Real-time Presentation Assistance System

Group 12

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, presentations serve as a key means of communication, and their importance has become increasingly emphasized. However, many presenters experience cognitive overload as they simultaneously manage speech delivery, script reference, slide transitions, and time management. To address this issue, this project proposes a real-time teleprompter and automatic slide transition system based on speech recognition and script synchronization. The system incorporates core features such as flexible speech-to-script matching, keyword omission detection, and a real-time feedback dashboard, while also providing pre- and post-presentation support including script consulting, Q&A assistance, and presentation analysis reports. Through these functions, the project aims to enhance presentation quality and delivery effectiveness, and further explore the potential for integration within LG's smart office ecosystem.**

Keywords—Speech Recognition, Script Synchronization, Real-Time Teleprompter, Slide Automation, Presentation Feedback, Human-Computer Interaction

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

In both professional and academic contexts, presentations have evolved into essential tools for conveying ideas and fostering collaboration. However, many presenters experience cognitive overload as they simultaneously manage speech delivery, script reference, slide transitions, and time tracking. This burden often disrupts the flow of presentations or leads to the omission of key content, which in turn reduces presentation quality and negatively affects organizational communication efficiency. There is a growing need for a context-aware system that not only recognizes the presenter's speech in real time but also supports slide transitions, script tracking, and adaptive feedback to reduce cognitive load and improve delivery consistency.

B. **Limitations of Existing Solutions**

Existing teleprompters or timer applications are limited to providing static information and fail to actively respond to real-time situations. Furthermore, practical software that presenters can effectively utilize during preparation and live presentations remains insufficient. While some related services and research efforts, such as PromptSmart's VoiceTrack function, Microsoft PowerPoint's real-time translated subtitles, and the SlideSpeech Corpus, attempt to address these issues, they primarily focus on displaying or generating information rather than providing real-time assistance to presenters. They do not fundamentally resolve the presenter's cognitive overload. Such limitations are particularly evident in business and educational contexts where effective delivery and organizational communication are critical.

C. Project Goals and Proposed Solution

To address these issues, this project proposes an integrated support system that covers the entire presentation process, from preparation to live delivery and post-presentation feedback. Unlike existing tools that merely offer static display or transcription functions, the proposed system integrates speech recognition, contextual understanding, and adaptive feedback mechanisms to provide comprehensive, real-time assistance throughout the presentation process. During presentations, the system helps with multitasking such as script reference, slide management, and time monitoring, which often diminishes the presenter's performance. In the preparation phase, it addresses the challenge of logically converting presentation materials into coherent scripts, which can be difficult with current large language models. After presentations, the system provides a more efficient feedback process, making it easier for presenters to identify and apply improvements. When combined with LG Electronics' webOS platform and voice-based AI services, the proposed system can offer enterprise-specific features such as seamless device connectivity, integrated voice control, and real-time data sharing across smart office devices. By linking with LG displays and projectors, it can automatically synchronize presentation visuals, monitor device usage patterns, and provide analytics on meeting efficiency, thereby contributing to the advancement of LG's Smart Office ecosystem.

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***

The system transcribes the presenter's 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.

- 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

- Constraints

• Presentation ≤ 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

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 Analysis Report

After the presentation, the system automatically analyzes collected data (speech logs, timing, emotion) and generates a report covering time management, speech habits, and content delivery.

- Acceptance Criteria

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

- Input & Output

• **Input:** Speech logs, slide transitions, emotion data
• **Output:** Analysis report (.html,.pdf)

- Constraints

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

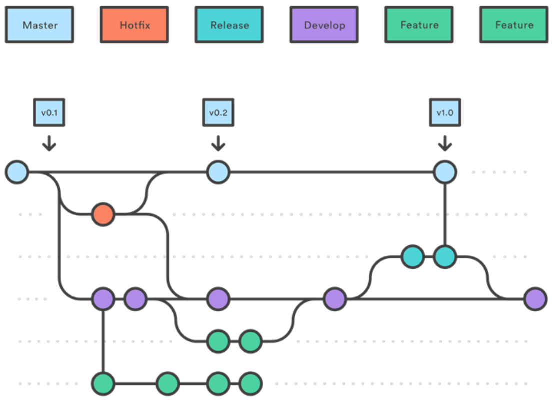
III. Version Control System

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

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: Python (version 3.11 or higher) using FastAPI for asynchronous API and WebSocket communication. Python is the de facto standard for

AI and machine learning workflows, offering a mature ecosystem that includes python-pptx, sentence-transformers, and google-cloud-speech libraries.

- 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.