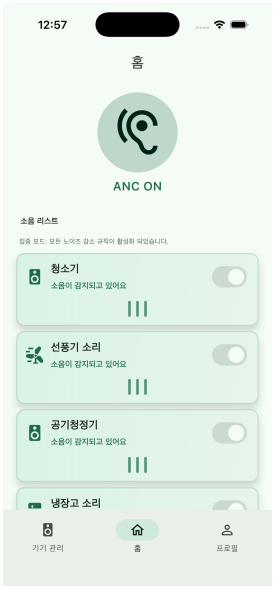


Fig. 14: Home page

On the Home Page, users can enable or disable noise cancellation. They can also view a real-time list of household appliances that are currently generating noise.

E. Device Management Page

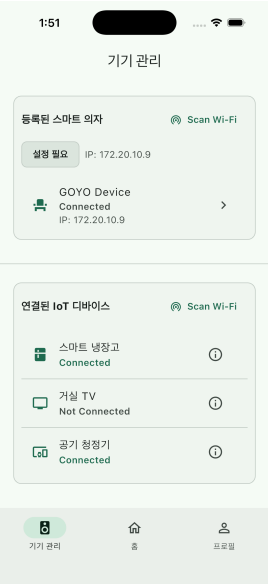


Fig. 15: Device management page

On the Device Management page, users can check the status of their connected GOYO device as well as other IoT appliances. They can also search for and add new devices using the Wi-Fi scan feature.

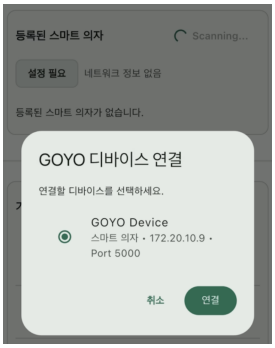


Fig. 16: Scan screen

F. Profile Page

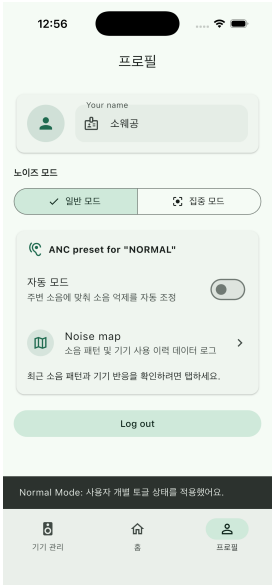


Fig. 17: Profile page

On the Profile Page, users can view their personal information and adjust noise-related settings. They can switch between Normal Mode and Focus Mode, check their noise logs, and log out of the app.

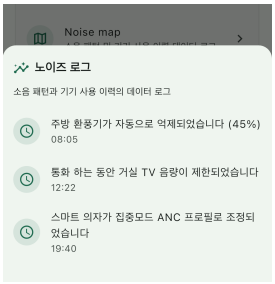


Fig. 18: Noise log screen

G. User Experience

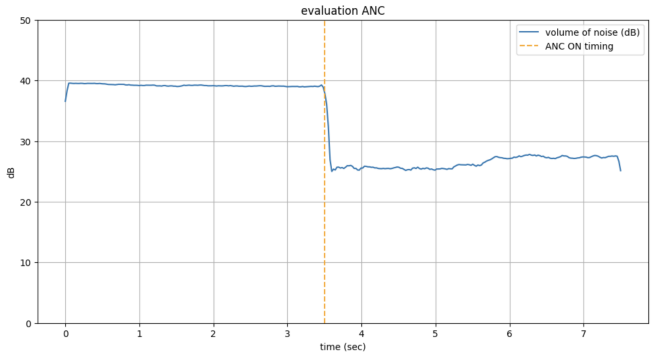


Fig. 19: dB transition graph

Users can experience a noise reduction of around 10 to 20 decibels. The graph above shows the measured decibel levels of the noise actually heard by the user.

VII. Discussion

A. Difficulties

Throughout the development and testing of the GOYO system, we encountered several technical challenges that impacted the overall performance and consistency of the active noise cancellation implementation.

Hardware limitations posed significant constraints on audio quality and system performance. The microphones and speakers used in our prototype setup exhibited modest fidelity, which affected both the accuracy of noise capture and the clarity of anti-noise signal reproduction. The limited frequency response and dynamic range of these components introduced distortions that complicated the adaptive filtering process, requiring additional signal conditioning and calibration efforts to achieve acceptable noise reduction.

The inherent sensitivity of active noise cancellation to environmental conditions presented substantial challenges in obtaining consistent experimental results. While we achieved successful noise cancellation in controlled test scenarios, the performance proved highly dependent on precise microphone and speaker positioning, room acoustics, and ambient environmental factors. Minor variations in device placement or changes in the acoustic environment led to noticeable fluctuations in cancellation effectiveness, making it difficult to establish reproducible baseline measurements. This variability highlighted the complexity of deploying ANC systems in real-world smart home environments where acoustic conditions are dynamic and unpredictable.

Network latency introduced through the MQTT-based communication architecture presented fundamental limitations for certain noise types. The cumulative delay across the entire signal path—from reference microphone capture, through MQTT transmission to the ANC server, signal processing, MQTT transmission back to the device, and finally error microphone feedback—resulted in processing latencies ranging from 120-150ms. This latency exceeded the critical threshold for effective cancellation of rapidly varying or transient noises, restricting the system's applicability primarily to continuous, quasi-stationary noise sources with relatively stable waveform characteristics, such as constant-frequency appliance hums. For noise signals with rapid temporal variations or unpredictable patterns, the delayed anti-noise output arrived too late to achieve destructive interference, significantly limiting the scope of addressable noise types in the current implementation.

B. Conclusion

This project presented GOYO, an AI-based active noise control system for smart home environments that integrates edge AI classification, distributed IoT sensing, and adaptive FxLMS filtering to achieve personalized noise management through appliance-specific targeting. Despite technical challenges, the system successfully demonstrated measurable noise reduction in controlled scenarios, validating the feasibility of combining MQTT-based IoT communication with adaptive filtering for residential applications.

Our initial design constrained the effective cancellation zone to the user's head vicinity, positioning error microphones and speakers on a smart chair for optimal feedback. However, successful test cases revealed noise reduction extending beyond this targeted region, with users experiencing perceptible cancellation even when positioned away from the designated listening position. This suggests potential for broader spatial coverage than initially anticipated, indicating that with optimized speaker placement and filter parameters, GOYO could extend effective noise cancellation to larger zones such as entire desk areas or small rooms.

Future work should address identified limitations through improved hardware, enhanced acoustic modeling, and latency reduction via edge processing. The promising extended spatial coverage results warrant investigation into scalable ANC configurations for whole-room noise management in smart home settings.

References

[1] B. Widrow *et al.*, "Adaptive noise cancelling: Principles and applications," *Proceedings of the IEEE*, vol. 63, no. 12, pp. 1692–1716, Dec. 1975, doi: 10.1109/PROC.1975.10036.

[2] S. J. Elliott and P. A. Nelson, "Active noise control," *IEEE Signal Processing Magazine*, vol. 10, no. 4, pp. 12–35, Oct. 1993, doi: 10.1109/79.248551.

[3] A. R. Thompson, J. M. Moran, and G. W. Swenson, Jr., *Digital Signal Processing*, in *Interferometry and Synthesis in Radio Astronomy*, 3rd ed., Taylor & Francis, 2017.

[4] M. Zhang, H. Lan, and W. Ser, "Cross-updated active noise control system with online secondary path modeling," *IEEE Transactions on Speech and Audio Processing*, vol. 9, no. 5, pp. 598–602, July 2001, doi: 10.1109/89.928924.

[5] B. Widrow and M. E. Hoff Jr., "Adaptive switching circuits," in *1960 IRE WESCON Convention Record*, Part 4, pp. 96–104, August 1960.

[6] I. T. Ardekani and W. H. Abdulla, "FxLMS-based Active Noise Control: A Quick Review," in *APSIPA Annual Summit and Conference*, Xi'an, China, 2011.

[7] 김성현, M. E. Altinsoy, and 김중관, "차량 능동 소음 제어 시스템에서 제어 위치 선택에 따른 소음 저감 성능에 관한 예비 평가," *한국소음진동공학회논문집*, vol. 32, no. 6, pp. 544–551, 2022, doi: 10.5050/KSNVE.2022.32.6.544.

[8] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv preprint arXiv:1704.04861*, 2017.

[9] Q. Kong, Y. Cao, T. Iqbal, Y. Wang, W. Wang, and M. D. Plumbley, "PANNs: Large-Scale Pretrained Audio Neural Networks for Audio Pattern Recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 28, pp. 2880–2894, 2020.

[10] J. F. Gemmeke, D. P. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal, and M. Ritter, "Audio Set: An ontology and human-labeled dataset for audio events," in *Proc. IEEE ICASSP*, 2017, pp. 776–780.