FoS(Focus on Speaking)

Potato Savior

https://github.com/april2901/ai-assistant-for-presentation

Sangyoon Kwon
Department of Computer Science
Backend Development
Seoul, Republic of Korea
is0110@hanyang.ac.kr

Dohoon Kim
Department of Computer Science
Backend Development
Seoul, Republic of Korea
april2901@hanyang.ac.kr

Daeun Lee
Division of Business Administration
UI Design, PM
Seoul, Republic of Korea
shinran2929@hanyang.ac.kr

Hyeyun Kwon
Department of Information Systems
Frontend Development
Seoul, Republic of Korea
herakwon1124@hanyang.ac.kr

Seohyun Kim
Department of Information Systems
Frontend Development
Seoul, Republic of Korea
dianwls0326@hanyang.ac.kr

Minhyuk Jang
Division of Business Administration
UI Design, PM
Seoul, Republic of Korea
jmh12230@hanyang.ac.kr

Abstract— In modern business and educational environments, meetings and presentations are key means of communication, and their importance continues to grow. However, presenters and participants often experience cognitive overload as they manage speech delivery, script reference, slide transitions, and time management while also trying to follow complex discussion flows and decisions. This leads to topic drift, loss of focus, and unclear outcomes. To address these issues, this project proposes an **LG display–linked real-time meeting AI prompter** that extends a traditional teleprompter into an active, context-aware assistant. The system listens to participants' speech, interprets the meeting context in real time, and presents the next required information—such as agenda structure, current topic, decisions, and action items—on LG displays, while providing a private coaching dashboard for the presenter or host. Core features include real-time STT, flexible speech-to-script matching, keyword omission detection, a real-time feedback dashboard, agenda visualization, decision and action-item extraction, and fact-check widgets. A Meeting Summary Report summarizes key topics, ideas, decisions, and action items after the session. Through these functions, the project aims to improve both individual presentation quality and overall meeting efficiency, and to explore integration within LG's smart office ecosystem.

Keywords—Speech Recognition, Real-Time STT, Script Synchronization, Real-Time Teleprompter, Slide Automation, Agenda Tracking, Presentation Feedback, Human-Computer Interaction, Meeting Intelligence

Role Assignment -

Roles	Name	Task description and etc.
User	Daeun Lee, Minhyuk Jang	Tests the prototype from the user's perspective, focusing on interface usability, speech synchronization accuracy, and overall user experience. Provides qualitative feedback for refinement.
Customer	LG Electronics (Assumed Client)	Defines requirements for smart office presentation support software and evaluates its feasibility for integration with LG's webOS-based business ecosystem.
Software Developer	Sangyoon Kwon, Dohoon Kim (Backend), Hyeyun Kwon, Seohyun Kim (Frontend)	Responsible for system implementation including backend server logic, database management, API communication, and frontend interface development. Ensures real-time synchronization and stable slide automation.
Development Manager& UI Designer	Daeun Lee, Minhyuk Jang	Oversees project planning, documentation, and communication between development teams. Manages task allocation, schedule tracking, design of interface and quality assurance.

I. INTRODUCTION

A. Challenges in Modern Presentations and Meetings

In professional and academic settings, presentations and meetings have become essential tools for sharing ideas and making decisions. Yet presenters and facilitators must simultaneously handle speech delivery, script reference, slide transitions, and time tracking, while participants struggle to follow complex discussion flows and remember key points. This often causes interruptions in the presentation flow, omission of important content, topic drift during discussions, and unclear conclusions, ultimately reducing communication efficiency.

B. Limitations of Existing Solutions

Existing tools such as teleprompters, timers, and subtitle features mainly provide static information or simple transcription. They are useful for displaying text but do not actively intervene in real time to prevent topic drift or support decision alignment. Many AI-based meeting services focus on post-meeting summaries or minutes, which help review what happened but do not improve the efficiency of the meeting while it is in progress. As a result, core issues such as cognitive overload and live meeting inefficiency remain unresolved.

C. Project Goals and Proposed Solution

This project proposes an integrated support system that covers preparation, live delivery, and post-meeting feedback. The core concept is an **"LG Display–Linked Real-Time Meeting AI Prompter."** The system continuously listens to meeting audio, analyzes the semantic context of each utterance, and surfaces what the meeting needs next: agenda structure, current topic, decisions, action items, and fact-check results. At the same time, it provides a private teleprompter and coaching dashboard for the presenter or host, helping with script tracking, omission alerts, pacing, and gesture suggestions. The ultimate goal is to reduce cognitive load and improve meeting focus and decision-making speed.

D. Dual-Screen Architecture for LG Displays

To support both individual coaching and shared awareness, the system adopts a dual-screen architecture. **Screen 1 (Presenter Dashboard)** is shown on the presenter's personal device (e.g., LG Gram) and provides a private teleprompter, omission alerts, pace metrics, and AI suggestions. **Screen 2 (Shared Meeting Board)** is shown on LG signage, LG One:Quick, or conference room TVs and visualizes slides, the real-time agenda map, decisions and action items, and fact-check widgets for all participants. This separation allows the presenter to receive rich guidance without overwhelming the audience, while participants share a clear view of where the meeting is and what has been decided.

II. Requirements

A. Before Presentation

This phase focuses on the preparation process before a presenter begins their presentation. Users interact with the system to upload materials, create a script, and adjust content to fit the presentation environment.

1) Slide-Script Alignment Recognition and Consulting

When a user uploads a PPTX or PDF file, the system extracts textual and visual elements using python-pptx and the Google Vision API (OCR). A multimodal LLM processes these elements to interpret textual and graphical contexts, generating a coherent draft script for each slide.

- Acceptance Criteria

• OCR text recognition accuracy ≥ 95%
• Draft script grammatical accuracy ≥ 95%
• Slide–text coherence ≥ 95%

- Input & Output

• Input**:** Presentation file (.pptx, .pdf)
• Output**:** Structured text/image metadata, draft script (3–6 sentences per slide)

- Constraints

• Max 100 slides
• Max file size 200 MB
• Max 10 images per slide
• Supported formats: PPTX, PDF

2) Presentation Environment-Specific Script Adjustment

The system adjusts the script's vocabulary level, tone, and length based on the audience type (non-expert / practitioner / expert), target presentation time, and speaker's pace. Using TensorFlow.js-based vision models, audience facial expressions are analyzed every 3 seconds to compute a "focus score" (0–100). When time is running short or audience engagement decreases, an LLM provides real-time summaries or interactive remarks.

- Acceptance Criteria

• Script adjustment time ≤ 5s
• Focus detection accuracy ≥ 85%
• Timing deviation ≤ ±5%
• Script fluency ≥ 90%

- Input & Output

• Input**:** Audience type, target duration, speech rate(WPM), tone, audience video data
• Output**:** Adjusted script (.txt), recommended timing table, real-time teleprompter feedback

- Constraints

• Camera ≥ 720p
• Max 10 audience members detectable
• LLM request frequency ≤ 1 per 10 s
• Presentation ≤ 60 min

B. Live Delivery Phase

This phase involves real-time interaction between the user and the system during the actual presentation. The system detects the presenter's speech and performs instant support tasks like synchronization, feedback, and suggestions.

1) Real-time Teleprompter(Meeeting Mode Support)

The system transcribes the presenter's and participants' speech in real time using Google Cloud or Naver Clova STT and aligns it with the prepared script via KoSentence-BERT semantic similarity. The current sentence is visually highlighted on the teleprompter. In meeting mode, the STT pipeline can capture speech from multiple people in the

room as a single mixed audio stream; downstream modules operate on this combined transcript.

- Acceptance Criteria

• Speech–script synchronization delay ≤ 1 s
• Highlight accuracy ≥ 95%
• Alignment deviation ≤ 1 sentence

- Input & Output

• Input**:** Microphone audio (.wav, .mp3, ≥16 kHz), script file (.txt)
• Output**:** Real-time highlighted script text and STT logs

- Constraints

• Session length ≤ 60 min
• STT throughput ≥ 50 words/s
• API cost ≈ $0.006/min

2) Automatic Slide Transition

Through the Microsoft PowerPoint COM API, slides are automatically advanced when the script reaches predefined transition points.

- Acceptance Criteria

• Transition latency ≤ 0.5 s
• Transition accuracy ≥ 95%
• Failure rate ≤ 5%

- Input & Output

• Input**:** Slide file (.pptx), predefined transition IDs
• Output**:** Automatically advanced slide display

- Constraints

• Max 100 slides
• Requires PowerPoint 2016 or later

3) Flexible Speech-to-Script Matching

The system maintains synchronization even when speech deviates lexically from the script. Primary matching uses KoSentence-BERT vector similarity (threshold ≥ 0.8), followed by secondary LLM-based contextual verification if necessary.

- Acceptance Criteria

• Matching success rate ≥ 90%
• False match rate ≤ 5%
• Matching latency ≤ 0.3 s per sentence

- Input & Output

• Input**:** STT transcript, script text
• Output**:** Matched sentence ID and highlight position

- Constraints

• LLM call limit ≤ 1 per second
• STT buffering ≤ 5 s

4) Key Content Omission Detection

Detects missing predefined key phrases using cosine similarity (threshold ≥ 0.75) and alerts the presenter within 3 seconds.

- Acceptance Criteria

• Detection precision ≥ 95%
• False alarm ≤ 5%
• Alert delay ≤ 2 s

- Input & Output

• Input**:** STT transcript, key phrase list (≤50)
• Output**:** Omission alert and log file

- Constraints

• Max 500 sentences compared
• STT buffering interval: 5 s

5) Real-time Script Reconstruction

When omissions are detected, missing segments are asynchronously sent to an LLM that generates supplementary sentences within 5 seconds. Approved sentences are integrated into the script in real time.

- Acceptance Criteria

• Supplement generation ≤ 5 s
• Contextual coherence ≥ 90%
• Integration success ≥ 90%

- Input & Output

• Input**:** Missing sentence ID, context text, LLM API key
• Output**:** Supplementary sentence, updated script

- Constraints

• Max 10 API calls per minute
• Sentence length ≤ 100 characters

6) Real-time Presenter Dashboard

The system visualizes metrics such as words per minute (WPM), voice volume, and progress rate using the Web Audio API, and applies TensorFlow Lite models for basic emotion recognition (e.g., tension/calmness).

- Acceptance Criteria

• Data refresh ≤ 2 s
• Emotion inference error ≤ ±5%
• Visualization accuracy ≥ 95%

- Input & Output

• Input**:** Audio stream, STT logs
• Output**:** Live dashboard showing progress, WPM, emotional state

- Constraints

• Sampling rate ≥ 16 kHz
• Dashboard latency ≤ 1 s

7) Speech Gesture Suggestions

Before the presentation, the system analyzes slide images with a multimodal LLM to detect key visual elements (graphs, photos, diagrams) and map them to related keywords. During speech, when such keywords appear in STT, gesture icons (e.g., pointing, emphasis) are displayed on the teleprompter.

- Acceptance Criteria

• Suggestion latency ≤ 2 s
• Gesture relevance ≥ 85%
• Recognition accuracy ≥ 90%

- Input & Output

• Input**:** Slide images, script keywords
• Output**:** Gesture icons displayed on teleprompter

- Constraints

• Max 3 visual mappings per keyword
• Display duration: 2–3 s

8) **Real-Time Context and Intent Tagging**

The system analyzes STT text in real time and classifies each utterance according to its intent and type. This information is used as the foundation for the agenda map, decision board, and fact-check triggers.

- Supported Tags

• General comment
• Idea proposal
• Negative/Positive feedback
• Decision
• Request / Action Item
• Question
• Fact-check request

- Acceptance Criteria

• all utterances receive at least one tag from the predefined set
• Intent classification accuracy ≥ 85% on test samples

- Input & Output

• Input: STT transcript segmented into utterances
• output: Tagged utterance stream (text + tag(s) + timestamp)

- Constraints

• Classification latency ≤ 0.5s per utterance
• the tagging model must operate within the WebSocket round-trip time budget

9) ***Real-Time agenda Map***

Using the tagged utterance stream, the system builds a real-time agenda map to prevent topic deviation and improve shared awareness. Each emerging topic is registered as a node in a network graph displayed on **Screen 2**

- Acceptance Criteria

• Screen 2 displays a live STT log and its mapping to agenda nodes
• Utterances tagged as “Idea,” “Decision,” or “Action Item” are grouped under appropriate agenda nodes
• Nodes are color-coded by agenda type
• The active topic is clearly highlighted
• Node detail view shows STT snippet and timestamp

- Input & Output

• Input**:** Tagged utterance stream (from Requirement 8), semantic embeddings
• Output**:** Real-time agenda network graph on Screen2

- Constraints

• Graph update interval ≤ 2s
• Max 30 agenda nodes per session

10) ***Dual-Screen synchronizatino between Presenter Shared Display***

The system keeps Screen 1(presenter Dashboard) and Screen 2 synchronized while respecting privacy boundaries

- Acceptance Criteria

• Slide transition latency between Screen 1 and Screen 2 ≤ 0.5 s
• No private elements (teleprompter text, omission alerts, AI suggestions) appear on Screen 2

- Input & Output

• Input**:** Slide control events, layout state, synchronization messages
• Output**:** Consistent view of the current slide and agenda state on Screen 2

- Constraints

• WebSocket synchronization interval ≤ 1s

11) ***Real-time Decisions and Action Item Widget***

The system captures utterances tagged as decision or action item and surfaces them in a dedicated widget on screen 2, often placed below or beside the agenda map.

- Acceptance Criteria

• Detection coverage for decision-like and action-like utterances ≥ 90% on test scenarios
• New items appear in the list within 2 seconds of the utterance

- Input & Output

• Input**:** Tagged utterance stream (Decision / Action Item tags)
• Output**:** Real-time decision & action-item list on Screen 2

- Constraints

• Max 100 items per session
• Each item stored with its text content and timestamp, with a link to the original transcript segment

12) ***Real-Time Fact-Check and Research Widget***

When the system detects a **Fact-check request** tag, it triggers a lightweight research pipeline (RAG or web search) and surfaces the result on Screen 2

- Acceptance Criteria

• Successful keyword extraction for at least 90% of fact-check requests
• Research results displayed within 5 seconds

- Input & Output

• Input**:** Tagged utterance stream (Fact-check request tag), knowledge base or web search API
• Output**:** Fact-check result widget with short answer and source link(s)

- Constraints

• Max 30 fact-check queries per session
• Each query result limited to brief, conference-friendly summaries

C. Post-Presentation Phase

This phase involves the user receiving feedback on their presentation and conducting a Q&A session after the presentation has concluded.

1) *Q&A Auto-Response*

In Q&A mode, the system uses Retrieval-Augmented Generation (RAG) to search a pre-built database and generate 2–3 candidate answers, each referencing supporting slides or pages.

- Acceptance Criteria

• Answer generation ≤ 5 s
• Relevance score ≥ 0.85
• Slide reference accuracy ≥ 98%

- Input & Output

• Input**:** Question (speech/text), presentation DB (JSON)
• Output**:** 2–3 candidate answers with referenced slides

- Constraints

• Max 20 questions per session
• Max 300 tokens per answer
• RAG cosine similarity ≥ 0.8

2) *Presentation and Meeting Analysis Report*

After the meeting or presentation, the system automatically analyzes collected data (speech logs, agenda map, decisions, action items, and referenced research results) and generates a Meeting Summary Report. This report covers time management, speech habits, content delivery, and meeting outcomes such as major topics, ideas, decisions, action items, and external facts referenced during the discussion.

- Acceptance Criteria

• Report generation ≤ 10 s
• Analysis accuracy ≥ 95%
• User satisfaction ≥ 4.2/5.0

- Input & Output

• Input**:** Speech logs, slide transitions, emotion data, agenda map, decision/action-item list, fact-chcek logs
• Output**:** Analysis report (.html,.pdf)

- Constraints

• Max presentation time: 60 min
• Max 100,000 words processed

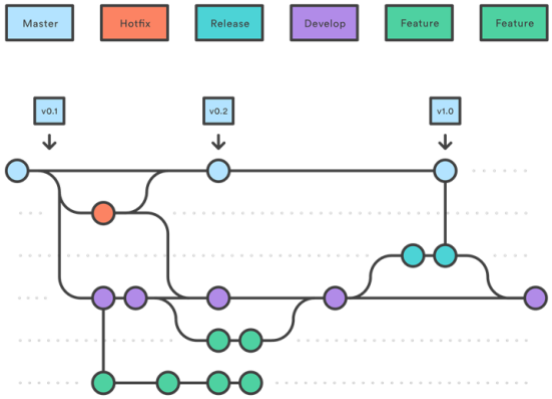
III. VERSION CONTROL SYSTEM

To manage source code and documentation, a version control system based on **Git** was established.
A public repository named **"ai-assistant-for-presentation"** was created on **GitHub**.

The following source code and documents have been uploaded to the repository:

- All backend and frontend source code
- Shared documents including this file (*project_documentation.md*) and other design files
- Configuration files and project-related assets

All team members (development, project management, and UI/UX design) have been granted access to the repository. For efficient GitHub management, the team will adopt a **branch management strategy** as illustrated in the diagram below.



IV. Development Environment

A. Choice of Software Development Platform

1) *Platform Selection and Rationale*

The project adopts a web-based client–server architecture as its primary development and deployment platform. The web environment ensures platform-independent accessibility without requiring users to install additional software, while providing seamless integration with cloud-based APIs such as Google Cloud Speech-to-Text and large-language-model (LLM) services. Modern web technologies, including WebSockets, Web Audio API, and TensorFlow.js, enable the implementation of essential real-time features such as live teleprompting and synchronized feedback dashboards.

2) *Programming Languages and Rationale*

- **Backend:** Node.js–based API Routes are used to handle REST APIs and real-time communication (e.g., WebSocket or streaming endpoints). Node.js provides a non-blocking, event-driven runtime that is well suited for real-time applications such as teleprompters and meeting assistants. It also enables tight integration with the React frontend and offers a rich JavaScript/TypeScript ecosystem for calling external AI services (e.g., STT APIs and LLM APIs) and managing asynchronous workflows.
- Frontend: TypeScript/JavaScript (Node.js 20 or higher) with React 18.2. JavaScript is the only natively supported browser language and is indispensable for client-side interaction. React combined with TypeScript supports modular, maintainable UI components and ensures type safety.

3. *Cost Estimation*
   The estimated total development cost is approximately USD 30.00, as summarized below:

- Hardware: Personal laptops (MacBook Air/Pro) – USD 0.00
- Software and IDE: Visual Studio Code (free), Cursor Pro (USD 20 per month)
- Cloud Services and APIs: AWS Free Tier, GCP STT and Vision API, OpenAI GPT Realtime Mini (approx. USD 10)

4. *Development Environment Details*

- Operating Systems: Windows 11, macOS 14 (Sonoma)
- IDEs: Visual Studio Code (v1.90 or higher), Cursor (v1.7 or higher)
- Version Control: Git (v2.39 or higher) and GitHub public repository ("ai-assistant-for-presentation")
- Backend Stack: Python 3.11+, FastAPI 0.110+, PostgreSQL 16 (Render hosted)
- Frontend Stack: Node.js 20.10+, npm 10.2+, React 18.2+, TypeScript 5.2+
- Major Libraries: python-pptx, sentence-transformers, websockets, TensorFlow.js, Web Audio API
- Hardware Resources: Three personal laptops (two MacBook Airs, one MacBook Pro) used for development and testing.

5. *Use of Commercial Cloud Platforms*

- Google Cloud Platform (GCP): Utilized for Speech-to-Text and Vision OCR services to enable high-accuracy transcription and slide text extraction. Services operate within free-tier quotas.
- Amazon Web Services (AWS): EC2 for backend deployment, S3 for file storage, RDS for database hosting, and Route 53 for domain management. Free-tier services are used for prototype deployment and demonstration.

B. Software in Use

Several existing software solutions and research studies were referenced in designing the system:

- PromptSmart (VoiceTrack): A commercial teleprompter offering real-time voice tracking. The proposed system extends its capabilities by adding semantic matching, omission detection, and automated slide control.
- Microsoft PowerPoint (Live Subtitles): Provides speech transcription but lacks contextual synchronization with scripts and automated slide transitions.

These benchmarks highlight the project's improvements in real-time adaptivity and AI-driven presentation assistance.

C. Task Distribution

Role	Members	Responsibilities
Backend Development	Sangyoon Kwon, Dohoon Kim	System architecture design, FastAPI server and WebSocket implementation, database schema (PostgreSQL), AI logic integration (STT, LLM, BERT), cloud deployment (AWS, Render)
Frontend Development	Hyeyun Kwon, Seohyun Kim	React-based UI implementation, client-side state management, real-time dashboard (Web Audio API), teleprompter interface, client-side AI (TensorFlow.js)
Project Management & UI Design	Daeun Lee, Minhyuk Jang	Project planning and scheduling, UI/UX design (Figma), documentation and VCS management, user testing and feedback analysis

V. Specification

1) Requirement 1. Real-Time Teleprompter

This process involves a tightly coordinated real-time loop between the client (web browser) and the server (Python backend) through WebSocket communication.

1. **Client Initialization**
   When the user presses "Start Presentation," the React frontend requests microphone access using navigator.mediaDevices.getUserMedia() and establishes a secure WebSocket (wss://) connection to the backend API server.
   The Web Audio API initializes an AudioContext and ScriptProcessorNode to capture raw audio chunks.
2. **Client-Side Real-Time Audio Streaming**
   The ScriptProcessorNode continuously triggers onaudioprocess events (e.g., every 500 ms).
   Each raw audio buffer (16-bit PCM) is sent to the backend server through WebSocket.
3. **Server-Side STT and Synchronization**
   The backend receives the audio chunks and streams them to the Google Cloud Speech-to-Text API.
   The API returns *interim* (fast but less accurate) and *final* (slower but more accurate) transcripts.
   When a final sentence is received, it is appended to the full transcript of the session.
   The backend then calls the FlexibleSpeechMatcher service to locate the new currentSentenceIndex within the user's script.
4. **Server Broadcast**
   The server immediately sends a WebSocket message to the client:
   { "action": "UPDATE_TELEPROMPTER", "index": currentSentenceIndex }

5. **Client Update**
   The frontend WebSocket listener receives the message and updates the highlighted text accordingly, scrolling the teleprompter to the current index in real time.

2) Requirement 2. Flexible Speech-to-Script Matching

This algorithm implements a hybrid matching mechanism combining fast vector similarity and fallback large-language-model (LLM) validation.

Pseudocode Overview

```
SIMILARITY_THRESHOLD = 0.75
SEARCH_WINDOW = 5
LLM_VALIDATION_THRESHOLD = 0.60

Function FindCurrentPosition(fullTranscript,
scriptSentences, lastIndex):
   latestTranscript = GetLastNWords(fullTranscript, 10)
   transcriptVector =
KoSentenceBERT.encode(latestTranscript)

   searchStart = lastIndex
   searchEnd = min(lastIndex + SEARCH_WINDOW,
len(scriptSentences))
   searchWindow = scriptSentences[searchStart :
searchEnd]

   bestMatchIndex = -1
   highestSimilarity = 0

   # Step 1: Fast vector similarity
   for index, sentence in enumerate(searchWindow):
      similarity = CosineSimilarity(transcriptVector,
sentence.vector)
      if similarity > highestSimilarity:
         highestSimilarity = similarity
         bestMatchIndex = searchStart + index

   # Step 2: Omission check
   if bestMatchIndex > lastIndex:
      CheckForOmissions(lastIndex, bestMatchIndex,
scriptSentences)

   if highestSimilarity >= SIMILARITY_THRESHOLD:
      return bestMatchIndex

   # Step 3: LLM fallback validation
   if highestSimilarity >=
LLM_VALIDATION_THRESHOLD:
      prompt = CreateLLMPrompt(latestTranscript,
searchWindow)
      AsyncCallLLM(prompt, HandleLLMResult)
      return bestMatchIndex



   return lastIndex
```

This process ensures low latency while maintaining semantic accuracy through adaptive matching.

3) Requirement 3. Key Content Omission Detection

This function integrates directly with the flexible matching process.
When a skipped section is detected, the system checks whether any omitted sentence contains a predefined *key phrase* and alerts the presenter.

1. **Database Preparation**
   When users upload a script, each sentence is marked with a boolean attribute isKeyPhrase (true / false) and stored in the database.
2. **Server-Side Omission Detection**
   When FindCurrentPosition() identifies a new bestMatchIndex greater than lastIndex + 1, the server calls CheckForOmissions().
3. **Omission Logic**
   If skipped sentences are found between the two indices, each is inspected for isKeyPhrase == true.
   When detected, the omitted sentence index is flagged, and the following message is broadcast:
   { "action": "OMISSION_DETECTED", "index": omittedSentenceIndex }
4. **Client Notification**
   The frontend highlights the corresponding part of the script (e.g., flashing red or adding a border) to visually warn the presenter in real time.
   Simultaneously, an asynchronous script-reconstruction task (Requirement 4) is triggered.

4) Requirement 4. Real-Time Script Reconstruction

This asynchronous process generates short “bridging sentences” whenever key content omissions are detected, ensuring smooth narrative flow without latency in the main loop.

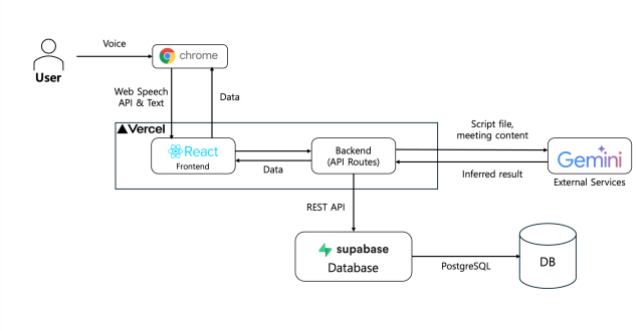
1. **Asynchronous Trigger**
   CheckForOmissions() launches HandleOmissionAsynchronously() in a separate asynchronous task.
2. **Prompt Generation for LLM**
   The function combines three inputs:
   (a) the omitted sentence, (b) the current context sentences, and (c) an instruction prompt such as:
   “You are a presentation coach. The presenter accidentally omitted '[omittedSentence]' and is now moving to '[contextSentences]'. Please generate one short, natural bridging sentence in Korean that connects these topics smoothly.”
3. **LLM API Invocation**
   The backend asynchronously requests an LLM (e.g., GPT or Gemini) to generate the bridging sentence.
4. **Response Delivery**
   Upon success, the server sends the message:

{ "action": "SCRIPT_SUGGESTION", "text": llm_generated_sentence }

5. **Client Interface Behavior**
   The React frontend displays the generated sentence in the AI Suggestion section as an alert message: "Do you approve this suggestion?" with **Accept (#0064FF)** and **No (#E0E6EA)** buttons.
   If the user selects *Accept*, the updated script is applied, and a small alert "💡 Update complete" appears at the top-right corner.

IV. ARICHITECTURE DESIGN

A. Overall Architecture



FoS consists of three main modules: the **Frontend (React web application)**, the **Backend (Main Server)**, and the **Data & AI Service layer (Supabase–PostgreSQL and Gemini)**. Together, these components form an end-to-end pipeline that captures the user's speech in Chrome, processes it, and reflects the results on the presentation and meeting interfaces in real time.

The first module, the **Frontend**, is implemented as a React + TypeScript web application running in the Chrome browser. The user speaks through the browser, and the Web Speech API converts the voice input into text. Using this text and the data received from the backend, the frontend renders the real-time teleprompter view, the meeting agenda map, and related dashboards. It communicates with the backend via WebSocket and REST APIs, sending user input and receiving updates such as AI analysis results, alerts, and script suggestions, which are then dynamically reflected in the user interface.

The second module, the **Backend (Main Server)**, is built on **Node.js–based API Routes** and serves as the core processing engine of the system. The backend aggregates utterance text and UI state from the frontend, manages them at the session level, and interprets the overall flow of presentations and meetings. It also integrates with external AI services such as Gemini and other STT/summary APIs by sending script files and meeting content and receiving various inferred results, including summaries, classifications, and suggested sentences. These results are returned to the frontend for visualization and are persisted in the database for later review and analysis. Thanks to Node.js's non-blocking, event-driven architecture, the backend can handle real-time text streaming and notification delivery with minimal latency.

The third module is the **Data & AI Service layer**. Through **Supabase**, the backend accesses a **PostgreSQL database** via REST APIs to store and retrieve user profiles, uploaded scripts, sentence-level metadata, meeting logs, and AI suggestion history. This layer ensures consistent state across sessions and provides the data foundation for both real-time operation and post-hoc analysis. On the AI side, **Gemini** functions as an external AI service positioned on the right side of the architecture diagram. The backend sends script and meeting content to Gemini and receives inferred results, which are then propagated to the frontend and stored in the database, completing the overall FoS architecture.

B. Directory Organization

Directory	File Name
fos/frontend/src	App.tsx, Attributions.md, index.css, main.tsx, vite-env.d.ts
fos/frontend/src/assets	favicon.png fos_logo.png google_logo.png kakao_logo.png
fos/frontend/src/components	AgendaTag.tsx Logo.tsx StatusPill.tsx TopNavBar.tsx
Fos/frontend/src/components/ui	button.tsx checkoutbox.tsx input.tsx label.tsx textarea.tsx utils.tsx
fos/frontend/src/lib	supabaseClient.tsx
fos/frontend/src/screens	AgendaTrackerScreen.tsx EndPresentationModal.tsx LoginScreen.tsx MainScreen.tsx MeetingSummaryScreen.tsx PresentationSetupScreen.tsx TeleprompterScreen.tsx
fos/frontend/src/styles	globals.css
fos/docs	SE_Assignment2_G12.pdf architecture_diagram.png fos_paper.pdf fos_paper.tex paperscreenshot.png structure.md webscreenshot.png
fos/api	extract-keywords.ts llm-regenerate.ts llm-test.ts speech-comparison.ts
fos/api/utils	comparison-service.ts llm-client.ts types.ts
fos/.vercel	README.txt

	Project.json

C. Module 1: Frontend

1) Purpose

The Frontend Module of FoS serves as the **client-facing interface**, allowing users to interact with the system through the web browser. It provides a seamless flow across core features such as presentation setup, real-time teleprompter, agenda tracking, and meeting summaries. The module visualizes analysis results from the server and AI modules, handles web speech input and user interactions, and aims to deliver a smooth and focused experience for presenters and meeting participants.

2) Functionality

The frontend module manages multiple screens—including login, main dashboard, presentation setup, teleprompter, agenda map, and meeting summary—using routing to support the entire presentation and meeting flow. It forwards speech recognized by the browser's Web Speech API to the backend and, based on the returned script-matching, keyword, and agenda information, renders highlights, progress indicators, and agenda nodes in real time. It also integrates Supabase-based social login to maintain user sessions and leverages reusable components such as buttons, navigation bars, and status pills to build a consistent interface that communicates smoothly with other modules.

3) Location of source code

https://github.com/april2901/fos/tree/main/frontend

4) Class component

(1) FoS/frontend/src: This is the root directory that contains the application entry point and global configuration files.

- **App.tsx**: The main component of the application; defines routing and the overall layout.
- **Attributions.md**: A document that records copyright and attribution information for external libraries and resources used in the project.
- **index.css**: The file that defines the base CSS styles for the application.
- **main.tsx**: The entry point of the React application; renders the root component into the DOM.
- **vite-env.d.ts**: A file that defines TypeScript environment types for the Vite build tool.

(2) **FoS/frontend/src/assets:** This folder contains images and other static resource files used in the application.

- **favicon.png**: The favicon image displayed in the browser tab.
- **FOS_Logo.png**: The main logo image of the FoS application.
- **google_Logo.png**: The logo image used for the Google social login button.
- **kakao_Logo.png**: The logo image used for the Kakao social login button.

(3) FoS/frontend/src/components: This folder contains reusable UI components.

- **AgendaTag.tsx**: A component that displays presentation agenda items as tags.
- **Logo.tsx**: A component that renders the FoS logo.
- **StatusPill.tsx**: A component that displays status information as a pill-shaped badge.
- **TopNavBar.tsx**: A component that renders the top navigation bar.

(4) **FoS/frontend/src/components/ui:** This folder contains base UI components built on top of shadcn/ui.

- **button.tsx**: A reusable button component.
- **checkbox.tsx**: A checkbox input component.
- **input.tsx**: A text input field component.
- **label.tsx**: A label component for form elements.
- **textarea.tsx**: A multi-line text input component.
- **utils.ts**: A file that defines utility functions used by UI components.

(5) **FoS/frontend/src/lib:** This folder contains configuration files for integrating external services and libraries.

- **supabaseClient.ts**: A file that creates and configures the Supabase client instance.

(6) FoS/frontend/src/screens: This folder contains screen components that make up each page of the application.

- **AgendaTrackerScreen.tsx**: The screen that tracks and displays agenda progress during a presentation.
- **EndPresentationModal.tsx**: A modal component displayed when a presentation ends.
- **LoginScreen.tsx**: The screen that handles user login and social authentication.
- **MainScreen.tsx**: The main dashboard screen shown after login.
- **MeetingSummaryScreen.tsx**: The screen that displays a summary of the meeting after the presentation ends.
- **PresentationSetupScreen.tsx**: The screen used to configure settings before starting a presentation.
- **TeleprompterScreen.tsx**: The screen that provides teleprompter functionality during a presentation.

(7) FoS/frontend/src/styles: This folder contains global style and theme-related CSS files.

- **globals.css**: The file that defines Tailwind CSS configuration and global styles.

D. Module 2: Backend & AI

1) Purpose

The Backend & AI Module acts as the **core processing engine** of FoS, handling requests from the frontend and executing business logic such as LLM calls, script–speech comparison, keyword extraction, and skipped-section reconstruction. Its purpose is to transform raw spoken and scripted content into structured insights, enabling intelligent features that help users stay focused on their speech during presentations and meetings.

2) Functionality

This module exposes API endpoints under /api for keyword extraction, script reconstruction, LLM connectivity testing, and script–speech comparison. Using comparison-service.ts, it performs text normalization, partial-range search, and skipped-range detection to compute the current matched index and skipped segments, providing the data needed for teleprompter highlighting and error visualization. Through llm-client.ts, it encapsulates calls to LLMs such as Gemini with shared configuration, returning keyword lists, meeting-analysis JSON objects, and reconstructed sentences in a consistent format. The types.ts file defines shared data structures used across the module, ensuring stable and type-safe communication between APIs and other parts of the system.

3) Location of source code

https://github.com/april2901/fos/tree/main/api

4) Class component

(1) FoS/api: this directory contains backend API route handlers that expose core LLM and speech-matching features to the frontend

- **extract-keywords.ts** API endpoint that receives a presentation script, calls the Gemini model, and returns 3–5 key keywords extracted from the script.
- **llm-regenerate.ts**: API used to reconstruct skipped parts of a presentation script by sending the skipped content and its surrounding context to the LLM and generating a natural bridging sentence.
- **llm-test.ts**: Test endpoint for checking LLM connectivity and prompt/response behavior, used to verify that model calls work correctly.
- **speech-comparison.ts**: API that compares recognized speech text with the full script, computes the current matched index, and returns matching confidence and skipped ranges. It powers the teleprompter's highlight position and skipped-segment visualization.

(2) FoS/api/utils: This directory contains shared business logic, utility modules, and type definitions used by the API routes

- **comparison-service.ts**: Module that implements the core logic for comparing script text and spoken text, including text normalization, partial range search, and skipped-range detection used by speech-comparison.ts.
- **llm-client.ts**: Client wrapper for calling LLM APIs (e.g., Gemini), encapsulating common configuration such as model name, temperature, token limits, and request/response handling.
- **types.ts**: File that defines TypeScript interfaces and types for speech comparison results, keyword extraction results, LLM responses, and other shared data structures.

E. Module 3: Configuration

1) Purpose

The Configuration Module defines how FoS is **recognized and deployed** within the Vercel environment. It manages project metadata, linked repositories, and environment settings so that the frontend and backend modules can run consistently and reliably in the cloud.

2) Functionality

The .vercel directory stores metadata such as the Vercel project ID, linked GitHub repository, and deployment environments (e.g., production and preview), controlling the deployment pipeline and domain configuration. Project.json defines these settings in JSON form, enabling branch-based automatic builds and deployments, while README.txt documents how the directory and deployment setup work so that team members can share a common understanding of the configuration and release process.

3) Location of source code

https://github.com/april2901/fos/tree/main/.vercel

4) Class component

(1) FoS/.vercel: This directory contains Vercel deployment metadata and configuration and configuration files that control how the project is recognized and deployed in the Vercel environment.

- **README.txt:** Text document that describes how the Vercel project is configured and deployed and how the .Vercel directory is used.
- **Project.json**: JSON file that stores Vercel project metadata and settings, including items such as the project ID, linked Git repository, and environment-related deployment configuration

V. INITIAL FUNCTIONAL TEST CASES

The system consists of 6 core screens, each representing a key stage in the user's presentation and meeting workflow. The overall flow begins with login, proceeds through preparation and live execution, and ends with meeting summarization.

A. Screen List

Screen 1. Login
Users enter their email and password to access the system.

Screen 2. Main Dashboard
Provides two primary entry points:
"Start Meeting" → navigates directly to Screen 5 (Agenda Map)

"Prepare for Presentation" → navigates to Screen 3 (Presentation Preparation Screen)

Screen 3. Presentation Preparation Screen
Users upload presentation slides, enter scripts, and verify estimated presentation time.

Screen 4. Smart Presentation Teleprompter (Presentation Mode)
Real-time STT matches the speaker's voice to the script, highlights spoken parts, and auto-advances slides.

Screen 4-2. Presentation End Modal (Switch to Meeting Mode)
After the presentation ends, users can either return to the main screen or directly begin a meeting with Agenda Map pre-generated from the presentation content.

Screen 5. Agenda Map (Real-Time Meeting Map)
Real-time STT analyzes discussions, categorizes statements (Idea, Decision, Action Item, etc.), and generates a branching visual map.

Screen 6. Meeting Summary Report
Displays the final agenda map, decisions, action items, and a chronological timeline of the meeting.

B. Initial Functional Test Case Table

Use Case	**Function Being Tested**	**Initial System State**	**Input**	**Expected Output**
User Login	System authenticates user and opens main dashboard	User is on Login Screen, not authenticated	User enters valid email & password, clicks a login button	User is authenticated and **Screen 2. Main** is displayed
User Login	System rejects invalid credentials	User is on Login Screen, not authenticated	User enters invalid email or password, clicks a login button	Login fails, error message is shown, user stays on Login Screen
Navigate to Presentation Prep	System routes to Screen 3	User authenticated, on Screen 2	Click "Prepare for Presentation"	Navigate to Screen 3
Start Meeting Immediately	System routes to Screen 5	User authenticated, on Screen 2	Click "Start Meeting"	Navigate to Screen 5 with empty Agenda Map
Enter Presentation Title	System accepts title input	Screen 3 loaded	Enter text in title field	Title saved in preparation state
Enter Script	Script is captured & character count calculated	Script field empty	Type script content	Estimated time updates in real-time
Upload Slides	System accepts ppt/pdf file	No file uploaded	Upload a valid ppt/pdf	Slides preview shown; auto-matching enabled
Auto Script-Slide Matching	System matches script segments to slide pages	Script + slides both provided	Click "Auto-Match"	Slide-script mapping is generated
Begin Presentation	System navigates to Screen 4	Inputs complete	Click "Start Presentation"	Navigate to Screen 4
Start Teleprompter	STT begins and script highlighting activates	Screen 4 loaded, STT OFF	Click "Start"	STT activated, real-time highlighting begins
Pause Teleprompter	STT pauses	Teleprompter running	Click "Pause"	Highlight freezes; STT stops listening
Auto Slide Advance	Slide moves based on script progression	STT running	Speak script matching slide threshold	Slide auto-advance triggered
Volume & Speed Monitoring	Dashboard displays metrics	Teleprompter active	Speak out loud	Volume level & speaking speed update dynamically
End Presentation	Presentation ends and modal opens	Teleprompter active	Click "End Presentation"	Screen 4-2 modal appears
Start Follow-Up Meeting	Modal transitions to Screen 5	Modal displayed	Click "Start Follow-Up Meeting"	Navigate to Screen 5 with auto-generated initial Agenda Map from presentation
Return to Main	Back to Screen 2	Modal displayed	Click "Finish"	Navigate to Screen 2
Real-Time STT Categorization	System tags utterances (Idea, Decision, etc.)	STT ON, meeting ongoing	User speaks	Node created with correct tag
Add Custom Node	System adds node from	Agenda map displayed	Enter text + press "+"	New node appears on map

	manual input			
Node Dragging	Users can repositio n nodes	Map displayed	Drag a node	Node moves to new position
Canvas Panning	Move entire map	Click empty canvas & drag	Map pans accordin gly	
View Node Details	Display full STT transcrip t + timestam p	Node exists	Click on a node	Tooltip/overla y near node displays transcript
Manage Decisions & Action Items	Cards editable	Cards appear in right panel	Click card	Options to edit/delete/reo rder appear
End Meeting	System generate s summary	Meeting active	Click “End Meeting”	Navigate to Screen 6
View Final Agenda Map	System displays saved map	Screen 6 loaded	None	Final map shown, draggable/zoo mable
Timeline Interactio n	Highligh t related nodes	Timeline displayed	Click topic on timeline	Matching nodes are highlighted
Complete Process	System returns to main	Summary displayed	Click “Return to Main”	Navigate back to Screen 2

VI. Discussion

When the project first began, our team focused exclusively on developing features related to presentations. However, midway through the semester, the possibility of integrating our system with LG’s smart office ecosystem arose, requiring us to introduce meeting-support functionalities and significantly shift our overall project direction. This transition forced us to rethink how the presentation and meeting stages could operate not as separate modules but as a single coherent workflow. Determining how data should transition smoothly from real-time teleprompter mode to meeting analysis—while still preserving the features originally designed for presentations—was one of the most complex challenges we faced. We spent substantial time evaluating architectural options, revising user flow designs, and ensuring that newly added meeting components did not conflict with or overly complicate the existing presentation pipeline.

Another major difficulty stemmed from subtle but important differences in how each team member envisioned the system. Although the high-level concept seemed aligned, our expectations regarding UI behavior, API boundaries, and real-time synchronization often diverged. As a result, every team meeting required deliberate coordination to clarify assumptions, unify interpretations, and adjust the design accordingly. This iterative process consumed more time than expected, but it ultimately led to a more consistent system architecture and strengthened our ability to collaborate effectively.

Overall, the project challenged us to adapt to evolving requirements, redesign the architecture under real-world constraints, and continuously refine our communication strategies. These experiences offered meaningful insight into practical software development, where feature expansion, integration complexity, and team alignment are recurring and essential aspects of the engineering process.

VII. Appendix

A. Open-Source Licences and Attributions

The FoS project uses Figma Make UI compoments and styles under the MIT License notices and attributions for Figma Make and other third-party

- source/src/Attributions.md
- source/src/index.css