A Major Project report on

**Airis: Smart Glasses for Smarter Learning**

Submitted for the partial fulfilment of the academic requirements for the Award of the Degree of

**BACHELOR OF TECHNOLOGY IN**

**ELECTRONICS AND COMMUNICATION ENGINEERING**

**Submitted By**

**KANDUKOORI SUSHMA 21911A0434**

**P. GANESH 21911A0445**

**PANTANGI ANJALI 21911A0447**

**KATKURI PREM KUMAR 21911A0435**

**Under the Esteemed Guidance of Dr. S. Thulasi Prasad, M.Tech., Ph.D.,**

**Professor**

****

**Department of Electronics and Communication Engineering**

**VIDYA JYOTHI INSTITUTE OF TECHNOLOGY**

**(An Autonomous Institute)**

Accredited by NAAC & NBA, Approved By A.I.C.T.E., New Delhi, Permanently Affiliated to JNTU, Hyderabad) (Aziz Nagar, C.B.Post, Hyderabad -500075)

## 2024-2025



**(Accredited by NAAC, Approved by AICTE New Delhi & Permanently Affiliated to JNTUH)**

## Aziz Nagar Gate, C.B. Post, Hyderabad-500 075

**Department of Electronics and Communication Engineering**

**(Accredited by NBA)**

**CERTIFICATE**

This is to certify that the project report titled “**AIRIS: SMART GLASSES FOR SMARTER LEARNING**” is being submitted by **Kandukoori Sushma (21911A0434), P. Ganesh (21911A0445), Pantangi Anjali (21911A0447)**, and **Katkuri Prem Kumar (21911A0435)** of **IV B.Tech II Semester, Department of Electronics and Communication Engineering**, as a record of bonafide work carried out by them. The results presented in this report are original and have not been submitted to any other University or institution for the award of any degree or diploma.

**INTERNAL GUIDE HEAD OF THE DEPARTMENT**

**Dr. S. Thulasi Prasad Dr**. **M. Rajendra Prasad Professor Professor**

## Department of ECE Department of ECE

**EXTERNAL EXAMINER**

**DECLARATION**

We hereby declare that the project work entitled “**Airis: Smart Glasses for Smarter Learning**” is a genuine and original piece of work carried out by us in the **Department of Electronics and Communication Engineering, Vidya Jyothi Institute of Technology**, under the affiliation of **Jawaharlal Nehru Technological University**, **Hyderabad**. This report is the outcome of our own efforts and research conducted during the course of the project. We affirm that it has not been copied, reproduced, or submitted elsewhere for any academic or professional purpose. All sources of information and assistance have been duly acknowledged.

# PROJECT ASSOCIATES

Ms. Kandukoori Sushma 21911A0434

Mr. P. Ganesh 21911A0445

Ms. Pantangi Anjali 21911A0447

Mr. Katkuri Prem Kumar 21911A0435

**ACKNOWLEDGEMENT**

We would like to express our sincere gratitude to our project guide **Dr. S. Thulasi Prasad, M.Tech., Ph.D.**, Professor, for the exceptional guidance, support, and encouragement provided at every phase of this project.

We are deeply thankful to **Dr. S. Thulasi Prasad, M.Tech., Ph.D.**, Professor and Project Coordinator, Department of ECE, for his timely and insightful guidance that greatly contributed to the successful completion of our work.

Our sincere thanks also go to **Dr. M. Rajendra Prasad, M.E., Ph.D.**, Professor and Head, Department of ECE, for his consistent support and constructive feedback during the course of our study.

We extend our heartfelt gratitude to **Dr. A. Srujana, M.Tech., Ph.D.**, Principal, Vidya Jyothi Institute of Technology, for his motivation and encouragement in the successful completion of our project.

Finally, we express our sincere appreciation to the **faculty and staff of the Electronics and Communication Engineering Department**,Vidya Jyothi Institute of Technology, for their help and cooperation throughout the duration of the project.

KANDUKOORI SUSHMA 21911A0434

P. GANESH 21911A0445

PANTANGI ANJALI 21911A0447

KATKURI PREM KUMAR 21911A0435

# ABSTRACT

This project focuses on the development of **AIris: Smart Glasses for Smarter Learning**, an AI-based wearable technology designed to enhance educational experiences. The glasses facilitate seamless interaction by accepting inputs through various devices (e.g., microphone, camera), activated by a simple button press. The input—whether speech or image—is converted into text using APIs supported by a **Wi-Fi-enabled microcontroller**. The text is then processed by **large language models (LLMs)** like ChatGPT or Gemini to generate insightful responses. These responses are displayed on an OLED screen and projected onto the lens using a reflection mechanism, providing a direct, intuitive view for the user. By combining AI with wearable tech, **AIris** aims to make learning more interactive, accessible, and efficient.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER**  1 | **TITLE**  **INTRODUCTION** | **PAGE NO**  1-2 |
|  | 1.1 Background | 1 |
|  | 1.2 Problem Statement | 1 |
|  | 1.3 Objectives | 1-2 |
| 2 | **LITERATURE SURVEY** | 3-5 |
|  | 2.1 Review of Existing Technologies | 3-4 |
|  | 2.2 Comparative Analysis | 4-5 |
|  | 2.3 Identified Research Gaps | 5 |
| 3 | **ANALYSIS** | 6-7 |
|  | 3.1 Features of Proposed System | 6-7 |
|  | 3.2 Advantages of Proposed System | 7 |
| 4 | **SYSTEM REQUIREMENTS** | 8-12 |
|  | 4.1 Software Requirements | 8 |
|  | 4.2 Hardware Requirements | 8-11 |
|  | 4.3 Functional Requirements | 12 |
|  | 4.4 Components used in Proposed System | 12 |
| 5 | **SYSTEM ARCHITECTURE** | 13-14 |
|  | 5.1 Block Diagram | 13 |
|  | 5.2 Components Description | 13-14 |
| 6 | **METHODOLOGY** | 15-17 |
|  | 6.1 Overview of the Approach | 15 |
|  | 6.2 Detailed Methodology with Schedule | 15-17 |
|  | 6.3 Summary | 17 |
| 7 | **IMPLEMENTATION** | 18-38 |
|  | 7.1 Source Code | 18-38 |
| 8 | **RESULTS** | 39-40 |
| 9 | **CONCLUSIONS AND FUTURE SCOPE** | 41-42 |
|  | **REFERENCES** | 43 |

**LIST OF FIGURES**

**FIGURE NO TITLE PAGE NO**

1. Comparison of Google Glasses and Meta AR Glasses 2
2. Proposed System Design and Component Placement 6
3. Arduino IDE Basic Interface 8
4. Deepgram and Gemini Symbols 8
5. ESP32 38-Pin Pinout 9
6. INMP441 MEMS Microphone Pinout 10
7. 0.96" OLED Module (I2C Interface) 10
8. SPI-Based SD Card Module 11
9. 18650 Battery Shield and 18650 Battery (3.7V) 11
10. Block Diagram of the Smart Glasses System 13
11. Final Prototype of the Smart Glasses 39

2D Diagram of Mirror, OLED, and Lens Setup with 39

1. Approximate Angles

# CHAPTER-I INTRODUCTION

## Background

The integration of artificial intelligence in educational technology has revolutionized learning paradigms. With the ubiquity of smartphones and computers, students are increasingly exposed to distractions that fragment their attention and hinder deep learning. AI-driven wearable devices, such as smart glasses, offer an alternative by providing a hands-free, immersive learning experience. By presenting AI-generated content directly in the user’s field of view, such devices can enhance focus and efficiency.

Recent advancements in microcontrollers, sensor technologies, and AI have made it possible to design compact, efficient, and versatile wearable devices. The ESP32 microcontroller, with its integrated Wi-Fi, offers an excellent platform for edge computing, while lightweight components like the OLED display and I2S MEMS microphone ensure the overall device remains comfortable for prolonged use.

## Problem Statement

Modern learners face significant challenges due to constant digital distractions. Traditional devices require users to switch between applications, thereby interrupting the flow of learning and reducing productivity. The primary problems addressed by this project include:

* + - **Distraction from Ads and Notifications:** Conventional smartphones and computers frequently display ads and notifications that disrupt learning.
    - **Fragmented Learning Experience:** The need to switch contexts between various apps can lead to cognitive overload.
    - **Inefficient Interaction with AI Tools:** Existing voice assistants and smart devices are not optimized for a seamless, real-time learning experience.
    - **Design Limitations:** Current wearable solutions either lack affordability or the necessary integration of high-performance AI capabilities.

## Objectives

The core objectives of AIris are as follows:

1. **Develop a Wearable AI-Powered Device:** Create smart glasses that deliver real-time, context-aware educational content.
2. **Integrate Seamless Speech Recognition:** Utilize an I2S MEMS microphone coupled with advanced speech-to-text APIs to capture and convert voice input with minimal latency.
3. **Implement Advanced AI Processing:** Leverage state-of-the-art language models (e.g., ChatGPT, Gemini) to process queries and generate insightful responses.
4. **Provide a Distraction-Free User Interface:** Design a system that minimizes digital distractions by delivering information directly to the user’s view through a reflective projection.
5. **Ensure Comfort and Usability:** Construct a lightweight, ergonomically designed device that is suitable for extended wear.
6. **Validate Performance through Rigorous Testing:** Establish metrics for evaluating speech recognition accuracy, AI response times, and overall system efficiency.



*Fig 1 : Comparison of Google Glasses and Meta AR Glasses*

# CHAPTER-II LITERATURE SURVEY

## Review of Existing Technologies

In the rapidly evolving field of wearable technology and AI-assisted learning devices, several pioneering projects have laid the foundation for modern smart glasses. This section reviews key studies and commercial products that influenced the design and concept of AIris.

## Google Glass

* + - * **Overview:**

Google Glass was one of the first consumer-focused smart glasses, introduced in 2013. It featured a transparent heads-up display (HUD) and voice-controlled commands, aiming to provide hands-free access to information.

## Key Features:

* + - * + Transparent display for real-time information.
        + Voice command capability for hands-free operation.
        + Lightweight design intended for everyday use.

## Limitations:

* + - * + Privacy concerns and public acceptance issues.
        + Limited AI integration and functionality in real-time applications.

## Relevance to AIris:

While Google Glass provided a proof-of-concept for wearable displays , its limited AI processing and heavy reliance on proprietary software highlighted the need for more integrated and adaptive solutions.

## Meta’s Ray-Ban Smart Glasses

* + - * **Overview:**

Developed in collaboration with Meta, these glasses integrate high-end components such as a 12 MP camera and a multi-microphone array, aimed primarily at content creation and social media.

## Key Features:

* + - * + High-resolution camera and advanced audio capture.
        + Integrated AI assistant for context-aware suggestions.
        + Sleek design with a premium build.

## Limitations:

* + - * + Higher cost and weight compared to budget-friendly alternatives.
        + Complexity in user interface which might not be ideal for distraction- free learning.

## Relevance to AIris:

The advanced sensor suite and AI integration in Meta’s glasses demonstrate the potential for real - time contextual assistance, inspiring Airis to Integrate lightweight yet powerful AI modules tailored for educational use.

## Low-Cost ESP32-Based Smart Glasses

* + - * **Overview:**

Various research projects have explored the use of ESP32 microcontrollers to build budget-friendly smart glasses. These designs often incorporate cameras, basic displays, and simple voice recognition systems.

## Key Features:

* + - * + Cost-effectiveness and ease of prototyping.
        + Utilization of open-source platforms for rapid development.
        + Emphasis on basic functionalities such as object recognition or simple notifications.

## Limitations:

* + - * + Limited processing power and storage.
        + Simplistic user interfaces that are not optimized for continuous learning.

## Relevance to AIris:

The low – cost models highlight the viability of using ESP32 as a core component. However, the lack of robust AI processing and refined display techniques in these models necessitates further enhancements, which AIris addresses by integrating cloud-based AI and advanced speech-to-text modules.

## Speech-to-Text Systems and AI Assistants

* + - * **Overview:**

Recent advancements in speech recognition and AI have been driven by cloud- based APIs such as Google Speech-to-Text and emerging language models like ChatGPT and Gemini.

## Key Features:

* + - * + High accuracy in converting spoken language into text.
        + Low latency and real-time processing capabilities.
        + Continuous learning and adaptation to user accents and dialects.

## Limitations:

* + - * + Dependence on reliable internet connectivity.
        + Challenges in maintaining low power consumption during intensive processing.

## Relevance to AIris:

The integration of robust speech -to- text conversion and AI processing is central to AIris. By leveraging these APIs , AIris can provide immediate, context - aware responses , significantly enhancing the user’s learning experience.

## Comparative Analysis

**Google Glass** was one of the earliest entrants in the smart glasses domain, featuring a transparent heads-up display (HUD) and basic voice command recognition. However, its artificial intelligence integration was minimal, focusing more on providing notifications and simple interactions rather than deep learning or contextual

understanding. It came with a high cost and moderate weight, positioning it for general consumer use but often criticized for its limited utility versus price.

**Meta’s Ray-Ban Glasses** represent a more evolved step, blending a stylish frame with integrated cameras and transparent display elements. They offer more advanced voice controls and moderate AI integration, especially optimized for social media use like hands-free content capture and interaction. However, these glasses are also high in cost and notably heavier, aimed primarily at content creators and social media users rather than the general public.

**ESP32-Based Smart Glasses** are on the opposite end of the spectrum, built with experimentation and affordability in mind. These glasses generally have a basic or minimal display, limited voice recognition capabilities, and almost no AI integration beyond simple microcontroller functions. However, they are low-cost and very lightweight, often used in academic or hobbyist environments for prototyping or specialized tasks.

**AIris**, the proposed model, is designed to overcome the limitations of the above options by offering a cost-effective, lightweight, and education-focused wearable experience. It incorporates an OLED display with reflective projection for clarity without distraction and includes high-accuracy voice-to-text processing for seamless interaction. Most importantly, it features robust AI integration through platforms like ChatGPT or Gemini, enabling real-time assistance, learning, and contextual understanding. AIris aims to serve as a distraction-free, intelligent learning companion, optimized for extended wear and educational enrichment.

## Identified Research Gaps

Despite the advancements made by previous projects, several research gaps remain:

## Integration of AI with Wearable Displays:

* + Most existing models either focus on content delivery or social media interaction without combining robust AI assistance tailored for educational purposes.

## Seamless User Interaction:

* + Many systems require multiple steps or manual intervention to access information, interrupting the flow of learning. AIris aims to provide a truly hands-free, intuitive interaction model.

## Energy Efficiency and Prolonged Use:

* + Maintaining low power consumption while integrating advanced AI modules and high-resolution displays remains a challenge. AIris addresses this by optimizing the hardware-software integration.

## Distraction-Free Design:

* + Current smart glasses often display notifications and other non-essential content. The design philosophy of AIris prioritizes delivering only pertinent, educational content to maintain focus.

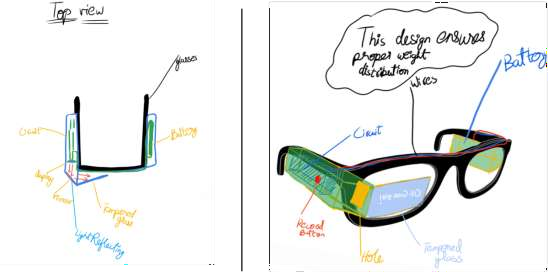
## Scalability and Cost-Effectiveness:

* + High-end smart glasses are not accessible to all segments of learners due to cost and complexity. AIris is designed to be scalable and affordable for widespread adoption.

# CHAPTER-III ANALYSIS

## Features of Proposed System

The AIris smart glasses integrate multiple cutting-edge technologies to deliver an immersive, distraction-free learning experience. Key features include:



*Fig 2 : Proposed System Design and Component Placement*

## Hands-Free Operation:

* + - * Users interact with the system via voice commands and a single tactile button.
      * Eliminates the need to handle smartphones or tablets, reducing distractions.

## Real-Time Speech Recognition and AI Processing:

* + - * Utilizes an I2S MEMS microphone for capturing clear voice inputs.
      * Employs cloud-based speech-to-text APIs that enable near-instant transcription.
      * Integrates advanced AI models (ChatGPT/Gemini) for generating context-aware responses.

## Optimized Display System:

* + - * Incorporates a high-contrast OLED display with a reflective projection mechanism, ensuring that outputs are visible directly in the user’s field of vision.
      * Adjustable brightness and projection settings to suit varying ambient lighting conditions.

## Compact and Ergonomic Design:

* + - * Designed for extended wear with a lightweight structure (approximately 60 grams) and balanced weight distribution.
      * Constructed using durable yet lightweight materials to ensure user comfort.

## Robust Connectivity and Storage:

* + - * ESP32 microcontroller provides Wi-Fi connectivity for seamless cloud integration.
      * Includes an SD card module for data logging and temporary caching.

## Efficient Power Management:

* + - * Powered by a high-capacity 18650 battery with an optimized charging circuit.
      * Low-power consumption design to support continuous usage over extended periods.

## Advantages of Proposed System

The AIris solution offers significant advantages over traditional learning devices and previous wearable systems:

## Distraction Reduction:

* + - * By displaying information directly onto the lens, AIris minimizes the interruptions from notifications and ads common on smartphones and computers.

## Enhanced Learning Experience:

* + - * Hands-free operation and real-time AI assistance enable users to maintain focus, leading to improved comprehension during lectures or study sessions.

## User-Friendly Interaction:

* + - * The intuitive activation via a tactile button, combined with accurate speech-to-text conversion, provides seamless and natural user interactions.

## Cost-Effective and Scalable:

* + - * Leveraging affordable components (e.g., ESP32, MEMS microphone, OLED) keeps production costs low, paving the way for mass-market adoption.

## Versatile Applications:

* + - * Suitable for educational settings (classrooms, libraries), professional environments (workshops, field service), and personal productivity.

## Reliable Performance:

* + - * Optimized for low latency and high accuracy, ensuring that the system responds promptly to user inputs, even under varying environmental conditions.

# CHAPTER-IV SYSTEM REQUIREMENTS

## Software Requirements

The software suite for AIris is designed for real-time operation and seamless interaction with cloud-based services. Requirements include:

## Firmware Development Platform:

* + - * **Arduino IDE:** Used to develop and upload firmware for the ESP32 microcontroller.



*Fig 3 : Arduino IDE Basic Interface*

## Cloud Service Integration:

* + - * API connections for speech-to-text conversion (Deepgram STT).
      * Connectivity with AI language models (Gemini) for query processing.



*Fig 4 : Deepgram and Gemini Symbols*

## Data Analysis Tools:

* + - * Software such as Python or Excel for post-simulation data analysis and performance validation.

## Operating System Compatibility:

* + - * Primarily developed for Windows (using Arduino IDE) with flexibility for other OS when using platform-independent libraries.

## Hardware Requirements

The selection of hardware components is crucial for ensuring that the AIris system delivers high performance, long-lasting durability, and comfortable user experience. Each component has been carefully chosen and detailed below:

## Microcontroller ( ESP32 ):

* + - *Capabilities:* The ESP32 microcontroller is chosen for its integrated Wi-Fi and Bluetooth functionality, which is essential for seamless cloud communication and data exchange.
    - *Performance:* It offers a dual-core processor with high clock speeds, which is sufficient for handling real-time processing of audio signals and managing the communication with cloud APIs.
    - *Memory and Storage:* Adequate RAM and flash memory ensure smooth operation of the firmware, including sensor data processing, API integration, and managing peripheral devices.
    - *Flexibility:* The availability of multiple I/O pins and support for various communication protocols (SPI, I2C, UART) allows integration with a range of components, making it highly versatile for this project.



*Fig 5 : ESP32 38-Pin Pinout*

## Audio Input:

* **I2S MEMS Microphone (e.g., INMP441)**
  + *Functionality:* The I2S MEMS microphone is designed to capture high- quality audio with minimal distortion. Its built-in analog-to-digital converter (ADC) ensures crisp voice signal capture.
  + *Design Benefits:* MEMS microphones are compact, energy-efficient, and have a small form factor, making them ideal for wearable applications.
  + *Noise Reduction:* The microphone's high signal-to-noise ratio (SNR) reduces background interference, ensuring that speech recognition remains accurate even in challenging acoustic environments.
  + *Integration:* Its compatibility with the I2S interface simplifies the digital audio processing in the ESP32, providing a steady stream of digital data for further processing.



*Fig 6 : INMP441 MEMS Microphone Pinout*

## OLED Display (typically 0.96 inch):

* *Display Quality:* OLED displays offer high contrast ratios and deep blacks due to their self-emissive nature, which makes them ideal for displaying clear text and graphics even in varying lighting conditions.
* *Size and Power Consumption:* A 0.96-inch display strikes a balance between screen readability and power efficiency, which is critical for battery-operated wearable devices.
* *Flexibility:* OLED technology allows for a thin and light display module, enabling easy integration into a compact form factor.
* *Interface:* It supports common serial communication protocols (SPI/I2C) that are easily interfaced with the ESP32 for reliable data transfer.



*Fig 7 : 0.96" OLED Module (I2C Interface)*

## Storage Module ( SD Card Module ):

* *Purpose:* The SD card module provides a convenient method for local data logging, which is useful for storing temporary voice-to-text data and system logs.
* *Capacity and Speed:* SD cards offer substantial storage capacity and are capable of high-speed data transfer, ensuring that large volumes of data can be stored and accessed swiftly.
* *Integration:* The module is interfaced with the ESP32 using the SPI communication protocol, offering reliable data exchange for recording system activity and temporary cache storage.



*Fig 8 : SPI-Based SD Card Module*

## Power System ( 18650 Battery with 18650 Battery Shield ):

* + Utilizes a 18650 battery paired with a dedicated battery shield, powered via micro USB to supply the ESP32.
  + The battery shield features six GND pins along with three 3.3V outputs and three 5V outputs, offering flexibility to power various components.
  + Provides a stable, regulated voltage supply that maintains constant levels for sensitive devices like the ESP32 and the OLED display, ensuring reliable system performance.
  + Integrates built-in protection against overcharging and deep discharge, which helps preserve battery longevity and prevent damage.
  + This configuration strikes a balance between achieving prolonged battery life and keeping the overall weight low—an essential consideration for wearable applications.



*Fig 9 : 18650 Battery Shield and 18650 Battery (3.7V)*

## Enclosure Materials:

* + *Ergonomics:* The choice of lightweight materials such as plastic or aluminum alloys (or even a combination with cardboard for prototyping) contributes significantly to user comfort, ensuring that the smart glasses are easy to wear for extended periods.
  + *Durability and Design:* Rigid plastics and aluminum provide the necessary durability and a premium finish while allowing for precise molding or cutting to achieve the desired ergonomic shape.
  + *Customization:* These materials can be easily modified or upgraded with custom 3D-printed parts in future iterations, offering flexibility in design and optimization for miniaturization and integration.

## Functional Requirements

The functional aspects of AIris are geared towards delivering a robust learning experience:

## Voice Command Capture:

* + - * The system must accurately capture and process voice commands in real-time.

## Speech-to-Text Conversion:

* + - * Convert spoken language to text with high accuracy (target ≥95% accuracy).

## AI Response Generation:

* + - * Process the transcribed text to produce contextual and accurate responses.

## Information Display:

* + - * Render AI responses on the OLED display with reflective projection for ease of reading.

## Connectivity and Data Handling:

* + - * Ensure consistent Wi-Fi connectivity and proper data storage on the SD card.

## User Interaction:

* + - * Simple interface management using a tactile button for activating voice commands.

## Components Used in Proposed System

A comprehensive list of key components along with their roles:

## ESP32 Microcontroller:

* + - * Serves as the system’s processing hub and manages communications.

## I2S MEMS Microphone:

* + - * Captures high-fidelity audio input required for speech recognition.

## OLED Display:

* + - * Provides a compact and efficient visual output mechanism.

## SD Card Module:

* + - * Facilitates local storage of data and logs.

## 18650 Battery and Charging Module:

* + - * Supplies consistent power and ensures portable operation.

## Arduino IDE:

* + - * The primary development tool used for coding and flashing firmware onto the ESP32.

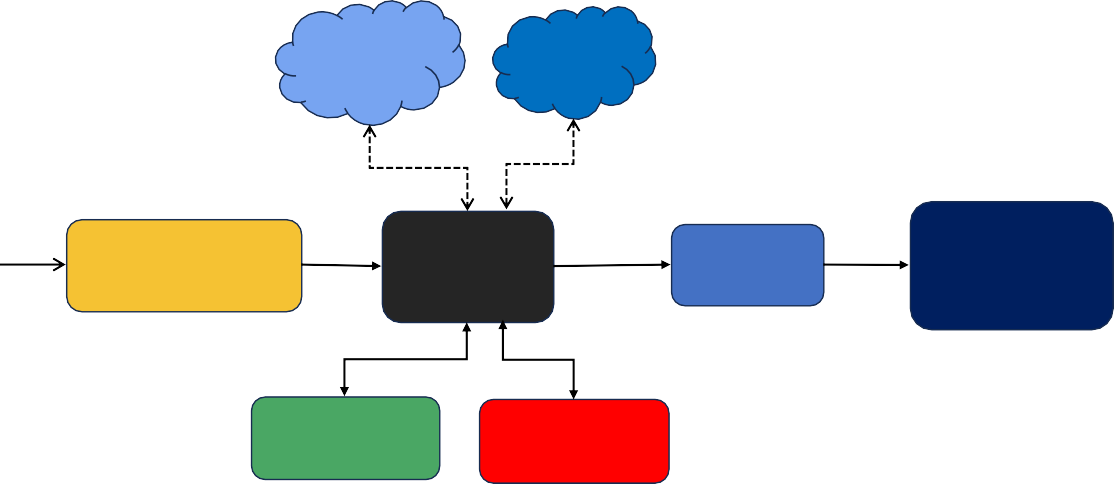
## Additional Components:

* + - * Cables, connectors, enclosures, and mounting hardware ensuring robust integration and assembly.

# CHAPTER-V SYSTEM ARCHITECTURE

## Block Diagram

User Input



Deepgram

Speech −to−text

Gemini LLM

INMP441 MIC MODULE

ESP−32

Microcontroller

0.96−inch OLED

Display

Physical lens for

displaying the reflection of the OLED

SD card module

18650 Battery

charging/power Module

*Fig 10 : Block Diagram of the Smart Glasses System*

## Components Description

A detailed description of the primary components:

## ESP32 Microcontroller:

* + - * **Role:** Serves as the central processing unit handling sensor input, cloud communications, and peripheral control.
      * **Attributes:** Offers low power consumption, integrated Wi- Fi/Bluetooth, and sufficient computing resources to process real-time data.

## I2S MEMS Microphone:

* + - * **Role:** Provides high-quality audio capture necessary for accurate speech-to-text conversion.
      * **Attributes:** Features high sensitivity and low noise characteristics, ensuring crisp audio input.

## OLED Display:

* + - * **Role:** Displays the AI-generated responses.
      * **Attributes:** Compact, high-contrast display ideal for portable, unobstructed viewing.

## SD Card Module:

* + - * **Role:** Stores data logs and temporary caching of interactions.
      * **Attributes:** Enhances system reliability by keeping data locally available.

## 18650 Battery and Charging Module/Battery Sheild:

* + - * **Role:** Provides power to the entire system, ensuring portability and extended use.
      * **Attributes:** Offers high energy density, which supports a sustained operation.

## Arduino IDE:

* + - * **Role:** The software tool used for developing and deploying firmware to the ESP32.
      * **Attributes:** Supports rapid prototyping and code testing with a vast array of libraries.

# CHAPTER-VI METHODOLOGY

## Overview of the Approach

The project methodology was structured into clearly defined phases—from the initial research and component evaluation to the final testing and documentation. The following phases were planned:

1. **Literature Review & Research**
2. **OLED Display & Reflection Setup**
3. **Hardware Selection & Coding**
4. **Hardware Testing with Code**
5. **Testing & Refinement**
6. **Final Implementation & Documentation**

Each phase focused on key deliverables and incremental system improvements.

* 1. **Detailed Methodology with Schedule**
     1. **Literature Review & Research (15/1/2025 – 1/2/2025)**
        + **Objective:**

To establish a thorough understanding of current wearable technologies, speech recognition APIs, and AI-assisted learning devices.

## Activities:

* + - * + Review academic papers, technical articles, and existing smart glasses projects.
        + Analyze different speech-to-text systems and AI integration techniques.

## Outcome:

A comprehensive literature review that informed the system's design and helped define project requirements.

## OLED Display & Reflection Setup (2/2/2025 – 12/2/2025)

* + - * **Objective:**

To identify and test suitable displays and develop a reflective projection mechanism that delivers clear output in the user’s field of view.

## Activities:

* + - * + Experiment with various display modules to determine optimal brightness and resolution.
        + Design and simulate the optical path for reflecting the display on a semi- transparent lens.

## Outcome:

Selection of an OLED display and a validated reflective setup that ensure readability without obstructing the user’s vision.

## Hardware Selection & Coding (13/2/2025 – 28/2/2025)

* + - * **Objective:**

To select the appropriate hardware components and begin the software development process.

## Activities:

* + - * + Evaluate and choose components such as the ESP32 microcontroller, I2S MEMS microphone, SD card module, and power management circuits.
        + Develop initial firmware using the Arduino IDE, focusing on voice capture, data processing, and basic connectivity.

## Outcome:

An integrated hardware configuration with a foundational codebase that supports voice data acquisition and system communication.

## Hardware Testing with Code (1/3/2025 – 25/3/2025)

* + - * **Objective:**

To validate the functionality of the integrated system modules through practical testing.

## Activities:

* + - * + Conduct unit tests for audio capture, Wi-Fi connectivity, and display output.
        + Perform integration tests to ensure that the voice data is correctly transmitted from the microphone to the microcontroller and further to the OLED display.

## Outcome:

Successful identification and resolution of initial issues, ensuring that each module operates as intended under real-world conditions.

## Testing & Refinement (26/3/2025 – 5/4/2025)

* + - * **Objective:**

To carry out iterative testing and refinements aimed at enhancing performance metrics such as latency, speech recognition accuracy, and display clarity.

## Activities:

* + - * + Field tests conducted in diverse environmental conditions (e.g., classrooms, libraries).
        + Gather and incorporate user feedback regarding ergonomics, response time, and overall user experience.
        + Fine-tune the firmware and hardware parameters for optimal performance.

## Outcome:

A refined prototype exhibiting improved reliability and user satisfaction, with measured performance reaching target criteria.

## Final Implementation & Documentation (6/4/2025 – 15/4/2025)

* + - * **Objective:**

To finalize the system integration and produce comprehensive documentation of the development process, testing results, and overall project performance.

## Activities:

* + - * + Final integration of all hardware and software modules into the completed smart glasses prototype.
        + Detailed recording of all modifications, test data, and iterative improvements.
        + Preparation of the final project report and supporting documentation.

## Outcome:

A fully functional AIris prototype along with detailed project documentation, ready for academic submission and future enhancements.

## Summary

The methodology for AIris was designed to be both systematic and modular. By structuring the project into well-defined phases with clear deliverables and timelines, the process ensured that each aspect—from research and design to testing and refinement—was meticulously addressed. Although the actual timeline might have seen some adjustments, this planned schedule underscores the project's rigorous planning and execution.

The project methodology not only facilitated systematic development but also provided a framework for assessing performance metrics, ensuring that the final implementation met the intended specifications and user requirements.

# CHAPTER-VII IMPLEMENTATION

## Source Code

Below three files must be in the same folder named as “Code”.

## Code.ino

// \*\*\* HINT: in case of an 'Sketch too Large' Compiler Warning/ERROR in Ard uino IDE (ESP32 Dev Module):

// -

* + - * select a larger 'Partition Scheme' via menu > tools: e.g. using 'No OTA ( 2MB APP / 2MB SPIFFS) \*\*\*

/\*

Library to be installed

ESP32 Audio I2S - https://github.com/schreibfaul1/ESP32-audioI2S ArduinoJSON - https://arduinojson.org/?utm\_source=meta&utm\_medium=library.p roperties

SimpleTimer - https://github.com/kiryanenko/SimpleTimer

\*/

#define VERSION "\nitsdinothor" #include <Wire.h>

#include <WiFi.h> // only included here

#include <SD.h> // also needed in other tabs (.ino)

#include <Audio.h> // needed for PLAYING Audio (via I2S Amplifier, e.g. MA X98357) with ..

// Audio.h library from Schreibfaul1: https://github.co m/schreibfaul1/ESP32-audioI2S

// .. ensure you have actual version (July 18, 2024 or newer needed for 8bit wav files!)

#include <Arduino.h> #include <WiFiClientSecure.h> #include <ArduinoJson.h> #include <Adafruit\_GFX.h> #include <Adafruit\_SSD1306.h> String text;

String filteredAnswer = "";

#define SCREEN\_WIDTH 128 // OLED display width, in pixels #define SCREEN\_HEIGHT 64 // OLED display height, in pixels

#define OLED\_RESET -1 // Reset pin # (or - 1 if sharing Arduino reset pin)

#define SCREEN\_ADDRESS 0x3C ///< See datasheet for Address; 0x3D for 128x64

, 0x3C for 128x32

const char\* ssid = "WIFI NAME";

// ## INSERT your wlan ssid const char\* password = "WIFI PASSWORD";

// ## INSERT your password

const char\* gemini\_KEY = "YOUR GEMINI API"; //gemini api String Max\_Tokens="50";

#define AUDIO\_FILE "/Audio.wav" // mandatory, filename for the AUDIO record ing

#define pin\_RECORD\_BTN 36

#define LED 2

// --- global Objects ---------- Audio audio\_play;

bool I2S\_Record\_Init();

bool Record\_Start(String filename);

bool Record\_Available(String filename, float\* audiolength\_sec); String SpeechToText\_Deepgram(String filename);

void displayText(String filteredAnswer); void Deepgram\_KeepAlive();

Adafruit\_SSD1306 display(SCREEN\_WIDTH, SCREEN\_HEIGHT, &Wire, OLED\_RESET);

//

void setup() {

// Initialize serial communication Serial.begin(115200); Wire.begin();

// Pin assignments:

pinMode(LED, OUTPUT);

pinMode(pin\_RECORD\_BTN, INPUT); // use INPUT\_PULLUP if no external Pull- Up connected ##

// Hello World Serial.println(VERSION);

// Connecting to WLAN WiFi.mode(WIFI\_STA); WiFi.begin(ssid, password);

Serial.print("Connecting WLAN "); while (WiFi.status() != WL\_CONNECTED) {

Serial.print("."); delay(500);

}

Serial.println(". Done, device connected.");

// Initialize SD card if (!SD.begin()) {

Serial.println("ERROR - SD Card initialization failed!"); return;

}

// initialize KALO I2S Recording Services (don't forget!) I2S\_Record\_Init();

if(!display.begin(SSD1306\_SWITCHCAPVCC, SCREEN\_ADDRESS)) {

Serial.println(F("SSD1306 allocation failed")); for(;;); // Don't proceed, loop forever

}

// INIT done, starting user interaction

Serial.println("> HOLD button for recording AUDIO .. RELEASE button for R EPLAY & Deepgram transcription");

}

void loop() {

if (digitalRead(pin\_RECORD\_BTN) == LOW) // Recording started (ongoing)

{

digitalWrite(LED,HIGH);// RED means 'Recording ongoing'

delay(30); // unbouncing & suppressing button 'click' noise in begin of audio recording

//Start Recording Record\_Start(AUDIO\_FILE);

}

if (digitalRead(pin\_RECORD\_BTN) == HIGH) // Recording not started yet .. OR stopped now (on release button)

{

digitalWrite(LED,LOW); float recorded\_seconds;

if (Record\_Available(AUDIO\_FILE, &recorded\_seconds)) // true once whe

n recording finalized (.wav file available)

{

if (recorded\_seconds > 0.4) // ignore short btn TOUCH (e.g. <0.4 sec s, used for 'audio\_play.stopSong' only)

{

String transcription = SpeechToText\_Deepgram(AUDIO\_FILE); digitalWrite(LED,LOW);

Serial.println(transcription);

//

WiFiClientSecure client;

client.setInsecure(); // Disable SSL verification for simplicity ( not recommended for production)

String Answer = ""; // Declare Answer variable here text = "";

if (client.connect("generativelanguage.googleapis.com", 443)) { String url = "/v1beta/models/gemini-1.5-

flash:generateContent?key=" + String(gemini\_KEY);

String payload = String("{\"contents\": [{\"parts\":[{\"text\":\" " + transcription + "\"}]}],\"generationConfig\": {\"maxOutputTokens\": " + Max\_Tokens + "}}");

// Send the HTTP POST request client.println("POST " + url + " HTTP/1.1");

client.println("Host: generativelanguage.googleapis.com"); client.println("Content-Type: application/json"); client.print("Content-Length: "); client.println(payload.length());

client.println(); client.println(payload);

// Read the response String response;

while (client.connected()) {

String line = client.readStringUntil('\n'); if (line == "\r") {

break;

}

}

// Read the actual response response = client.readString();

parseResponse(response);

} else {

Serial.println("Connection failed!");

}

client.stop(); // End the connection

//

if (filteredAnswer != "") // we found spoken text .. now starting Demo examples:

{

Serial.print("Gemini speaking: "); Serial.println(filteredAnswer); displayText(filteredAnswer);

}

}

}

}

if (digitalRead(pin\_RECORD\_BTN) == HIGH && !audio\_play.isRunning()) // b ut don't do it during recording or playing

{

static uint32\_t millis\_ping\_before;

if (millis() > (millis\_ping\_before + 5000)) { millis\_ping\_before = millis();

digitalWrite(LED,HIGH); // short LED OFF means: 'Reconnection server, can't record in moment'

Deepgram\_KeepAlive();

}

}

}

// Revised section to handle response parsing void parseResponse(String response) {

// Extract JSON part from the response

int jsonStartIndex = response.indexOf("{"); int jsonEndIndex = response.lastIndexOf("}");

if (jsonStartIndex != -1 && jsonEndIndex != -1) {

String jsonPart = response.substring(jsonStartIndex, jsonEndIndex + 1);

// Serial.println("Clean JSON:");

// Serial.println(jsonPart);

DynamicJsonDocument doc(1024); // Increase size if needed DeserializationError error = deserializeJson(doc, jsonPart);

if (error) {

Serial.print("DeserializeJson failed: "); Serial.println(error.c\_str());

return;

}

if (doc.containsKey("candidates")) {

for (const auto& candidate : doc["candidates"].as<JsonArray>()) {

if (candidate.containsKey("content") && candidate["content"].contai nsKey("parts")) {

for (const auto& part : candidate["content"]["parts"].as<JsonArra

y>()) {

if (part.containsKey("text")) { text += part["text"].as<String>();

}

}

'') {

text.trim();

// Serial.print("Extracted Text: ");

// Serial.println(text); filteredAnswer = "";

for (size\_t i = 0; i < text.length(); i++) { char c = text[i];

if (isalnum(c) || isspace(c) || c == ',' || c == '.' || c == '\

filteredAnswer += c;

} else {

filteredAnswer += ' ';

}

}

// filteredAnswer = text;

// Serial.print("FILTERED - "); Serial.println(filteredAnswer);

}

}

} else {

Serial.println("No 'candidates' field found in JSON response.");

}

} else {

Serial.println("No valid JSON found in the response.");

}

}

// Function to display text on OLED void displayText(String text) {

display.clearDisplay(); display.setTextSize(1); display.setTextColor(SSD1306\_WHITE); display.setCursor(0, 10);

display.print(text.c\_str()); // Convert String to char\* display.display();

}

## lib\_audio\_recording.ino

#include "driver/i2s\_std.h" // instead of older legacy #include <driver

/i2s.h>

/\* #include <SD.h> // also needed, but already included in Mai n.ino \*/

// --- defines & macros --------

#ifndef DEBUG // user can define favorite behaviour ('tru e' displays addition info)

# define DEBUG true // <- define your preference here

# define DebugPrint(x); if(DEBUG){Serial.print(x);} /\* do not tou ch \*/

# define DebugPrintln(x); if(DEBUG){Serial.println(x);} /\* do not tou ch \*/

#endif

// --- PIN assignments ---------

#define I2S\_WS 14 // add-

on: L/R pin INMP441 on Vcc is RIGHT channel, connected to GND is LEFT chann el

#define I2S\_SD 35

#define I2S\_SCK 33

// --- define your settings ----

#define SAMPLE\_RATE 16000 // typical values: 8000 .. 44100, us e e.g 8K (and 8 bit mono) for smallest .wav files

service produces lot of wrong words: try 16000

#define BITS\_PER\_SAMPLE 8 // 16 bit and 8bit supported (24 or 32 bits not supported)

TT services than a low 8kHz sample rate

8 bit.

// hint: 8bit is less critical for S

// for fastest STT: combine 8kHz and

#define GAIN\_BOOSTER\_I2S 45 // original I2S streams is VERY sile nt, so we added an optional GAIN booster for INMP441

i2s\_std\_config\_t std\_cfg =

{ .clk\_cfg = // instead of macro 'I2S\_STD\_CLK\_DEFAULT\_CONFIG(SAMPLE\_RATE

),'

{ .sample\_rate\_hz = SAMPLE\_RATE,

.clk\_src = I2S\_CLK\_SRC\_DEFAULT,

.mclk\_multiple = I2S\_MCLK\_MULTIPLE\_256,

},

.slot\_cfg = // instead of macro I2S\_STD\_MSB\_SLOT\_DEFAULT\_CONFIG(I2S\_DAT A\_BIT\_WIDTH\_16BIT, I2S\_SLOT\_MODE\_STEREO),

{ // hint: always using \_16BIT because I2S uses 16 bit slots anyhow (even in case I2S\_DATA\_BIT\_WIDTH\_8BIT used !)

.data\_bit\_width = I2S\_DATA\_BIT\_WIDTH\_16BIT, // not I2S\_DATA\_BIT\_WIDTH

\_8BIT or (i2s\_data\_bit\_width\_t) BITS\_PER\_SAMPLE

.slot\_bit\_width = I2S\_SLOT\_BIT\_WIDTH\_AUTO,

.slot\_mode = I2S\_SLOT\_MODE\_MONO, // or I2S\_SLOT\_MODE\_STERE

O

.slot\_mask = I2S\_STD\_SLOT\_RIGHT, // use 'I2S\_STD\_SLOT\_LEFT

' in case L/R pin is connected to GND !

.ws\_width = I2S\_DATA\_BIT\_WIDTH\_16BIT,

.ws\_pol = false,

.bit\_shift = true, // using [.bit\_shift = true] similar PHILIPS or PC M format (NOT 'false' as in MSB macro) ! ..

.msb\_right = false, // .. or [.msb\_right = true] to avoid overdriven a nd distorted signals on I2S microphones

},

.gpio\_cfg =

{ .mclk = I2S\_GPIO\_UNUSED,

.bclk = (gpio\_num\_t) I2S\_SCK,

.ws = (gpio\_num\_t) I2S\_WS,

.dout = I2S\_GPIO\_UNUSED,

.din = (gpio\_num\_t) I2S\_SD,

.invert\_flags =

{ .mclk\_inv = false,

.bclk\_inv = false,

.ws\_inv = false,

},

},

};

// [re\_handle]: global handle to the RX channel with channel configuration [std\_cfg]

i2s\_chan\_handle\_t rx\_handle;

// [myWAV\_Header]: selfmade WAV Header:

struct WAV\_HEADER

{ char riff[4] = {'R','I','F','F'}; /\* "RIFF"

\*/

long flength = 0; /\* file length

in bytes ==> Calc at end ! \*/

char wave[4] = {'W','A','V','E'}; /\* "WAVE"

\*/

char fmt[4] = {'f','m','t',' '}; /\* "fmt "

\*/

long chunk\_size = 16; /\* size of FMT

chunk in bytes (usually 16) \*/

short format\_tag = 1; /\* 1=PCM, 257=M

u-Law, 258=A-Law, 259=ADPCM \*/

short num\_chans = 1; /\* 1=mono, 2=st

ereo \*/

long srate = SAMPLE\_RATE; /\* samples per second, e.g. 44100 \*/

long bytes\_per\_sec = SAMPLE\_RATE \* (BITS\_PER\_SAMPLE/8); /\* srate \* byte s\_per\_samp, e.g. 88200 \*/

short bytes\_per\_samp = (BITS\_PER\_SAMPLE/8); /\* 2=16- bit mono, 4=16-bit stereo (byte 34) \*/

short bits\_per\_samp = BITS\_PER\_SAMPLE; /\* Number of bi ts per sample, e.g. 16 \*/

char dat[4] = {'d','a','t','a'}; /\* "data"

\*/

long dlength = 0; /\* data length

in bytes (filelength - 44) ==> Calc at end ! \*/

} myWAV\_Header;

bool flg\_is\_recording = false; // only internally used

bool flg\_I2S\_initialized = false; // to avoid any runtime errors in ca se user forgot to initialize

bool I2S\_Record\_Init()

{

// Get the default channel configuration by helper macro (defined in 'i2s

\_common.h')

i2s\_chan\_config\_t chan\_cfg = I2S\_CHANNEL\_DEFAULT\_CONFIG(I2S\_NUM\_AUTO, I2S

\_ROLE\_MASTER);

i2s\_new\_channel(&chan\_cfg, NULL, &rx\_handle); // Allocate a new RX ch annel and get the handle of this channel

i2s\_channel\_init\_std\_mode(rx\_handle, &std\_cfg); // Initialize the chann el

i2s\_channel\_enable(rx\_handle); // Before reading data, start the RX channel first

flg\_I2S\_initialized = true; // all is initialized, checked in procedure Record\_Start()

return flg\_I2S\_initialized;

}

bool Record\_Start( String audio\_filename )

{

if (!flg\_I2S\_initialized) // to avoid any runtime error in case user missed to initialize

{ Serial.println( "ERROR in Record\_Start() - I2S not initialized, call ' I2S\_Record\_Init()' missed" );

return false;

}

if (!flg\_is\_recording) // entering 1st time -

* remove old AUDIO file, create new file with WAV header

{

flg\_is\_recording = true;

if (SD.exists(audio\_filename))

{ SD.remove(audio\_filename); DebugPrintln("\n> Existing AUDIO file rem oved.");

} else {DebugPrintln("\n> No AUDIO file found");}

// Kalo WAV header

File audio\_file = SD.open(audio\_filename, FILE\_WRITE); audio\_file.write((uint8\_t \*) &myWAV\_Header, 44); audio\_file.close();

DebugPrintln("> WAV Header generated, Audio Recording started ... "); return true;

}

if (flg\_is\_recording) // here we land when recording started already -

* task: append record buffer to file

{

// Array to store Original audio I2S input stream (reading in chunks, e

.g. 1024 values)

int16\_t audio\_buffer[1024]; // 1024 values [2048 bytes] <- for the original I2S signed 16bit stream

uint8\_t audio\_buffer\_8bit[1024]; // 1024 values [1048 bytes] <- self calculated values in case BITS\_PER\_SAMPLE == 8

// now reading the I2S input stream (with NEW <I2S\_std.h>) size\_t bytes\_read = 0;

i2s\_channel\_read(rx\_handle, audio\_buffer, sizeof(audio\_buffer), &bytes\_ read, portMAX\_DELAY);

// Optionally: Boostering the very low I2S Microphone INMP44 amplitude (multiplying values with factor GAIN\_BOOSTER\_I2S)

if ( GAIN\_BOOSTER\_I2S > 1 && GAIN\_BOOSTER\_I2S <= 64 ); // check your own best values, recommended range: 1-64

for (int16\_t i = 0; i < ( bytes\_read / 2 ); ++i) // all 1024 v alues, 16bit (bytes\_read/2)

{ audio\_buffer[i] = audio\_buffer[i] \* GAIN\_BOOSTER\_I2S;

}

if (BITS\_PER\_SAMPLE == 8) // in case we store a 8bit WAV file we fill t he 2nd array with converted values

{ for (int16\_t i = 0; i < ( bytes\_read / 2 ); ++i)

{ audio\_buffer\_8bit[i] = (uint8\_t) ((( audio\_buffer[i] + 32768 ) >>8

) & 0xFF);

}

}

// Save audio data to SD card (appending chunk array to file end) File audio\_file = SD.open(audio\_filename, FILE\_APPEND);

if (audio\_file)

{

if (BITS\_PER\_SAMPLE == 16) // 16 bit default: appending original I2S chunks (e.g. 1014 values, 2048 bytes)

{ audio\_file.write((uint8\_t\*)audio\_buffer, bytes\_read);

}

if (BITS\_PER\_SAMPLE == 8) // 8bit mode: appending calculated 1014 v alues instead (1024 bytes, 2048/2)

{ audio\_file.write((uint8\_t\*)audio\_buffer\_8bit, bytes\_read/2);

}

audio\_file.close(); return true;

}

if (!audio\_file)

{ Serial.println("ERROR in Record\_Start() - Failed to open audio file!"

);

return false;

}

}

}

bool Record\_Available( String audio\_filename, float\* audiolength\_sec )

{

// Do nothing to in case no Record was started, recap: 'false' means: 'no thing is stopped' -> no action at all

// important because typically 'Record\_Stop()' is called always in main l oop()

if (!flg\_is\_recording)

{ return false;

}

if (!flg\_I2S\_initialized) // to avoid runtime errors: do nothing in cas e user missed to initialize at all

{ return false;

}

// here we land when Recording is active .. task: stop recording, finaliz e WAV file, return true/done to main loop()

if (flg\_is\_recording)

{

// Update leading WAV haeder - do NOT use 'FILE\_WRITE' we need a 'r+'); see here:

// https://github.com/espressif/arduino-esp32/issues/4028

// https://cplusplus.com/reference/cstdio/fopen/

File audio\_file = SD.open(audio\_filename, "r+"); long filesize = audio\_file.size();

audio\_file.seek(0); myWAV\_Header.flength = filesize; myWAV\_Header.dlen gth = (filesize-8);

audio\_file.write((uint8\_t \*) &myWAV\_Header, 44); audio\_file.close();

flg\_is\_recording = false; // important: this is done only here

\*audiolength\_sec = (float) (filesize-

44) / (SAMPLE\_RATE \* BITS\_PER\_SAMPLE/8); // return Audio length (via refe rence)

DebugPrintln("> ... Done. Audio Recording finished.");

DebugPrint("> New AUDIO file: '" + audio\_filename + "', filesize [bytes

]: " + (String) filesize);

DebugPrintln(", length [sec]: " + (String) \*audiolength\_sec);

return true;

}

}

## lib\_audio\_transcription.ino

#include <WiFiClientSecure.h> // only here needed

/\* #include <SD.h> // library also needed, but already include d in main.ino: \*/

// --- defines & macros --------

#ifndef DEBUG // user can define favorite behaviour ('tru e' displays addition info)

# define DEBUG true // <- define your preference here

# define DebugPrint(x); if(DEBUG){Serial.print(x);} /\* do not tou ch \*/

# define DebugPrintln(x); if(DEBUG){Serial.println(x);} /\* do not tou ch \*/

#endif

// --- PRIVATE credentials & user favorites -----

const char\* deepgramApiKey = "SPEECH TO TEXT API KEY "; // ## INSERT y our Deepgram credentials , CURRENTLY IAM USING DEEPGRAM API ,BUT GOOGLE SPE ECH TO TEXT CAN ALSO BE USED.

#define STT\_LANGUAGE "en-

IN" // forcing single language: e.g. "de" (German), reason: improving reco gnition quality

// keep EMPTY ("") if you want Deepgram to detect & understand 'your' language automatically,

// language abbreviation examples: "en", "e

n-US", "en-IN", "de" etc.

// all see here: https://developers.deepgra

m.com/docs/models-languages-overview

#define TIMEOUT\_DEEPGRAM 12 // define your preferred max. waiting time [sec] for Deepgram transcription response

#define STT\_KEYWORDS "&keywords=KALO&keywords=Janthip&keywords=G oogle" // optional, forcing STT to listen exactly

/\*#define STT\_KEYWORDS "&keywords=KALO&keywords=Sachin&keywords=Go ogle" // optional, forcing STT to listen exactly \*/

// --- global vars ------------- WiFiClientSecure client;

//

String SpeechToText\_Deepgram( String audio\_filename )

{

uint32\_t t\_start = millis();

//

- Connect to Deepgram Server (only if needed, e.g. on INIT and after lost c onnection)

if ( !client.connected() )

{ DebugPrintln("> Initialize Deepgram Server connection ... "); client.setInsecure();

/\* no effect: client.setConnectionTimeout(4000); \*/ if (!client.connect("api.deepgram.com", 443))

{ Serial.println("\nERROR - WifiClientSecure connection to Deepgram Ser ver failed!");

client.stop(); /\* might not have any effect, similar with client.clea r() \*/

return (""); // in rare cases: WiFiClientSecure freezes (library iss

ue?)

}

DebugPrintln("Done. Connected to Deepgram Server.");

}

uint32\_t t\_connected = millis();

// ---------- Check if AUDIO file exists, check file size

File audioFile = SD.open( audio\_filename ); if (!audioFile) {

Serial.println("ERROR - Failed to open file for reading"); return ("");

}

size\_t audio\_size = audioFile.size(); audioFile.close();

DebugPrintln("> Audio File [" + audio\_filename + "] found, size: " + (Str ing) audio\_size );

String socketcontent = ""; while (client.available())

{ char c = client.read(); socketcontent += String(c);

} int RX\_flush\_len = socketcontent.length();

// ---------- Sending HTTPS request header to Deepgram Server

String optional\_param; // see: https://developer s.deepgram.com/docs/stt-streaming-feature-overview

optional\_param = "?model=nova-2-

general"; // Deepgram recommended model (high readability, lowest word error rates)

optional\_param += (STT\_LANGUAGE != "") ? ("&language="+(String)STT\_LANGUA GE) : ("&detect\_language=true"); // see #defines

optional\_param += "&smart\_format=true"; // applies formatting (Pu nctuation, Paragraphs, upper/lower etc ..)

optional\_param += "&numerals=true"; // converts numbers from written to numerical format (works with 'en' only)

optional\_param += STT\_KEYWORDS; // optionally too: keywor d boosting on multiple custom vocabulary words

client.println("POST /v1/listen" + optional\_param + " HTTP/1.1"); client.println("Host: api.deepgram.com"); client.println("Authorization: Token " + String(deepgramApiKey)); client.println("Content-Type: audio/wav"); client.println("Content-Length: " + String(audio\_size));

client.println(); // header complete, now sending binary body (wav byte s) ..

DebugPrintln("> POST Request to Deepgram Server started, sending WAV data now ..." );

uint32\_t t\_headersent = millis();

File file = SD.open( audio\_filename, FILE\_READ );

const size\_t bufferSize = 1024; // best values seem anywhere between 1024 and 2048;

uint8\_t buffer[bufferSize]; size\_t bytesRead;

while (file.available())

{ bytesRead = file.read(buffer, sizeof(buffer));

if (bytesRead > 0) {client.write(buffer, bytesRead);} // sending WAV AUDIO data

}

file.close();

DebugPrintln("> All bytes sent, waiting Deepgram transcription"); uint32\_t t\_wavbodysent = millis();

//

- Waiting (!) to Deepgram Server response (stop waiting latest after TIMEOU T\_DEEPGRAM [secs])

String response = ""; // waiting until available() true and all data co mpletely received

while ( response == "" && millis() < (t\_wavbodysent + TIMEOUT\_DEEPGRAM\*10 00) )

{ while (client.available())

{ char c = client.read(); response += String(c);

}

// printing dots '.' each 100ms while waiting response DebugPrint("."); delay(100);

}

DebugPrintln();

if (millis() >= (t\_wavbodysent + TIMEOUT\_DEEPGRAM\*1000))

{ Serial.print("\n\*\*\* TIMEOUT ERROR - forced TIMEOUT after " + (String) T IMEOUT\_DEEPGRAM + " seconds");

Serial.println(" (is your Deepgram API Key valid ?) \*\*\*\n");

}

uint32\_t t\_response = millis();

// ---------- closing connection to Deepgram

client.stop(); // observation: unfortunately needed, otherwise the 'a udio\_play.openai\_speech() in AUDIO.H not working !

// so we close for now, but will be opened again on ne xt call (and regularly in Deepgram\_KeepAlive())

int response\_len = response.length();

String transcription = json\_object( response, "\"transcript\":" ); String language = json\_object( response, "\"detected\_language\":" ); String wavduration = json\_object( response, "\"duration\":" );

DebugPrintln( " " );

/\* DebugPrintln( "-

* Total Response: \n" + response + "\n"); // ### uncomment to see the comp lete server response \*/

DebugPrintln( "-

* Latency Server (Re)CONNECT [t\_connected]: " + (String) ((float)((t\_conn

ected-t\_start))/1000) );; DebugPrintln( "-

* Latency sending HEADER [t\_headersent]: " + (String) ((float)((t\_head ersent-t\_connected))/1000) );

DebugPrintln( "-

* Latency sending WAV file [t\_wavbodysent]: " + (String) ((float)((t\_wavb odysent-t\_headersent))/1000) );

DebugPrintln( "-

* Latency DEEPGRAM response [t\_response]: " + (String) ((float)((t\_resp onse-t\_wavbodysent))/1000) );

DebugPrintln( "=> TOTAL Duration [sec]: ..................... " + (String

) ((float)((t\_response-t\_start))/1000) );

DebugPrintln( "=> RX data prior request [bytes]: " + (String) RX\_flush\_le n );

DebugPrintln( "=> Server detected audio length [sec]: " + wavduration );

DebugPrintln( "=> Server response length [bytes]: " + (String) response\_l en );

DebugPrintln( "=> Detected language (optional): [" + language + "]" ); DebugPrintln( "=> Transcription: [" + transcription + "]" ); DebugPrintln( " \n" );

return transcription;

}

void Deepgram\_KeepAlive() // should be called each 5 seconds about ( to overcome the default autoclosing after 10 secs)

{

uint32\_t t\_start = millis(); DebugPrint( "\* Deepgram KeepAlive | " );

//

- Connect to Deepgram Server (on INIT and every time in case closed)

if ( !client.connected() )

{ DebugPrint("NEW Reconnection ... "); client.setInsecure();

/\* no effect: client.setConnectionTimeout(4000); \*/ if (!client.connect("api.deepgram.com", 443))

{ Serial.println("\n\* PING Error - Server Connection failed.");

/\* no effect: client.clear(); client.stop(); \*/

return; // in rare cases: WiFiClientSecure freezes (library issue?)

}

DebugPrint( "Done, connected. --> Connect Latency [sec]: "); DebugPrintln( (String)((float)((millis()-t\_start))/1000) ); return; // done, on next cycle (after e.g. 5 secs) we ping data

}

uint8\_t empty\_wav[] = {

0x52,0x49,0x46,0x46, 0x40,0x00,0x00,0x00, 0x57,0x41,0x56,0x45,0x66,0x6D,0 x74,0x20,

0x10,0x00,0x00,0x00,0x01,0x00,0x01,0x00,0x80,0x3E,0x00,0x00,0x80,0x3E,0x0

0,0x00,

0x01,0x00,0x08,0x00,0x64,0x61,0x74,0x61, 0x14,0x00,0x00,0x00, 0x80,0x80,0 x80,0x80,

0x80,0x80,0x80,0x80,0x80,0x80,0x80,0x80,0x80,0x80,0x80,0x80,0x80,0x80,0x8

0,0x80 };

client.println("POST /v1/listen HTTP/1.1"); client.println("Host: api.deepgram.com");

client.println("Authorization: Token " + String(deepgramApiKey)); client.println("Content-Type: audio/wav"); client.println("Content-Length: " + String(sizeof(empty\_wav))); client.println(); // header complete, now sending wav bytes .. client.write(empty\_wav, sizeof(empty\_wav));

// ---------- RX - Receiving DATA from Deepgram Server ....

// keep in mind: we do NOT want to wait until 'related' answer received ( 'available()' needs about ~1sec to become TRUE)

// means: typically we read the data from earlier TX ;) .. doesn't matter as long we flush before final user request :)

String response = ""; while (client.available())

{ char c = client.read(); response += String(c);

} int RX\_len = response.length();

// ---------- NOT closing Deepgram connection

// keep in minsd: we do NOT close connection, so audio\_play.openai\_speech () in AUDIO.H won't work (library issue)

// connection will be closed in SpeechToText\_Deepgram(), allowing to play audio after user voice transcription done

/\* client.stop(); \*/

DebugPrint( "TX (WAV): " + (String) sizeof(empty\_wav) + " bytes | " );

DebugPrint( "RX (TXT): " + (String) RX\_len + " bytes --> " ); DebugPrintln( "Total Latency [sec]: " + (String) ((float)((millis()-

t\_start))/1000) );

}

//

// JSON String Extract: Searching [element] in [input], example: element = "transcript": -> returns content 'How are you?'

String json\_object( String input, String element )

{ String content = "";

int pos\_start = input.indexOf(element);

if (pos\_start > 0) // if element fou

nd:

{ pos\_start += element.length(); // pos\_start poin ts now to begin of element content

int pos\_end = input.indexOf( ",\"", pos\_start); // pos\_end points to ," (start of next element)

if (pos\_end > pos\_start) // memo: "garden"

.substring(from3,to4) is 1 char "d" ..

{ content = input.substring(pos\_start,pos\_end); // .. thats why w e use for 'to' the pos\_end (on ," ):

} content.trim(); // remove optiona

l spaces between the json objects

if (content.startsWith("\"")) // String objects typically start & end with quotation marks "

{ content=content.substring(1,content.length()-

1); // remove both existing quotation marks (if exist)

}

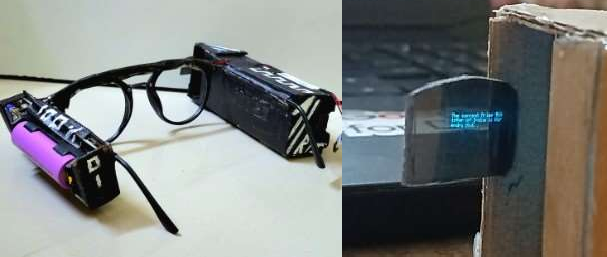
}

return (content);

}

# CHAPTER-VIII RESULTS

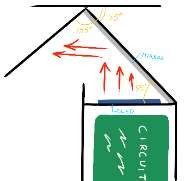
The developed system was thoroughly tested and demonstrated the intended functionalities successfully. On pressing the input button, the **MEMS microphone** activates and captures the user’s voice. This audio signal is transmitted to the **Deepgram API**, which processes it in real-time into text. The converted text is saved temporarily in an **SD card** and is then used as an input for the **Gemini AI API**. The Gemini model returns an AI-generated response that is displayed on a compact **OLED screen**.



*Fig 11 : Final Prototype of the Smart Glasses*

The interaction among the hardware components (microphone, ESP32, SD card module, OLED) and the software services (Deepgram, Gemini API) occurred smoothly without notable latency or errors. The displayed output matched the expected AI-generated responses, confirming the correctness and efficiency of the speech processing and text generation pipeline.

The system also satisfied the power and connectivity constraints during testing. The ESP32 module maintained stable Wi-Fi connections for cloud-based API communication, and the system components operated without noticeable overheating or instability.



*Fig 12 : 2D Diagram of Mirror, OLED, and Lens Setup with Approximate Angles*

## Display Observation

* The OLED output was bright and readable under standard indoor lighting conditions.
* The mirror successfully reflected the text to the transparent lens, allowing it to float in front of the user visually.
* Though no **magnifying lens** was used in this iteration (due to scale and simplicity), the clarity was acceptable, and future versions may include a magnifier to enhance readability.
* The system avoids projection onto the retina or any invasive method, ensuring it remains safe for prolonged viewing.

# CHAPTER-IX CONCLUSION AND FUTURE SCOPE

## Conclusion

The AIris project demonstrates a successful implementation of a wearable, voice- interactive display system that integrates real-time speech processing, AI-generated responses, and a novel optical projection mechanism. Key functionalities were verified through comprehensive testing:

* **Voice Input and Processing:** On activation via a tactile button, the MEMS microphone captured voice commands effectively. The Deepgram API converted speech to text in real time, and the Gemini AI API delivered accurate, contextual responses.
* **Data Management and Display:** The processed text was temporarily stored on an SD card and subsequently displayed on an OLED screen. The optical setup—comprising a mirror and a transparent lens—successfully projected the output into the user's field of view.
* **Optical Arrangement:** Measurements confirmed that the mirror was positioned at approximately 55° relative to the top boundary, with the OLED mounted perpendicularly (around 90°), and the lens arranged to form an effective projection angle of roughly 155° with the top boundary. This configuration adhered to the law of reflection, ensuring that the projected content was clear and legible.

Overall, the project meets its primary objectives by providing a functional prototype that delivers real-time AI-powered responses through a wearable optical display system. The successful integration of hardware and cloud-based services validates the design approach and confirms the system’s potential in educational and professional settings.

## Future Scope

While the current prototype fulfills the core objectives, several enhancements can further improve the system’s performance, compactness, and user-friendliness:

## Custom PCB Design:

Replace the provisional breadboard setup with a custom-designed printed circuit board (PCB) to minimize bulk, enhance durability, and streamline component integration.

## Enclosure Optimization via 3D Printing:

Develop a tailored, ergonomic enclosure using 3D printing. This would improve aesthetics, ensure better component protection, and provide a more comfortable wear for prolonged use.

## Incorporation of a Magnifying Lens:

Although the current setup successfully projects the OLED display output on a transparent lens, integrating a magnifier could further enhance readability, especially in challenging lighting conditions or for users who require larger text display.

## Advanced User Interaction:

Future iterations might include additional input methods—such as gesture recognition or touch controls—to provide more seamless and intuitive interactions.

## Power Optimization:

Further refinement of the power management system could extend battery life, enabling longer periods of autonomous operation, which is crucial for continuous use in real-world scenarios.

## Enhanced AI Capabilities:

Integrating more sophisticated AI models could provide more personalized and contextually relevant responses. Over time, the system may also incorporate learning algorithms that adapt to individual user preferences.

## Scalability and Mass Production:

With iterative improvements and rigorous testing, the design can be scaled for mass production, making AIris an accessible solution for a wider audience in both academic and professional environments.

# REFERENCES

* T. Y. G, N. B. G, S. R. M, R. K and L. M. Mohamed. (2024) **“Advancements in Speech-to-Text Systems for the Hearing Impaired”,** *2024 IEEE North Karnataka Subsection Flagship International Conference (NKCon)*, Bagalkote, India, 2024.
* D. Rossos, A. Mihailidis and B. Laschowski. (2024) **“AI-Powered Smart Glasses for Sensing and Recognition of Human-Robot Walking Environments”**, *2024 10th IEEE RAS/EMBS International Conference for Biomedical Robotics and Biomechatronics (BioRob)*, Heidelberg, Germany, 2024.
* K. Wu and Z. Xue. (2023) **“Construction of Smart Glasses Assisted Road Sign Recognition System”**, *2023 IEEE International Conference on Electrical, Automation and Computer Engineering (ICEACE)*, Changchun, China, 2023.
* S. Das, S. Saxena and N. K. Rout. (2021) **“Low-Cost Smart-Glass using ESP-32”**, *2021 IEEE 2nd International Conference on Applied Electromagnetics, Signal Processing, & Communication (AESPC)*, Bhubaneswar, India, 2021.
* N. M. Kumar, P. R. Krishna, P. K. Pagadala and N. M. Saravana Kumar. (2018) **“Use of Smart Glasses in Education-A Study”**, *2018 2nd International Conference on I- SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, Palladam, India, 2018.