Faculty of Natural and Mathematical Sciences
Department of Informatics

King's College London
Strand Campus, London,
United Kingdom

7CCSMPRJ

Individual Project Submission 2024/25

Name: Dwitee Krishna Panda

Student Number: K24112349

Degree Programme: Advanced Software Engineering

Project Title: Making Audio and Video Media More Accessible

Supervisor: Dr. Timothy Neate

Word Count: 11880

RELEASE OF PROJECT

Following the submission of your project, the Department would like to make it publicly available via the library electronic resources. You will retain copyright of the project.

☑ I **agree** to the release of my project

☐ I **do not** agree to the release of my project

Signature: Dwitee Krishna Panda **Date:** August 7, 2025

Department of Informatics
King's College London
United Kingdom

7CCSMPRJ Individual Project

Making Audio and Video Media More Accessible

Name: **Dwitee Krishna Panda**
Student Number: K24112349
Course: Advanced Software Engineering

Supervisor: Dr. Timothy Neate

This dissertation is submitted for the degree of MSc in Advanced Software Engineering.

Abstract

This project addresses the growing demand for accessible media content tailored specifically to individuals with complex communication needs, particularly those affected by aphasia and cognitive impairments. Traditional video accessibility methods—such as subtitles and full transcripts—often fall short due to their heavy reliance on sustained reading and text-processing skills, posing significant comprehension challenges for this user group. To overcome these limitations, this work proposes an AI-driven pipeline that enhances understanding through concise, chapter-based textual summaries combined with interactive visual mind maps.

The project centers around an end-to-end system capable of processing both user-uploaded videos and live recordings. The processing pipeline integrates multiple Natural Language Processing (NLP) and Machine Learning (ML) techniques, including automatic speech recognition (ASR), semantic summarization, and visual representation. Specifically, the system employs the Whisper model for highly accurate speech-to-text transcription, subsequently leveraging advanced transformer-based summarization models—particularly T5-small, BART, and Gemini. Models such as BART provide robust abstractive summarization capabilities, effectively condensing lengthy video content into clear, topic-segmented chapters, while Gemini enhances summary coherence, contextual relevance, and reduces linguistic complexity. Additionally, lightweight transformer models (e.g., T5-small, Zephyr), optimized using quantization frameworks such as llama.cpp, ensure computational efficiency and enable real-time performance.

A web application, built using Next.js (React) on the frontend and supported by Node js, provides an intuitive interface explicitly designed for user with complex communication needs. The backend infrastructure consists of a Python Flask REST API deployed as a Web Server Gateway Interface (WSGI) service, hosted on a GPU-enabled Google Cloud Virtual Machine, running efficient, scalable, and cost-effective model inference.

The principal contributions of this project include developing a practical, scalable solution for accessible media, substantially improving comprehension for users with Complex Communication Needs, through multi-modal summarization and visual representation techniques. Key findings from user evaluations indicate that the combination of visually structured mind maps and simplified textual summaries significantly reduces cognitive load and enhances comprehension compared to traditional subtitle-based methods. Developed collaboratively with the CA11y initiative [15] and BBC R&D, this project provides a robust prototype, demonstrating the potential of AI-powered assistive technologies to be integrated effectively into future inclusive media platforms and accessibility frameworks.

Nomenclature

AI	Artificial Intelligence
ASR	Automatic Speech Recognition
ASD	Autism Spectrum Disorder
BERT	Bidirectional Encoder Representations from Transformers
BART	Bidirectional Auto-Regressive Transformer
CA11y	Communication Accessibility Initiative
GPU	Graphics Processing Unit
ML	Machine Learning
NLP	Natural Language Processing
REST	Representational State Transfer
T5	Text-to-Text Transfer Transformer
$X = (x_1, x_2, \ldots, x_n)$	Sequence of observed tokens
$Y = (y_1, y_2, \ldots, y_n)$	Sequence of hidden states
$P(X, Y)$	Joint probability of observations and hidden states
$P(y_t \mid y_{t-1})$	Transition probability from state y_{t-1} to y_t
$P(x_t \mid y_t)$	Emission probability of observation x_t given hidden state y_t
$P(y_1)$	Initial state probability
Q	Query matrix
K	Key matrix
V	Value matrix
d_k	Dimensionality of the key vectors
$\mathrm{softmax}(\cdot)$	Normalization function that converts scores to probabilities
WSGI	Web Server Gateway Interface
CI/CD	continuous integration and continuous delivery/deployment

Contents

1 Introduction **1**
1.1 Scene Setting . . . 1
1.2 Objectives . . . 1
1.3 What to expect from this report . . . 2
1.4 Dataset . . . 3

2 Background **4**
2.1 Natural Language Processing (NLP) . . . 4
2.2 Speech recognition. . . . 5
2.3 Transformer Architecture and its Significance . . . 5
2.3.0.1 Significance in ReMindMap: . . . 7
2.4 T5: Text-to-Text Transfer Transformer . . . 7
2.5 Bidirectional Encoder Representations from Transformers (BERT) . . . 7

3 Implementation Details **9**
3.1 Infrastructure and Deployment . . . 9
3.2 High-Level Architecture . . . 9
3.3 System Design . . . 10
3.4 Model Selection and Optimization . . . 11
3.5 Cost Efficiency through Lightweight Models . . . 12
3.6 Non-Functional Requirements . . . 12
3.7 Summary of System Design and Implementation . . . 13
3.8 Low Level Design for Backend Implementation . . . 14
3.8.1 API: /upload-thumb . . . 14
3.8.2 API: /upload-video . . . 14
3.8.3 API: /submit-video-to-summarize . . . 15
3.8.4 Module: job_processor.py . . . 16
3.8.5 ASR Module: Whisper-based Transcription . . . 16
3.8.6 Caching Mechanism for Transcript, Summary, and Metadata . . . 17
3.8.7 Summarization using T5-small Model . . . 18
3.8.8 Summarization using Gemini Model . . . 18
3.8.9 Mind Map Generation with Gemini and Mistral Models . . . 19
3.8.10 API: /save-summary . . . 20
3.8.11 API: /list-summaries . . . 20
3.8.12 API: /download-youtube-and-submit . . . 21
3.9 Low Level Design for Frontend Implementation . . . 22
3.9.1 Frontend Design and Framework Decisions . . . 22
3.9.2 CI/CD pipeline . . . 23
3.9.3 Responsiveness and Device Compatibility . . . 23
3.9.4 Frontend Libraries and Their Purposes . . . 25
3.10 Results, Analysis and Evaluation . . . 26
3.10.1 Overview of System Outputs . . . 26

3.11 Quntitative Runtime Analysis 31
3.12 Execution Time: Transcribing vs. Summarization 32
3.13 Inference Performance: T5 vs Gemini vs Zephyr 32
3.14 System Utilization Metrics from Google Cloud 33
3.15 Local Run Model Vs API based Model 34
3.16 Pros and Cons: GPU vs. CPU 35

4 Conclusion **36**
4.1 Future Work and Scalability 37

References **39**

A Appendix **41**

Appendix **41**
A.1 Installing GPU-accelerated version of PyTorch and dependencies 41
A.2 /upload-thumb API 41
A.2.1 Expected Request 41
A.2.2 Success Response 41
A.2.3 Error Response 42
A.3 /upload-video API 42
A.3.1 Expected Request 42
A.3.2 Success Response 42
A.3.3 Error Response 42
A.4 /submit-video-to-summarize calls the following function from jobprocessor 43
A.4.1 /submit-video-to-summarize API 43
A.4.2 Expected Request 44
A.4.3 Success Response 44
A.4.4 Error Response 44
A.5 Job Processor 44
A.6 Whisper-based transcription 46
A.7 t5-small transformer model 47
A.8 Summarization using Gemini model 47
A.9 MindMap generator 48
A.10 Simple Json Output used to create mindmap 49
A.11 Sample Json for Chaptertized summary 50
A.12 video-summarizer-frontend/pages/index.tsx 55
A.13 video-summarizer-frontend/pages/video/[id].tsx 59
A.14 video-summarizer-frontend/lib/config.ts 74
A.15 video-summarizer-frontend/lib/summarizer.ts 74
A.16 video-summarizer-frontend/hook/useVideoSummarizer.ts 77
A.17 video-summarizer-frontend/package.json 82
A.18 video-summarizer-frontend/global.css 83

List of Figures

1 (left) Scaled Dot-Product Attention and (right) Multi-Head Attention consists of several attention layers running in parallel. [16] 6
2 Google Cloud VM config 9
3 System Design Diagram , high-res Image is here `https://github.com/Dwitee/video-summarizer-frontend/blob/main/system_design.png` . 10
4 T4 Tesla GPU overloaded with llama.cpp model 11
5 Frontend–Backend Interaction Sequence Diagram 22
6 Gradio based UI (old) 24
7 Next.js based UI (new) 24
8 User uploading or recording content through the ReMindMap interface . . 26
9 Mind map in standard (expanded) mode with all chapters visible 27
10 Mind map in radial (compact) mode with interactive expansion 28
11 Narration mode with emoji, focused node, and text display 29
12 Dual device interaction: User can scan the QR Code from his device to see and interact in his device. 30
13 Cross Platform as a web App and supporting different resolutions 31
14 Execution Time Comparison: Whisper vs Summarization 32
15 Performance Comparison of various models , in x-axis video duration in y-axis response time (all included). 33
16 Gemini 2.0 Flash Pricing by Google Vertex AI) 33
17 System Metrics During Summarization (CPU Load, Process Rate, Thread Count) 34
18 Dual device interaction: viewing video on a desktop while the same mind map is accessible on a iPad device 37
19 POC of Chrome plugin , check the yellow box 84
20 POC of Chrome plugin summary, check the yellow box 84
21 Deployment Pipeline using Vercel connected to github repo 85

List of Tables

1 Whisper models comparation [13] 5
2 Frontend Dependencies and Their Purposes 25

1 Introduction

1.1 Scene Setting

Effective communication is fundamental to human interaction, and the accessibility of media content is very crucial in ensuring inclusivity for diverse user groups. As the amount of digital media content grows exponentially over the past few years, ensuring that such media is comprehensible to people with varying cognitive and communication abilities has become increasingly important aspect [11] . However, individuals with complex communication needs, especially those affected by neurological conditions such as aphasia and ASD, often encounter significant challenges when interacting with regular forms of media [12]. Aphasia is a disorder resulting from damage—most commonly due to a stroke or traumatic brain injury—to the regions of the brain responsible for language. It impairs an individual's ability to express and understand language across different modalities, including speaking, reading, writing, and comprehension [10]. Traditional accessibility methods such as closed captions or full transcripts are often inadequate for users with aphasia [2]. These tools rely on sustained reading ability and the cognitive effort required to follow fast-paced or complex dialogues, which may overwhelm or exclude users with language-processing deficits [1] . In response to this pressing need, advancements in Artificial Intelligence (AI), Natural Language Processing, and Machine Learning have opened new doors for creating assistive technologies that go beyond trabditional text-based solutions. With the help of recent breakthroughs in multi-modal summarization and visual representation, it is now possible to re-imagine how video content is presented to users with cognitive impairments [3]. Advances in deep learning, multi-modal interaction, and real-time processing have revolutionized assistive technologies, enabling more inclusive and empowering user experiences [9]. This project explores one such solution, a comprehensive system that transforms spoken content in videos into simplified chapter-based summaries and corresponding visual mind maps to improve clarity and retention.

1.2 Objectives

The primary objective of this project is to develop an accessibility tool in the form of an web application that enhances the comprehension of video content for individuals with aphasia and related cognitive impairments. This tool aims to go beyond conventional subtitle-based solutions by offering multi-modal representations that simplify complex information and reduce cognitive load. It will rely on the latest advancement on NLP, transformer models and AI/ML concepts as well as applications. Also while keeping in mind the cost, practicability and future scope and extensibility of the application.

To achieve this, the project sets out the following specific objectives:

1. **Automated Transcription and Summarization** Implement a pipeline that automatically extracts speech from video content using advanced ASR models (e.g., Whisper) and generates concise, topic-based textual summaries using transformer-based models (e.g., `bart-large-cnn, t5-small, gemini-2.0-flash`).

2. **Visual Representation through Mind Maps** Design a system that transforms the textual summaries into interactive mind maps, representing the logical structure and key concepts of the video content visually using tools like Vis-network.
3. **Lightweight and Scalable Model Deployment** Optimize computational performance by integrating quantized and lightweight transformer models (e.g., Zephyr, llama.cpp), enabling deployment on resource-constrained environments (e.g., cloud VM with GPU).
4. **Simple and Intuitive Frontend Design** Develop a responsive, user-centric frontend interface using Next.js and React, guided by aphasia-friendly UX principles to ensure usability and clarity for neurodiverse users as per Accessible Information Guideline [6].
5. **Backend Integration and Cloud Hosting** Implement a backend system using Python Flask and deploy it as a WSGI service on Google Cloud VM, ensuring scalability and real-time inference capabilities.
6. **Evaluate Effectiveness and Usability** Conduct preliminary evaluation and qualitative feedback collection to assess the system's effectiveness in improving comprehension and reducing cognitive load for target users.

1.3 What to expect from this report

Henceforth, in this report this application developed during the Individual Project will be called as **ReMindMap** (Name is short form of "Revision of Content through interactive mindmap").

This report presents the development of ReMindMap, a summarization and mind mapping system designed to enhance accessibility for individuals with complex communication needs, such as those affected by aphasia. The system integrates state-of-the-art NLP and transformer-based models with interactive visualizations and synchronized narration to improve comprehension of audio-visual material through structured summarization and intuitive visual aids like simplified and interactive mindmap.

The report is structured to guide the reader through a detailed exploration of recent advances in Natural Language Processing (NLP), transformer architectures, and their transformative impact on assistive technologies. These foundational breakthroughs in semantic understanding, summarization, and entity extraction enable a wide range of applications across domains involving long-form textual, audio, and video content. The report underscores how the core concepts of NLP, AI, and ML are not only central to this research, but also serve as the enabling pillars for building intelligent, human-aligned systems.

To support rigorous understanding, a dedicated Background section outlines the mathematical and algorithmic principles behind Named Entity Recognition (NER), attention mechanisms, and sequence modeling. This includes detailed descriptions of models such as T5-small, Whisper, and BERT, all of which serve as foundational components of ReMindMap.

The report also addresses the critical need for cost-effectiveness, low latency, and practical scalability in real-world deployments. In the Technical Details section, it will be demonstrated how running open-source models locally on a virtual machine (VM) — with or without GPU acceleration — can eliminate reliance on expensive API-based solutions like commercial GPT models. This design choice not only makes the system more affordable and responsive but also offers greater control over data, compute resources, and deployment environments.

Subsequent sections, such as Related Work and Project Approach, will place this research in context by comparing similar systems and detailing how multiple models are orchestrated in a self-contained pipeline on a custom VM infrastructure. These discussions will emphasize the system's portability, extensibility, and suitability for assistive technologies in both academic and applied contexts.

Finally, it is important to note that ReMindMap has been developed as a production-ready application. Readers of this report will gain not only conceptual understanding but also practical guidance to replicate and customize the system. Leveraging freely available open-source models from Hugging Face and a lightweight backend architecture, the solution can be deployed with minimal cost. Both frontend and backend source code repositories will be shared in the Technical Details section, enabling others to fork, extend, and tailor the system to their specific needs.

1.4 Dataset

The input does not rely on a fixed dataset. Instead, it dynamically processes video input from two sources:

- **User-uploaded audio video files**
- **Live audio video recording**

The system supports all known audio video formats and makes use of real-time processing and summary and mindmap generation.

2 Background

This section outlines the theoretical and computational foundations that inform the design of ReMindMap. The ReMindMap project is situated at the intersection of natural language processing (NLP), automatic speech recognition (ASR), and neural network-based summarization. To construct a cost-effective, scalable, and locally deployable solution for summarizing audio-visual media, it is essential to understand the underlying models and algorithms upon which the system is built. t-5 , BERT and whisper are some of building blocks of ReMindMap application, whose core concepts has been explained in following section.

2.1 Natural Language Processing (NLP)

Natural Language Processing (NLP) forms the theoretical backbone of the ReMindMap system, enabling both transcription and summarization of video content. Foundational techniques such as Hidden Markov Models, Conditional Random Fields, and Support Vector Machines have historically been applied to key NLP tasks like part-of-speech tagging and named entity recognition [7]. The evolution from these traditional statistical models to modern transformer-based architectures—such as Whisper for speech recognition and T5 for summarization—marks a significant paradigm shift that enables more fluent, accurate, and context-aware language processing, as required in an accessibility-focused application like ReMindMap.

Key algorithmic techniques used in modern NLP include:

- **Tokenization**: Splitting text into discrete units (tokens), with subword techniques like Byte-Pair Encoding (BPE) and WordPiece.
- **Attention Mechanism**: Assigns weights to input tokens dynamically to capture contextual dependencies.
- **Self-Attention**: A form of attention where all tokens in a sequence attend to each other, enabling long-range contextual learning.
- **Encoder-Decoder Framework**: Used in tasks like translation and summarization, where input text is encoded into latent representations and decoded into output text. A latent representation is just a compressed and abstract form of the input text. This is how the model internally understands the real meaning of a sentence after encoding it. In context of summarization, Encoder-Decoder is quite important. The working can be understood in the following steps. First the encoder reads the input (transcribed text in our case) then transforms it into vector of numbers (the latent representation). This vector does not look like human language, but it captures sematics information (like the topic, intent, structure etc). Next the decoder takes latent vector and generate human readable output (chapterized summary in our case).

- **Analogy**: We can think latent representation as our brain's mental image. Lets someones says "beach vacation" . Then its is not stored literally, instead, our brain holds an abstract concept like sun, waves, sand, relaxation. With this we can describe it in different ways depending on the need.

2.2 Speech recognition.

For Automatic speech recognition Wishper is used . Whisper is a general-purpose speech recognition model. It is trained on a large dataset of diverse audio and is also a multi-tasking model that can perform multilingual speech recognition, speech translation, and language identification [13].

Table 1: Whisper models comparation [13]

Model	Layers	Width	Heads	Parameters
Tiny	4	384	6	39M
Base	6	512	8	74M
Small	12	768	12	244M
Medium	24	1024	16	769M
Large	32	1280	20	1550M

In the context of the ReMindMap system, selecting the appropriate Whisper model variant was critical for achieving a balance between transcription accuracy and system responsiveness. Since, ReMindMap is designed to run on a GPU-enabled VM(fall back to CPU when necessary), lightweight variants like Whisper-Tiny or Base were preferred. These smaller models provide sufficiently accurate transcriptions while maintaining fast inference speeds which is critical to reduce the processing time. Using a larger Whisper model would have improved accuracy marginally but at the cost of increased latency and memory usage, potentially degrading the accessibility experience for users.

2.3 Transformer Architecture and its Significance

The transformer model, revolutionized NLP by replacing sequential recurrence with a parallelizable attention-based mechanism [16]. It consists of an encoder-decoder architecture with multi-head self-attention, positional encoding, and feed-forward layers. Transformers enable rich contextual understanding and scale efficiently. Two widely adopted transformer-based models that form the backbone of ReMindMap are T5 and BERT.

The attention mechanism operates by mapping a query vector and a set of key-value pairs to an output vector. This output is formed as a weighted sum of the value vectors, where each weight reflects the relevance between the query and its corresponding key, typically computed using a compatibility function [16].

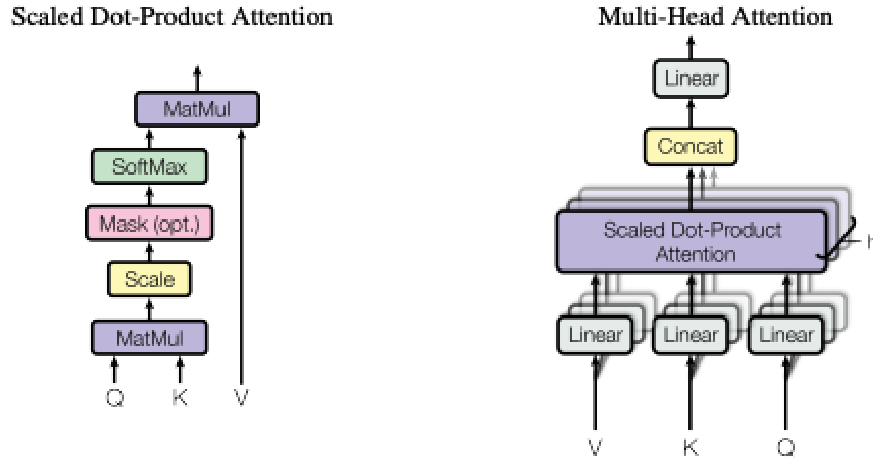

Figure 1: (left) Scaled Dot-Product Attention and (right) Multi-Head Attention consists of several attention layers running in parallel. [16]

Models like T5 and Gemini use transformer decoders to generate textual summaries from transcripts. These models are trained to maximize the conditional probability of a summary given a transcript, using attention to capture relevant parts of long documents. These formulas form the mathematical foundation of this capability.

Modern abstractive summarization models such as T5 and Gemini are based on the transformer architecture, which relies heavily on the self-attention mechanism to capture relationships between words in a sequence. The most critical component of this architecture is the *scaled dot-product attention*, defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{2.1}$$

Here, Q (query), K (key), and V (value) are linear projections of the input embeddings, and d_k is the dimensionality of the key vectors. This formulation allows the model to dynamically focus on different parts of the input when generating the output, making it highly effective for tasks such as summarization where salient content must be extracted from longer texts [16].

In Transformer models, the concept of **multi-headed attention** enhances the ability of the model to focus on different parts of the input simultaneously. Instead of computing a single attention output, the input queries(Q), keys(K), and values(V) are each projected into multiple lower-dimensional spaces using different learned weight matrices. This process is done h times, producing multiple attention ”heads”:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \tag{2.2}$$

where each attention head is calculated as:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.3}$$

Here, $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ are learned projection matrices for the i-th head, and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$ is the output projection matrix.

2.3.0.1 Significance in ReMindMap: In the ReMindMap system, models such as `T5` and `Gemini` rely on multi-head attention to summarize transcribed text content effectively. Each attention head can focus on different semantic aspects—such as tone, topic shifts, or speaker emphasis—across the input text. This is especially important in processing long and complex speech segments, allowing the summarization model to produce structured and coherent chapters or mind map elements. Without multi-head attention, a single attention vector would average out different contextual signals, leading to loss of important nuances—something critical to maintain when serving users with cognitive impairments.

2.4 T5: Text-to-Text Transfer Transformer

The Text-to-Text Transfer Transformer (T5) is a unified framework for natural language processing tasks that reformulates all problems into a text-to-text format. [14]. The T5 model is built upon the standard Transformer architecture and pre-trained on a large corpus called the Colossal Clean Crawled Corpus (C4) [5], consisting of cleaned web data. A key contribution of T5 is its emphasis on task uniformity. Inputs and outputs are always text strings, regardless of whether the task is summarization, translation, question answering, or classification. . To integrate the T5 model into the ReMindMap backend, the Hugging Face `transformers pipeline` interface was selected due to its simplicity, flexibility, and robust community support. The `pipeline` abstraction enables seamless loading of pre-trained models, automatic handling of tokenization and decoding, and provides a consistent interface for multiple NLP tasks—including summarization. This significantly reduces boilerplate code and accelerates development, especially in prototypes where rapid iteration is essential. Using the pipeline allowed the system to leverage the `t5-small` model with just a few lines of code, without manually configuring model architecture, tokenizer behavior, or inference pipelines. Please refer the model card of `t5-small` from hugginface `https://huggingface.co/google-t5/t5-small`

2.5 Bidirectional Encoder Representations from Transformers (BERT)

BERT (Bidirectional Encoder Representations from Transformers) [4] is a foundational model in modern NLP that introduced the concept of deeply bidirectional pretraining. It operates as a masked language model, learning contextual representations by predicting randomly masked tokens in the input sequence. Unlike encoder-decoder models like T5, BERT is based solely on the encoder architecture of the transformer and is typically fine-tuned with task-specific heads for various downstream applications. Its pretraining

objectives include Masked Language Modeling (MLM) and Next Sentence Prediction (NSP) [4], which together help the model capture both token-level and sentence-level relationships. BERT has been widely applied to tasks such as named entity recognition (NER), sentiment analysis, question answering, and sentence classification. Over time, several efficient and specialized variants have emerged, including BERT-base, BERT-large, bart-large-cnn, DistilBERT, RoBERTa, and ALBERT—each offering trade-offs between speed, accuracy, and model size.

In ReMindMap, `facebook/bart-large-cnn` has been tried using hugging face pipeline . BART is a transformer-based model that combines the strengths of both BERT and GPT. It uses a bidirectional encoder to understand the input text and an autoregressive decoder to generate output. During pretraining, BART learns by first adding noise to text and then training itself to fix or reconstruct it.

BART performs especially well in tasks like summarization and translation, but it's also effective in tasks like text classification and question answering. The version used in this project is trained on the CNN/DailyMail dataset, which contains many examples of news articles and their summaries [8].

3 Implementation Details

This section presents a detail walkthrough of the technical architecture, system implementation, and engineering decisions that shaped the development of **ReMindMap** which is also production-ready, video summarization and mind mapping system.

3.1 Infrastructure and Deployment

The backend runs on a Google Cloud VM, provisioned with optional NVIDIA T4 GPU. The backend uses:

- Flask + Gunicorn to serve the APIs.
- Google Cloud Storage for media file storage.
- Redis (managed or local) for lightweight, fast-access data storage.
- Vercel for hosting the Next.js frontend, with proxy rewrites to communicate with the Flask backend.

Environment variables such as `GCS_BUCKET_NAME` and `REDIS_URL` are managed through VM or `.env` for secure access. Please refer to the screenchot from google cloud Virtual Machine 2

Machine configuration

Machine type	n1-standard-2 (2 vCPUs, 7.5 GB Memory)
CPU platform	Intel Skylake
Minimum CPU platform	None
Architecture	x86/64
vCPUs to core ratio	—
Custom visible cores	—
All-core turbo-only mode	—
Display device	Disabled Enable to use screen capturing and recording tools
GPUs	1 x NVIDIA T4
Resource policies	

Figure 2: Google Cloud VM config

3.2 High-Level Architecture

The ReMindMap system is designed around a modular and scalable architecture that separates concerns across distinct subsystems:

- **Frontend**: A Next.js-based application that facilitates video upload and record, summary visualization, mind map interaction, and user-driven controls (e.g., narration, playback speed).
- **Backend**: A Flask application hosted on a Google Cloud VM. It handles model inference, video/audio transcription, summarization, mind map generation, caching, and persistence.
- **Storage**: Google Cloud Storage is used to store video and thumbnail assets, and Redis is employed for fast-access persistence of summary metadata.
- **AI/ML Models**: Multiple open-source models from Hugging Face and other providers are used to handle transcription, summarization, and mind map generation.

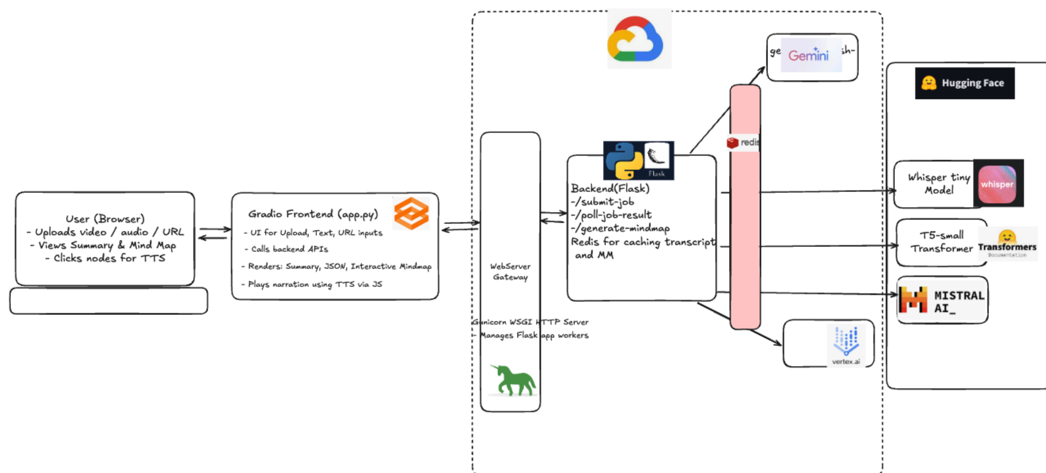

Figure 3: System Design Diagram , high-res Image is here `https://github.com/Dwitee/video-summarizer-frontend/blob/main/system_design.png`

3.3 System Design

The system is structured as a loosely coupled microservice pipeline, which supports the following stages:

1. **Video Input**: Users can upload a video file or record directly from webcam.
2. **Transcription**: The audio is extracted and transcribed using the OpenAI Whisper model.

3. **Summarization**: The transcript is summarized using models like `t5-small` or Gemini via a prompt-based approach.

4. **Mind Map Generation**: The generated summary is structured into a mind map using models like Zephyr, Mistral, or Gemini.

5. **Persistence**: Summaries and metadata are saved into Redis with minimal latency.

6. **Frontend Rendering**: The results are rendered interactively using `vis-network` for mind maps and a custom UI for summaries and chapters.

Each component communicates over HTTP APIs, and Redis acts as a lightweight and efficient persistence layer.

3.4 Model Selection and Optimization

Transcription was handled using OpenAI's Whisper model. The `tiny` variant of Whisper was selected over larger alternatives due to its efficient balance between transcription accuracy and performance on CPU-based machines. In contrast, larger models such as `whisper-large` demanded GPU acceleration to avoid memory bottlenecks and introduced significant inference latency that rendered them unsuitable for a lightweight, scalable system.

Summarization initially leveraged the `T5-small` model, which provided a minimal memory footprint and acceptable summarization quality, making it ideal for early experimentation. However, when attempting to use larger models like facebook/bart-large-cnn through huggingface pipeline api or LLaMA through the `llama.cpp` library. For both the models resource limitations were encountered. Figure 4 shows an example where the Tesla T4 GPU was overloaded when attempting to run a quantized LLaMA model. This led to additional exploration into more structured summarization capabilities. For example, Gemini and Zephyr (quantized GGUF format) were used to produce chapter-based summaries enriched with timestamps. These models showed strong potential for generating coherent summaries with embedded structure. While BERT (in form of /facebook/bart-large-cnn) was also considered, it was ultimately rejected due to its unsuitability for generative summarization without additional decoder layers. However, if we add GPU to our VM bart-large-cnn works well. This has been tested using hugginface inference (Please the huggingface model card for detail `https://huggingface.co/facebook/bart-large-cnn` for bart-large-cnn [8].

```
dwiteekrishnapanda@flask-summarizer-gpu: ~/repo/youtube-summary-backend-gcp (ssh)
CPU[||||||||||||||||||||||||||||||||||||||||||||||||||||||||          48.1%]   Tasks: 42, 65 thr; 1 running
Mem[|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||3.29G/3.58G]   Load average: 0.83 0.57 0.28
Swp[                                                                0K/0K]   Uptime: 00:15:24
  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 1342 dwiteekri  20   0 31.6G 3107M  201M D 25.3 84.9  0:31.83 /usr/bin/python3 /usr/local/bin/gunicorn -w 1 -b 0.0.0.0:8080 app:app --timeout 900 --log-level debug
```

Figure 4: T4 Tesla GPU overloaded with llama.cpp model

Mind Map Generation was achieved using transformer-based language models including Mistral and Gemini. Through careful prompt engineering, these models were guided to output JSON-based hierarchical structures that could be visualized as mind maps. While Gemini produced more context-aware and logically structured output, its API-based usage incurred high costs. As a result, the system prioritized local inference by deploying quantized versions of these models in GGUF format to ensure affordability and efficiency without compromising on quality.

3.5 Cost Efficiency through Lightweight Models

Every transformer model (like T5, BART, GPT) has a maximum number of tokens it can process at once — called its context window.

Examples:

- t5-small: 512 tokens max
- bart-large-cnn: 1024 tokens
- gpt-3.5: 4096–16,000 tokens depending on version

To ensure the system is cost-effective:

- Quantized GGUF models were used with `llama.cpp` to run inference locally on CPU or a T4 GPU-enabled GCP VM.
- Caching of results using Redis avoids repeated computation.
- Summarization input is truncated to a fixed token limit (e.g., 400 words) to reduce inference cost and latency.

To leverage this format effectively, the system implementation included local deployment of the Zephyr model using an NVIDIA Tesla T4 GPU-enabled VM instance. Installation of the necessary runtime environment, refer 1. The GPU-compatible versions of PyTorch and supporting libraries has been installed.

3.6 Non-Functional Requirements

Low Latency

- Transcription and summarization are optimized via caching and truncation.
- Mind map generation uses fast local inference.
- Redis ensures millisecond-range lookup for previous results.

High Availability

- Stateless Flask APIs support horizontal scaling.

- GCS and Redis are highly available managed services.

High Throughput

- Whisper and T5 are lightweight enough to allow parallel processing.
- Job queuing and Redis cache reduce bottlenecks during multiple requests.

Resilience and Fault Tolerance

- All endpoints implement defensive error handling with meaningful HTTP responses.
- Fallback logic is included for GPU unavailability (`device = "cuda" if torch.cuda.is_available() else "cpu"`).

3.7 Summary of System Design and Implementation

The technical implementation behind ReMindMap is optimized for low-cost deployment, fast inference, and production-readiness. By combining lightweight models, local GPU/CPU execution, and a modular architecture, this system demonstrates how state-of-the-art NLP and generative AI can be brought to production for accessibility use cases—without incurring high cloud AI costs.

3.8 Low Level Design for Backend Implementation

This section describes few of the important backedn API as well as the methods they call as per the pipeline sequence (See appendix listing 26 for dependencies and libraries).

3.8.1 API: /upload-thumb

- **URL:** `/upload-thumb`
- **Method:** `POST`
- **Content-Type:** `multipart/form-data`
- **Functionality:**
  - Expects an image file (typically a PNG) in the request with form field name `file`.
  - Uploads the file to the **`thumbnails/`** folder in the configured Google Cloud Storage bucket .
  - Generates a public URL for the uploaded file and returns it in the response.
- **Use case:** This endpoint handles the upload of thumbnail images for videos to Google Cloud Storage. As soon as a video is selected on the frontend the first this the frontend does it to get a thumbnail from that video . This Thumbnail will be used when we list down the already summarized video. There will be a separate API called /list-summaries which send the list of all summarized content with all meta data , one of the meta data would be the Thumbnail.
- Please refer Appendix listing 3 for code snippet

3.8.2 API: /upload-video

- **URL:** `/upload-video`
- **Method:** `POST`
- **Content-Type:** `multipart/form-data`
- **Functionality:**
  - Expects a video file (e.g., WebM or MP4) in the request with form field name `file`.
  - Uploads the file to the `videos/` folder in the configured Google Cloud Storage bucket.
  - Generates a public URL for the uploaded video and returns it in the response.

- **Use case:** This endpoint handles the upload of the actual video content that will later be summarized. Once the frontend extracts a clip or full-length video, it uploads it through this API. The backend stores the video in the cloud and returns a public URL. This URL is then stored as part of the video summary metadata and is retrievable via the `/list-summaries` API for playback or reference. This is only required to play the video along side the summarized content as we don not have the local video file if list summary is viewed from some other device.
- Please refer Appendix listing 7 for code snippet

3.8.3 API: /submit-video-to-summarize

- **URL:** `/submit-video-to-summarize`
- **Method:** `POST`
- **Content-Type:** `application/json`
- **Functionality:**
  - Accepts metadata about the video to be summarized, including `id`, `title`, `thumbnailUrl`, and `videoUrl`.
  - Performs basic validation to ensure all required fields are present.
  - Delegates processing to `submit_video_to_summarize_handler()` in `job_processor.py`, which executes the following steps:
    1. Downloads the video from the provided `videoUrl` and saves it temporarily.
    2. Extracts the audio stream using FFmpeg and saves it in WAV format.
    3. Starts a new background thread to run `process_job()` which:
       * Transcribes the audio using Whisper.
       * Summarizes the transcript using the selected AI model (e.g., Gemini).
       * Saves the output summary in Redis using `/save-summary`.
    4. Deletes the temporary video file after audio extraction.
  - Returns a unique `job_id` that can be used to track the job status via `/job-result/<job_id>`.
- **Use case:** This endpoint acts as the central trigger for the summarization pipeline. After uploading the video and thumbnail, the frontend calls this API with the metadata and video URL. The backend takes care of all heavy processing in the background. The decoupled job-based design ensures responsiveness and scalability while allowing the user to retrieve the result later through polling.
- Please refer Appendix listing 11 for code snippet

3.8.4 Module: job_processor.py

- **Purpose:** Handles all long-running and asynchronous tasks related to transcription and summarization of audio or video files.
- **Functionality:**
  - Accepts incoming requests via handlers like `submit_video_to_summarize_handler()` and `submit_job_handler()`.
  - Downloads video files from a given URL and extracts the audio stream using `ffmpeg`.
  - Uses background threads to call the `process_job()` function without blocking the main API response.
  - Computes a unique hash of the audio file content to check for existing cached results in Redis.
  - If cache is found, returns the existing transcript and summary instantly.
  - Otherwise, transcribes the audio using Whisper and generates a summary using the chosen model (e.g., Gemini or T5).
  - Caches both transcript and summary in Redis for 2 days (172800 seconds).
  - Tracks job completion status and stores the summary in the `job_results` dictionary using the job ID.
- **Use case:** This module is critical to ensuring scalability and responsiveness of the backend system. Transcription and summarization for long videos can take several seconds to minutes. Without this asynchronous design, API requests would timeout and overload the server. By using threads and caching, `job_processor.py` enables real-time interaction for users while jobs are safely executed in the background.
- **Key Components:**
  - `process_job(file_path, job_id, model_name)` – Handles the full summarization workflow.
  - `submit_video_to_summarize_handler()` – Accepts metadata and video URL, extracts audio, and queues the job.
  - `submit_job_handler()` – Accepts raw audio file uploads and queues the job.
  - `job_result_handler(job_id)` – Provides job status and summary result.
  - Please refer Appendix listing 16 for snippets.

3.8.5 ASR Module: Whisper-based Transcription

- **Purpose:** To transcribe audio extracted from video files into plain text as input for summarization.

- **Model:** The system uses the **open-source Whisper model**, developed by OpenAI, and made available via Hugging Face's `whisper` package.
- **Device Optimization:**
  - Automatically detects GPU availability using `torch.cuda.is_available()`.
  - Loads the model on `cuda` if available; otherwise defaults to `cpu`.
- **Functionality:**
  - Loads the `tiny` Whisper model for low-latency transcription.
  - Transcribes audio using `model.transcribe()` and extracts the text from the result.
  - Provides debug logs including device usage and word count.
- **Use case:** This transcription layer forms the first step in the summarization pipeline. It converts spoken content into text, which is then summarized using LLMs like Gemini or T5.
- Please refer Appendix for code snippet 17

3.8.6 Caching Mechanism for Transcript, Summary, and Metadata

- **Purpose:** To avoid redundant computation and reduce API costs by reusing already generated results for identical video/audio content.
- **Mechanism:**
  - A unique MD5 hash is computed for every uploaded audio file using its binary content.
  - Before transcription and summarization, the system checks Redis for existing keys:
    * `<hash>_transcript`
    * `<hash>_summary`
    * `<hash>_metadata`
  - If all exist, the job uses the cached versions and skips reprocessing.
  - If not, it runs Whisper and the selected summarization model, then stores:
    * Full transcript
    * Generated summary
    * Metadata (e.g., model used, summary length, transcript length) as a JSON blob
  - Each cache entry TTL is set 172800 which is 2 days.

- The above are individual cache means if individually the same audio is sent for transcribing then it will return cached transcript, and individually a mind map is requested then is its the same text is sent within the TTL then it will return the cached mindmap. For the fully summarized video is also kept on redis but the key pattern is as follow (please see Appendix listing 27 for the value of the key stored in redis).
- Redis key pattern for a fully processed video is `summary:<hash>`
- **Use case:** This caching strategy significantly reduces processing time and cost when summarizing duplicate or repeated content, especially in academic or multi-user environments where the same media might be submitted more than once.

3.8.7 Summarization using T5-small Model

- **Purpose:** To generate a concise summary of long-form text using the lightweight `t5-small` transformer model.
- **Model:** Hugging Face's pre-trained `t5-small` model via the `transformers.pipeline` API.
- **Functionality:**
  - The input text is first stripped and tokenized into words.
  - The words are split into fixed-size chunks (400 words per chunk) to meet the input token limit of the model.
  - Each chunk is summarized independently using the T5 pipeline.
  - All chunk-level summaries are concatenated to form the final summarized output.
- **Chunk-wise Processing:** This strategy helps bypass token limits imposed by smaller transformer models without losing significant information across long documents.
- **Use case:** This function is particularly useful when:
  - The Gemini model is unavailable or expensive.
  - Fast CPU-only summarization is required.
- Please refer Appendix for code snippet 18

3.8.8 Summarization using Gemini Model

- **Purpose:** To generate rich, chapter-based summaries of long video transcripts using Google's Gemini multimodal language model.

- **Model:** `gemini-2.0-flash-001`, accessed through Google's Vertex AI via the Python SDK `google.genai`.
- **Functionality:**
  - The input transcript is inserted into a carefully designed prompt template called `CHAPTERIZE_PROMPT_TEMPLATE`.
  - A Gemini chat session is created using the specified model and region.
  - The prompt is sent using the chat interface, and the response is expected to be a valid JSON array of chapters.
  - The response is parsed as JSON and returned as a formatted string; if parsing fails, a runtime error is raised.
- **Use case:** I am using this to generate chapter-level summaries with time stamp for long videos, especially where comprehension and accessibility are key. These time stamps are used to jump video time line when a chapter is clicked on UI.
- Please refer Appendix for code snippet 19
- Please refer Appendix for sample json for the chapterized summary with time stamp

3.8.9 Mind Map Generation with Gemini and Mistral Models

- **Purpose:** To convert the AI-generated summary into a hierarchical mind map structure to improve accessibility and comprehension for users with cognitive challenges.
- **Prompt Design:** Both models use a common prompt template that:
  - Specifies the structure of the mind map: central topic, branches, subpoints.
  - Requires each node to include both a `label` (with an emoji) and a `narration` (simple explanation).
  - Asks for output in strict JSON format with no extra text.
- **Gemini-based Generator (`generate_mindmap_gemini`)**
  - Uses Google's Vertex AI SDK to send the structured prompt to `gemini-2.0-flash-001`.
  - Parses the response using regex and attempts to deserialize it into JSON.
  - If parsing fails, raises a structured error for debugging.
- **Mistral-based Generator (`generate_mindmap_mistral`)**
  - Prepares the same prompt for local or Hugging Face-hosted Mistral model.
  - Currently serves as a placeholder for full integration.

- **Use case:** Once a video transcript is summarized, these functions convert the result into a visual mind map. This enables comprehension via both verbal and visual structures. Please refer 21 to see a sample structure of json that is used in the Front to render and narrate the mindmap.
- Please refer Appendix listing 22 for code snippet 20

3.8.10 API: /save-summary

- **URL:** `/save-summary`
- **Method:** `POST`
- **Content-Type:** `application/json`
- **Functionality:**
  - Receives a JSON object representing a full summary entry.
  - Extracts the `id` field from the payload and saves the entry in Redis using the key pattern `summary:{id}`.
  - Does not persist the `thumbnail` field to save Redis storage space.
- **Use case:** Called automatically after a summarization pipeline completes to persist summary metadata and allow frontend listing/retrieval.
- **Redis key format:** `summary:{id}`
- Please refer Appendix for code snippet 23

3.8.11 API: /list-summaries

- **URL:** `/list-summaries`
- **Method:** `GET`
- **Functionality:**
  - Retrieves all keys from Redis that match the prefix `summary:*`.
  - Parses and returns the stored JSON values as a list.
- **Use case:** Enables the frontend to fetch a list of all previously summarized videos, including metadata like title, summary, video URL, etc.
- **Response format:** JSON array of summary objects
- Please refer Appendix for code snippet 24

3.8.12 API: /download-youtube-and-submit

- **URL:** `/download-youtube-and-submit`
- **Method:** `POST`
- **Functionality:**
  - Accepts a YouTube video URL in the request body.
  - Validates the URL and extracts the video ID.
  - Downloads only the first 7 minutes of the video using `yt-dlp`.
  - Uses cookies for authentication if available (`youtube_cookies.txt`).
  - Uploads the downloaded video to Google Cloud Storage (GCS).
  - Returns a metadata JSON object including video ID, title, thumbnail URL, and public GCS video URL.
- **Use case:** Allows users to submit a YouTube URL and get it preprocessed for summarization. This is useful in scenarios where users prefer to analyze public video content directly from YouTube instead of uploading files manually.
- **Response format:** JSON object with fields `id`, `title`, `thumbnailUrl`, and `videoUrl`.
- Please refer Appendix for code snippet 25. Please note this is currently not been called from frontend as mentioned in section 4.1.

3.9 Low Level Design for Frontend Implementation

Figure 5 illustrates the end-to-end interaction between the frontend and backend in the ReMindMap system. The user initiates the process by uploading or recording a video. This triggers backend APIs responsible for storing the video, generating thumbnails and queuing the summarization job. Once processed, the frontend then retrieves and renders chapterized summaries and visual mind maps. This diagram highlights how the frontend manages asynchronous operations such as job polling and dynamic content rendering to maintain a smooth user experience.

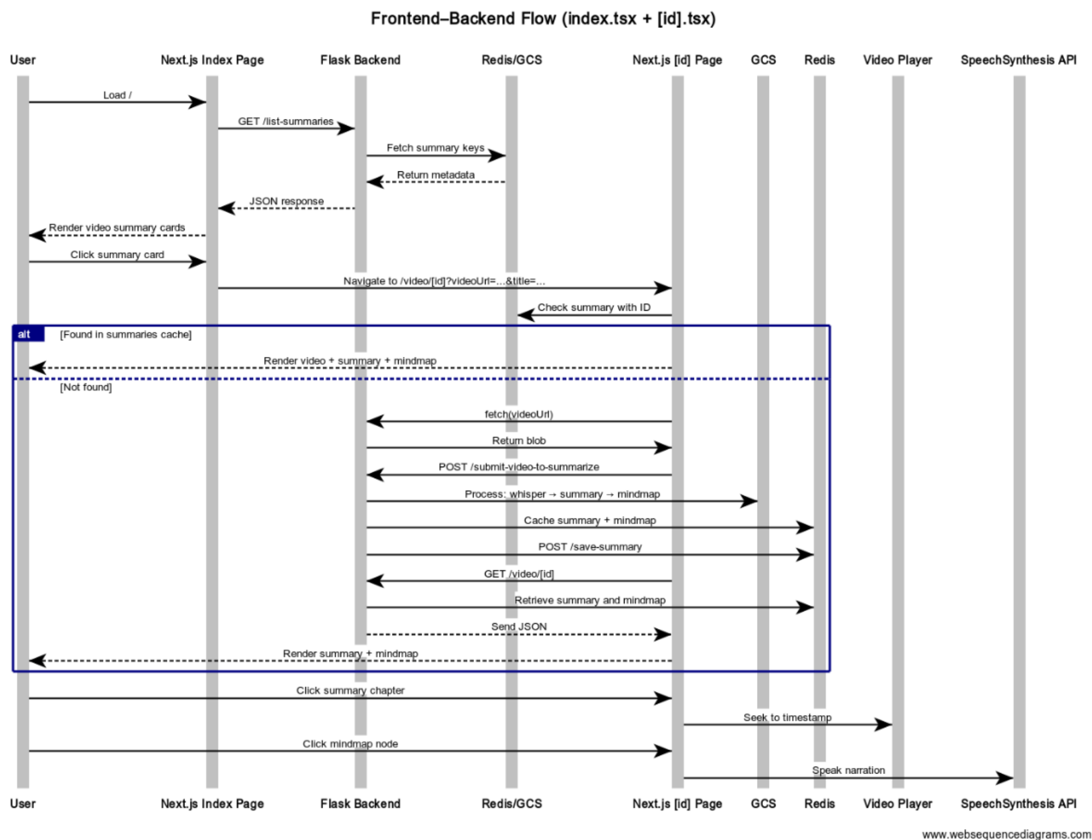

Figure 5: Frontend–Backend Interaction Sequence Diagram

High-Resolution Version: `https://github.com/Dwitee/video-summarizer-frontend/blob/main/sequence_diagram.png`

3.9.1 Frontend Design and Framework Decisions

The frontend development for ReMindMap began with Gradio, a Python-based UI toolkit tailored for rapid prototyping of machine learning applications. Gradio pro-

vided an excellent starting point due to its seamless integration with Python scripts and its ability to render functional interfaces quickly without needing separate frontend frameworks. This allowed early experimentation with Whisper-based transcription and summarization workflows, providing fast feedback during the research phase.

However, as the project evolved toward a production-grade system, Gradio began to show its limitations. It lacked support for responsive layout, customizable components, and advanced user interactions like sticky panels, fullscreen toggles, and reusable UI states. It also became clear that integrating visually complex components such as interactive mind maps or synchronized video playback would require greater control over the frontend rendering logic than Gradio could provide.

To address these issues, the entire frontend was rewritten using Next.js 14—a full-stack React framework that offers a modern web development environment. This transition allowed for modular component development, custom styling, and support for dynamic routing. The use of dynamic pages (e.g., `[id].tsx`) made it possible to deep link to specific video summaries and restore application state based on video IDs. Next.js also provided seamless integration with server-side rendering and incremental static regeneration, which greatly improved performance and loading times.

Accessibility was a key focus throughout the redesign. The interface followed aphasia-friendly principles by minimizing cognitive overload, reducing interface clutter, and using larger fonts and simplified language. The interface supported text-to-speech playback, sticky video/mind map panels for persistent context, and intuitive playback controls—all essential for neurodiverse users.

Overall, the migration from Gradio to Next.js with Vercel deployment was a strategic shift that allowed the project to scale beyond its prototype phase. It provided the robustness, performance, and user experience required for real-world accessibility applications, and aligned the frontend with modern web development practices while maintaining full control over every interactive element of the application.

The migration of UI from gradio to next.js , which result in great UI Intutive improvement can be clearly seen when comparing both old and new UI layout. (see the figure 6 and 7 and compare them).

3.9.2 CI/CD pipeline

In terms of deployment, Vercel was chosen as the hosting platform due to its tight integration with Next.js and features such as automatic builds, fast CDN-based delivery, and rollback capabilities. With Vercel, every commit to the GitHub repository triggered a build and deployment pipeline, ensuring continuous integration and fast iterations. (see the appendix fig 21)

3.9.3 Responsiveness and Device Compatibility

The frontend layout is fully responsive, adapting to both desktop and mobile/iPads (please see 13) viewports using Tailwind's responsive utility classes. Elements such as the

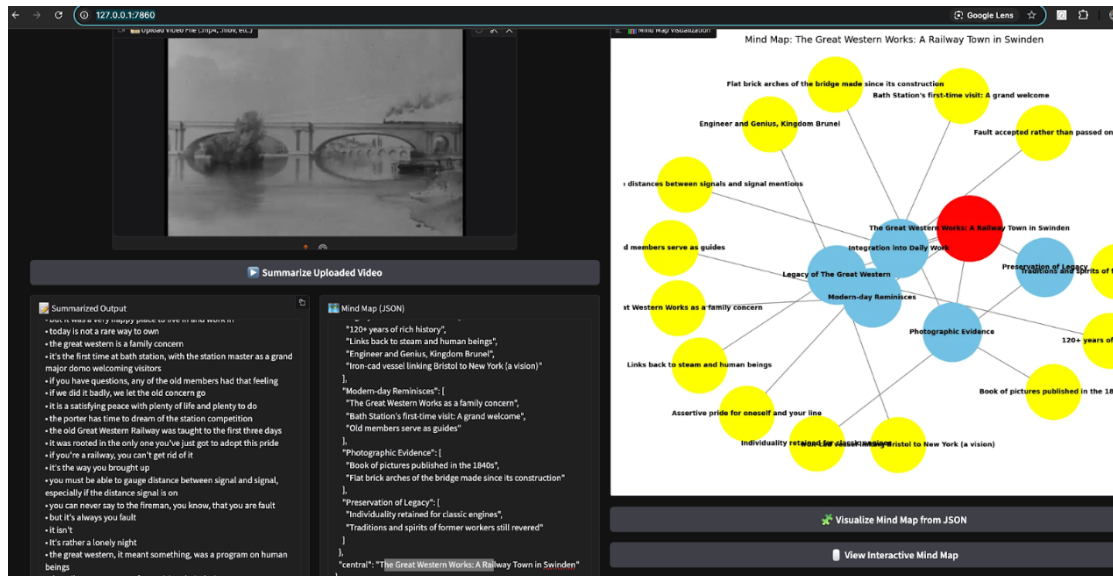

Figure 6: Gradio based UI (old)

Figure 7: Next.js based UI (new)

video player, summary sections, and interactive map adjust their widths and stacking order based on screen size to ensure usability across a wide range of devices.

3.9.4 Frontend Libraries and Their Purposes

The table below lists all the major libraries and packages used in the frontend, along with a brief description of their role in the system.

Table 2: Frontend Dependencies and Their Purposes

Package	**Purpose**
`next`	Full-stack React framework used for routing, server-side rendering, and static site generation.
`react`	Core UI library used to build reusable components and manage state.
`react-dom`	Provides DOM-specific rendering methods for React components.
`uuid`	Generates unique identifiers for video IDs and DOM nodes.
`vis-network`	Used to render interactive mind maps with dynamic, hierarchical graph layouts.
`vis-data`	Provides the node and edge data model consumed by `vis-network`.
`@ffmpeg/ffmpeg`	Enables video processing in the browser using FFmpeg compiled to WebAssembly.
`@ffmpeg/core`	Core FFmpeg WebAssembly binary used by `@ffmpeg/ffmpeg`.

3.10 Results, Analysis and Evaluation

3.10.1 Overview of System Outputs

The `ReMindMap` system is designed to make video and audio content more accessible by providing structured summaries and interactive visualizations. Users can initiate the system in multiple ways: by uploading a pre-recorded video file (in any standard format), recording a live video/audio (e.g., a lecture, conversation, or radio clip). (See Figure 8) Once the content is submitted, the backend processes the input through several components to generate a simplified and structured output.

The system extracts audio from the input, performs automatic speech recognition using the Whisper model, and then generates a chapterized summary using a transformer-based summarization model. The resulting output includes a concise, timestamped summary broken down into semantically coherent chapters. Additionally, the summary is visualized as an interactive mind map, which adapts to various comprehension needs.

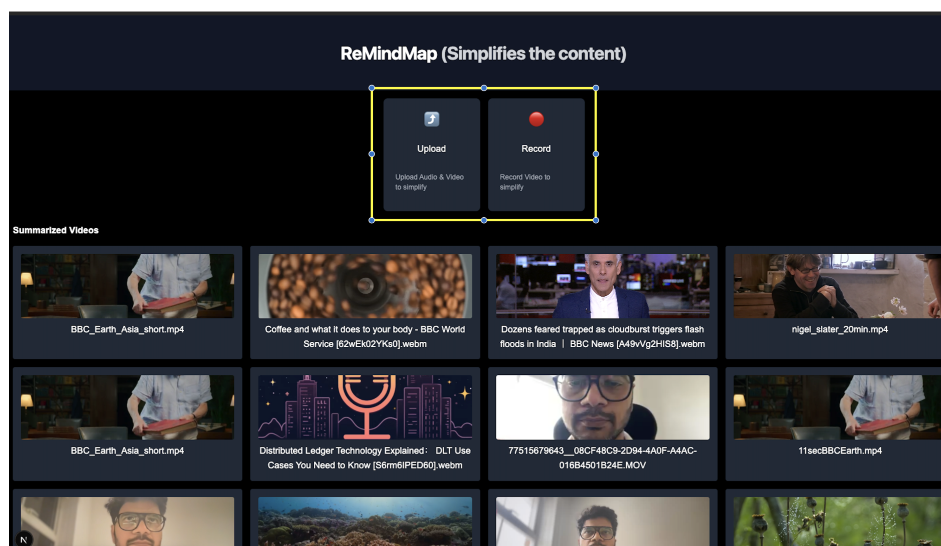

Figure 8: User uploading or recording content through the ReMindMap interface

The mind map offers two modes of visualization: the **Standard Mode** (See Figure 9), which lays out all nodes hierarchically, and the **Radial Mode** (See Figure 10), which keeps the mind map compact by expanding subtopics only upon user interaction. Both layouts are designed to suit different cognitive preferences and reduce information overload for neurodiverse users.

Figure 9: Mind map in standard (expanded) mode with all chapters visible

Figure 10: Mind map in radial (compact) mode with interactive expansion

The Narration Mode (See Figure 11) provides automatic text-to-speech (TTS) support. Users can play the narrated mind map at adjustable speeds (0.5× and 0.25×), particularly useful for users with speech or language impairments such as aphasia. During narration, each node is focused sequentially; the associated concept is spoken aloud while the corresponding text is simultaneously shown at the bottom of the screen in an aphasia-friendly font. To enhance cognitive anchoring, each node also displays a contextually relevant emoji in both the node body and the top-left corner.

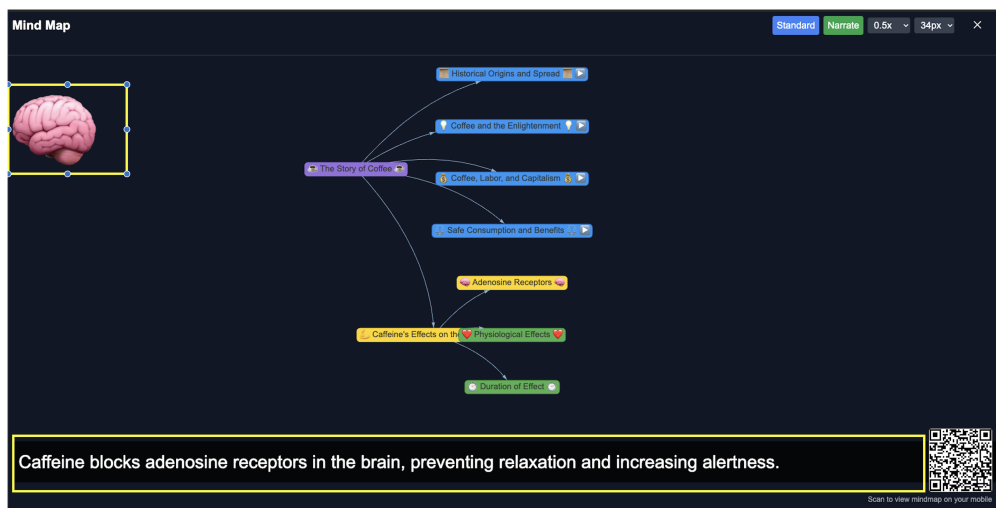

Figure 11: Narration mode with emoji, focused node, and text display

In **Interactive Mode** (See Figure 11, users can manually click on any node to hear its content narrated. This enables them to selectively explore areas of interest or re-listen to parts they find challenging. The same accessibility features—adjustable speed, on-screen narration text, and pictographic emojis—remain active in this mode.

Furthermore, **Dual Mode** interaction enables a distributed experience. A QR Code displayed on the main screen (See Figure 12) can be scanned to open the mind map on a secondary handheld device (e.g., phone or tablet). This facilitates multi-user environments such as classrooms, where each participant can access their own synchronized visual representation of the same content (See Figure 18) .

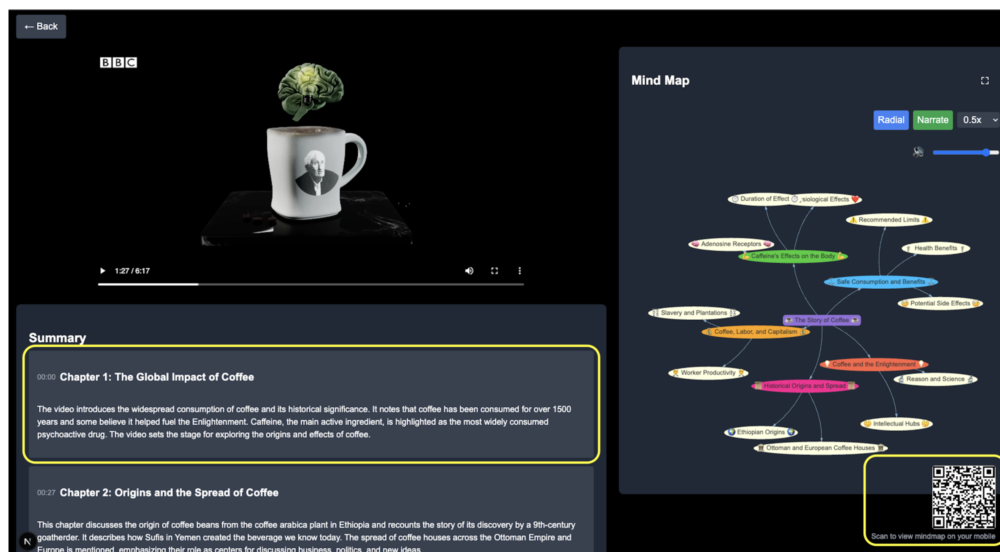

Figure 12: Dual device interaction: User can scan the QR Code from his device to see and interact in his device.

Once the user scans the QR Code displayed on the screen with Video , he can see and interact with the mindmap in his own device.

The end-to-end design ensures accessibility, interactivity, and customization. From flexible input methods to multimodal output presentation, `ReMindMap` supports diverse learning styles and cognitive needs, fulfilling its mission to make media more comprehensible and inclusive. Also supports various resolutions device like iPad , Phone and Computer and TV Screen (Please see Figure 13 for various Resolution Support and cross platform capability.

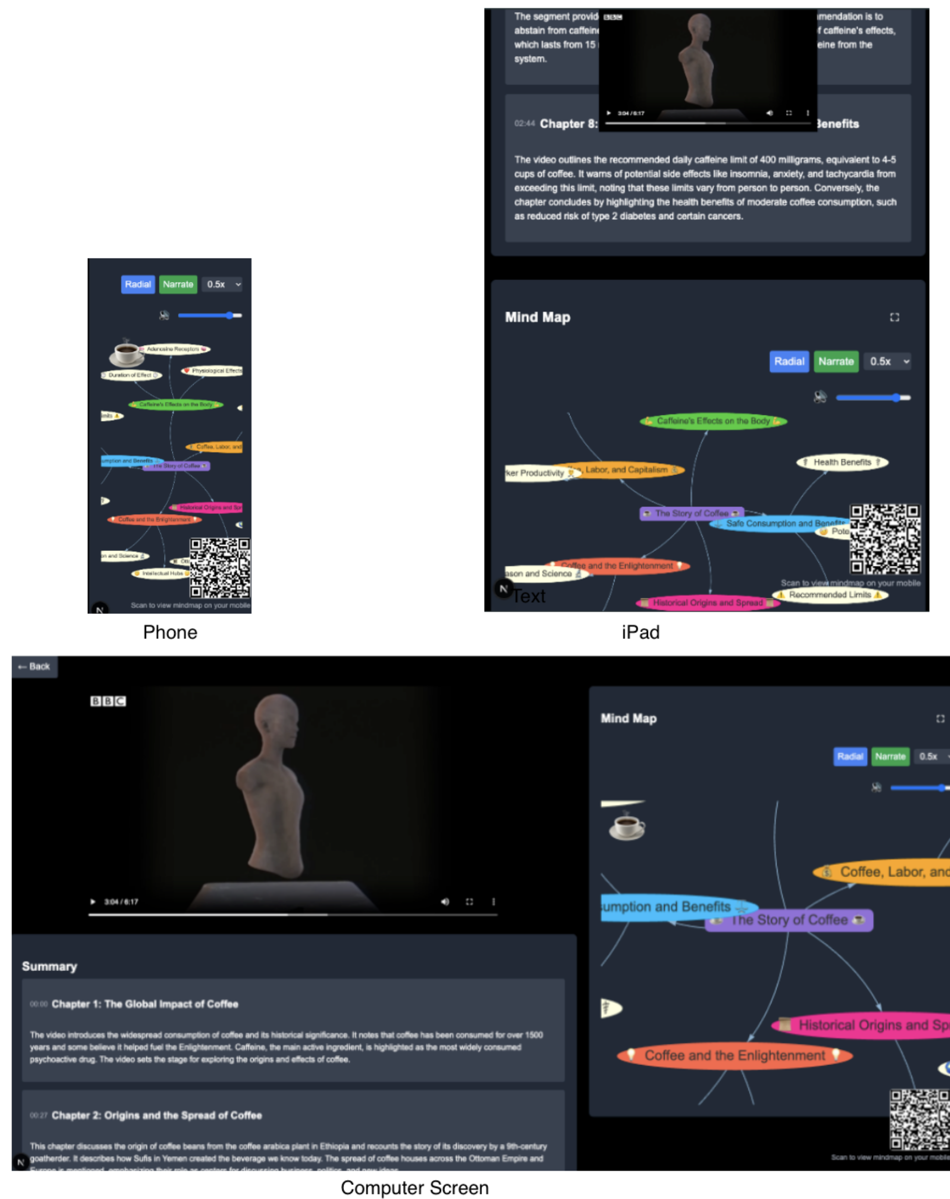

Figure 13: Cross Platform as a web App and supporting different resolutions

3.11 Quntitative Runtime Analysis

This section evaluates the runtime performance of different components of the **ReMindMap** system, with a focus on transcription, summarization, and system-level resource utilization. The analysis was conducted using custom log instrumentation as well as metrics collected from Google Cloud Monitoring using **CloudLoggingHandler** provided by GCP.

3.12 Execution Time: Transcribing vs. Summarization

The backend logs record the time taken for each function execution. Two core components were benchmarked:

- `transcribe_with_whisper` – Performs automatic speech recognition using OpenAI's Whisper model.
- `summarize_text` – Uses a transformer-based model (such as T5) to generate concise summaries.

As observed in the figure below, Whisper-based transcription consistently takes between 25–35 seconds, while summarization is significantly faster, requiring only 5–8 seconds on average.

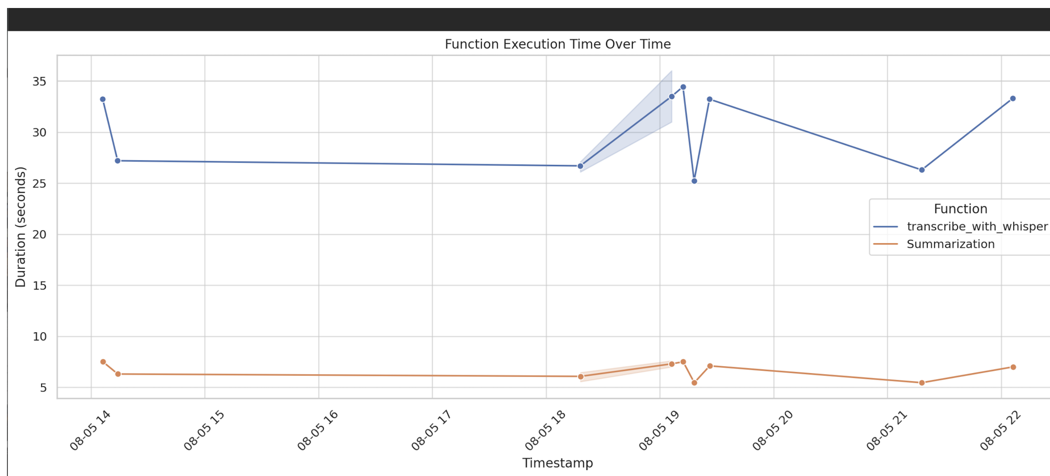

Figure 14: Execution Time Comparison: Whisper vs Summarization

Observation: If a subtitle file (such as `.srt`) is available alongside the video, the transcription step can be skipped. This leads to a significant speed-up in the overall pipeline.

3.13 Inference Performance: T5 vs Gemini vs Zephyr

This subsection compares the runtime response time using various models used for summarization tasks. The Input data set is video file with duration ranging from 2 minutes to 10 minutes. The different models were using for the same set of video file to compare the response time. Please note that Zephyr model could not process videos longer then 4 minutes with out a GPU. You can see this as the graph for Zephyr is broken. The Gemini 2.0 flash performs best but it a paid model . Further discussion on trade-off has been done in the Following sections.

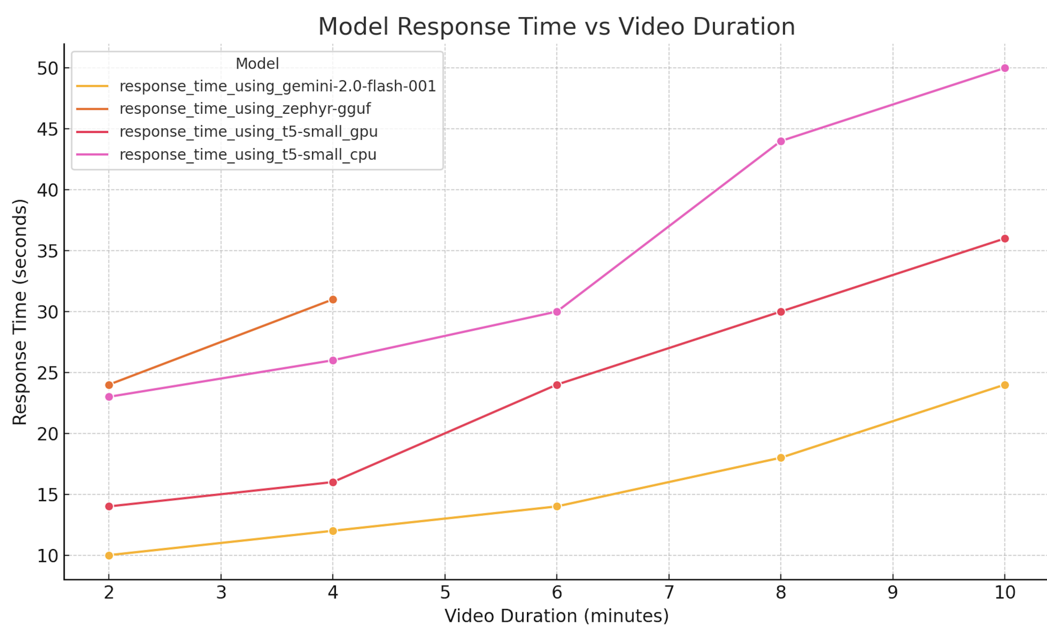

Figure 15: Performance Comparison of various models , in x-axis video duration in y-axis response time (all included).

Although gemini-2.0-flash provides higher-quality generative summaries with lesser response time, but its worth noting that this is part of Google cloud vertex AI as it follows pricing structure. Please see `https://cloud.google.com/vertex-ai/generative-ai/pricing` .The choice depends on the trade-off between quality and latency and cost.

Token-based pricing   Modality-based pricing

Model	Type	Price	Price with Batch API
Gemini 2.0 Flash	1M Input tokens	$0.15	$0.075
	1M Input audio tokens	$1.00	$0.50
	1M Output text tokens	$0.60	$0.30

Figure 16: Gemini 2.0 Flash Pricing by Google Vertex AI)

3.14 System Utilization Metrics from Google Cloud

Google Cloud's observability tools were used to collect VM-level metrics during backend API execution.

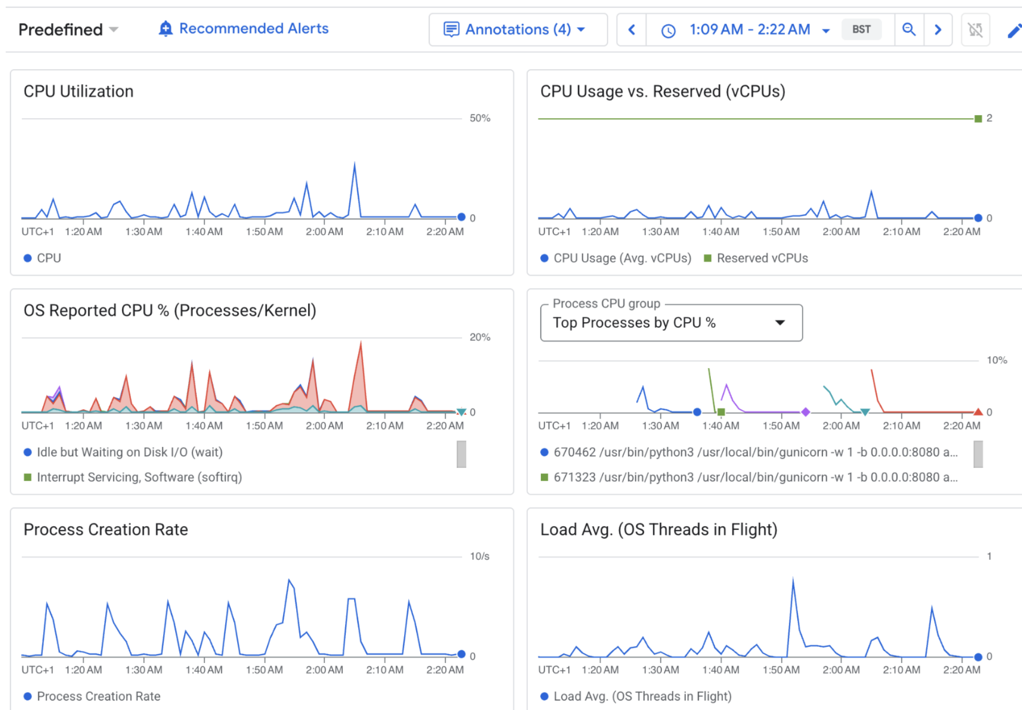

Figure 17: System Metrics During Summarization (CPU Load, Process Rate, Thread Count)

Key Insights: CPU utilization remained below 50% even during peak inference periods, indicating efficient resource usage. The primary contributors to CPU load were the Gunicorn worker threads responsible for handling API requests. Additionally, noticeable spikes in process creation and thread count were observed at the initiation of summarization or transcription tasks, suggesting that these operations are the primary drivers of transient load on the system.

3.15 Local Run Model Vs API based Model

When we compare t5-small model running locally and gemini api based model using vertex api, we consider response time of a 10 minute video , using ts-small (with and without CUDA) and then gemini 2.0 flash model using google cloud Vertex AI.The t5-small runs on a single 7.5GB Tesla GPU so we only consider this VM running cost with a GPU. For Gemini 2.0 flash model we use the pricing structure provided by Google cloud (please see `https://cloud.google.com/vertex-ai/generative-ai/pricing#token-based-pricing`) This pricing of gemini model is per 1 Million tokens and typically , 10 minute video as around 800 words can corresponds to 1000 tokens. So as per the gemini 2.0 flash pricing mode (see Figure 16) a 10 minute video with 1000 input token would cost $0.15/1000 = 0.00015$ USD, We also have output tokens which are luckily

always be lesser because summarization always will have less then half of input token. So, for 10 minute video the outputs token are arround 500. So the Cost is 0.0007 USD. The running cost of a VM with 1 GPU is marginally very less but we need to pay this to run the server even if not using the processing power for summarization. For 1 Month 10 hour a day typically it would be less then 50 USD.

3.16 Pros and Cons: GPU vs. CPU

GPU Execution: Executing models on a GPU substantially reduces inference latency, often achieving 3x to 5x faster processing compared to CPU-only execution. This speed advantage makes GPU-based processing suitable for real-time or interactive use cases, such as in classrooms or live captioning environments. However, this performance comes at the cost of higher operational expenses and potential resource constraints, especially when deployed at scale.

CPU Execution: In contrast, CPU-based execution is generally more cost-effective and easier to scale for applications with limited throughput requirements. It is particularly effective when subtitle or transcript files are already available, as the need for intensive audio processing is eliminated. Nonetheless, processing raw video or audio inputs on a CPU results in slower performance, making it less ideal for time-sensitive applications.

Conclusion:

For high-availability deployments, GPU-backed infrastructure is preferred. However, with intelligent preprocessing (e.g., skipping Whisper when transcripts exist), a CPU-only setup can still be viable and cost-effective. GPU based infra would be great in a situation where the response time is insignificant and we have huge amount of video to process (which can also be processed in batch to save processing power. How ever for real time usage such as live recording when you want instantly the summarized content of a live speech or lecture, API based inference model like gemini 2.0 flash can be used for higher quality and faster response. Although Dist

4 Conclusion

This project set out to develop **ReMindMap**, an AI-powered accessibility tool designed to make video and audio media more comprehensible for individuals with communication challenges, particularly those with aphasia. By combining modern natural language processing models with user-centric interface design, the system successfully meets its objectives of delivering chapterized summaries and interactive visual mind maps from multimedia inputs.

The backend, implemented in Python and hosted via Gunicorn on a GPU-enabled Google Cloud VM, integrates Whisper for speech recognition and transformer-based models like T5 and Gemini for summarization. The frontend, developed using Next.js and deployed through Vercel, offers both intuitive video interaction and interactive mind map exploration across devices. Key accessibility features such as aphasia-friendly narration, dual-screen view via QR code, and speed-customized playback were implemented to cater to diverse user needs.

This project has explored various AI Models, specifically transformer models, to achieve its goal of making audio and video media content more accessible. Users can tweak the interface—such as changing narration speed or selecting mind map display modes—and customize the experience to their individual needs. Nonetheless, default settings, including font size and narration speed, are designed with simplicity in mind to suit most neurodiverse users.

The Live Recording mode shows significant potential in various settings such as classroom lectures, spoken instructions, or even real-time radio content. It instantly produces a structured, chapterized summary and an interactive mind map to enhance comprehension. The Dual Mode (please see figure 18 where each individual user can have own summarized mind in his personal device while watching a video in a group has immense potential for extensibility.This dual mode can be done just by scanning a QR Code displayed with video. This application is production-ready, fully deployed on Vercel, and can be tested to experience all of its described features.

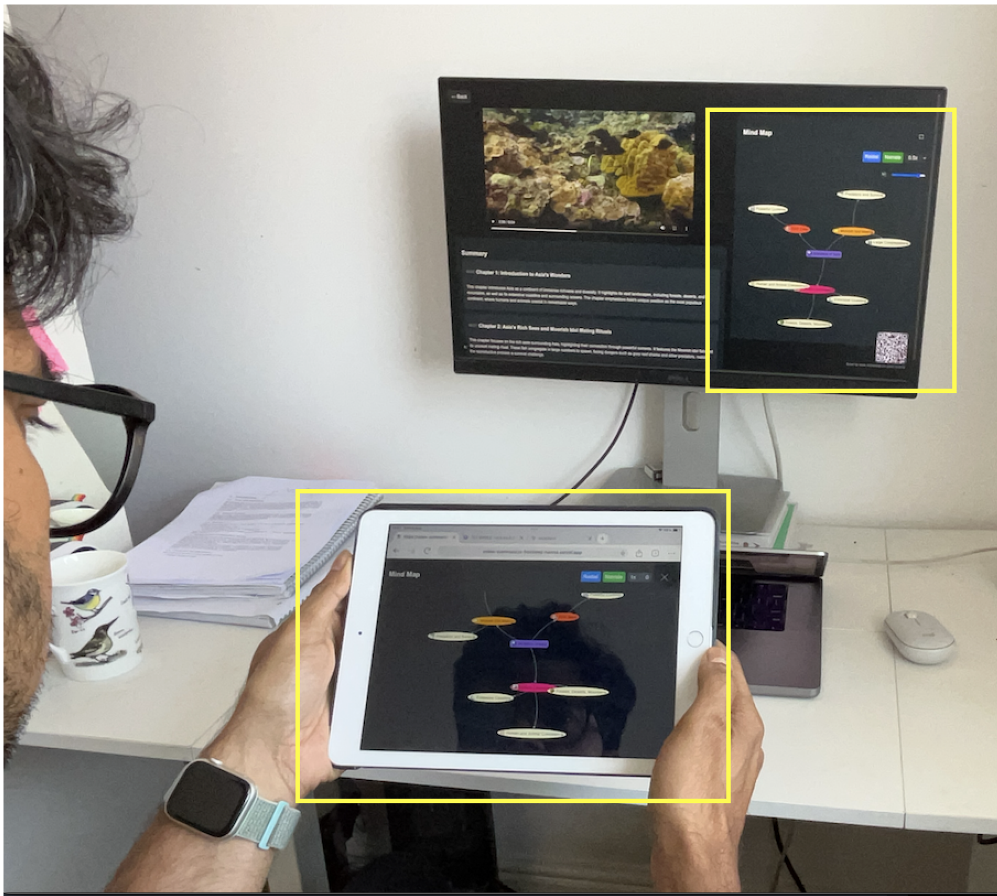

Figure 18: Dual device interaction: viewing video on a desktop while the same mind map is accessible on a iPad device

Through performance benchmarks and response time logging, the system demonstrated efficient GPU utilization and highlighted trade-offs between local inference and API-based summarization models. The results underscore the benefit of using quantized local models for real-time summarization, while acknowledging the scalability advantages of cloud APIs such as `gemini 2.0 flash` .

4.1 Future Work and Scalability

ReMindMap is designed with scalability in mind. To support higher demand, additional VM instances can be added with load balancing. GPU support can be scaled up to further reduce processing latency after which instead of `whisper-tiny` higher model like `whisper-small` can be used for better performance (please see 1).

The system is already capable of summarizing content from YouTube URLs, the only requirement is to provide valid cookies to bypass restrictions on certain videos such as those for kids or age-restricted content. This is reason as of now the `youtube url`

`paste` card has been disabled. However, this feature has been tested and successfully uses `yt-dlp` to download YouTube videos, after which the same summarization pipeline is applied. Furthermore, a chrome browser plugin can also be made with the same source-code to summarize over youtube on a browser (please see proof-of-concept screenshots on Appendeix 19 and 20) .

The modular backend architecture ensures that any future locally-run model can be integrated seamlessly through HuggingFace's pipeline framework. Already the bart-large-cnn has been tried through huggingface pipeline api. If add more GPU to VM then we can run the `bart-large-cnn` or similar large model can be used for further accuracy and faster response. This makes ReMindMap future-proof and ready for rapid model upgrades.

Code and Deployment

The complete source code for both backend and frontend, along with the production deployment link, is available below:

- **Backend GitHub Repo**: `https://github.com/Dwitee/youtube-summary-backend-gcp`
- **Frontend GitHub Repo**: `https://github.com/Dwitee/video-summarizer-frontend`
- **Production Application Endpoint**: `https://video-summarizer-frontend-henna.vercel.app/`

Please scan the QR Code to see the application live and running

Looking forward, future work can focus on enabling real-time multilingual summarization using larger models with more compute power, and refining the UI/UX based on user studies and A/B testing. ReMindMap holds significant promise as an assistive technology platform for inclusive education, therapy, and media accessibility.

References

[1] Madeline N Cruice Stephanie Wilson Elena Simperl Timothy Neate Alexandre Nevsky, Filip Bircanin. ”i wish you could make the camera stand still”: Envisioning media accessibility interventions with people with aphasia. In *Proceedings of the 26th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS ’24)*. Association for Computing Machinery, 2024. doi:10.1145/3663548.3675598.

[2] Andy Brown, Rhia Jones, Mike Crabb, James Sandford, Matthew Brooks, Mike Armstrong, and Caroline Jay. Dynamic subtitles: The user experience. In *Proceedings of the 2nd ACM International Conference on Interactive Experiences for TV and Online Video (TVX ’15)*. Association for Computing Machinery, 2015. doi:10.1145/2745197.2745204.

[3] A Carter, G Anderson, and A Omoseebi. Multimodal deep learning for assistive technology: Integrating sign language recognition, speech processing, and medical image analysis for inclusive communication and healthcare. *arXiv preprint arXiv:2401.12345*, 2024. URL: https://tinyurl.com/z9ep4enj.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

[5] Jorge Gabín, M Eduardo Ares, and Javier Parapar. Enhancing automatic keyphrase labelling with text-to-text transfer transformer (t5) architecture: A framework for keyphrase generation and filtering. *arXiv preprint arXiv:2409.16760*, 2024.

[6] Ruth Herbert. accessible information guidelines. URL: https://www.stroke.org.uk/sites/default/files/accessible_information_guidelines.pdf.

[7] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd edition, 2025. Online manuscript released January 12, 2025. URL: https://web.stanford.edu/~jurafsky/slp3/.

[8] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461, 2019. URL: http://arxiv.org/abs/1910.13461, arXiv:1910.13461.

[9] Prudhvi Naayini, Praveen Kumar Myakala, Chiranjeevi Bura, Anil Kumar Jonnalagadda, and Srikanth Kamatala. Ai-powered assistive technologies for visual impairment. *arXiv preprint arXiv:2503.15494*, 2025. Preprint.

[10] National Institute on Deafness and Other Communication Disorders. Aphasia: What is aphasia? https://www.nidcd.nih.gov/health/aphasia. Accessed: 2025-07-30.

[11] Alexandre Nevsky, Filip Bircanin, Elena Simperl, Madeline N Cruice, and Timothy Neate. To each their own: Exploring highly personalised audiovisual media accessibility interventions with people with aphasia. In *Proceedings of the 2025 ACM Designing Interactive Systems Conference*, pages 1826–1843, 2025. URL: https://dl.acm.org/doi/full/10.1145/3715336.3735771.

[12] Alexandre Nevsky, Timothy Neate, Elena Simperl, and Radu-Daniel Vatavu. Accessibility research in digital audiovisual media: What has been achieved and what should be done next? In *Proceedings of the 2023 ACM International Conference on Interactive Media Experiences (IMX '23)*. Association for Computing Machinery, 2023. doi:10.1145/3573381.3596159.

[13] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *International conference on machine learning*, pages 28492–28518. PMLR, 2023.

[14] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL: http://jmlr.org/papers/v21/20-074.html.

[15] UK Research and Innovation. Content accessibility (ca11y): Highly individualised digital content for supporting diverse needs. URL: https://gtr.ukri.org/projects?ref=EP%2FX012395%2F1#/tabOverview.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL: https://arxiv.org/abs/1706.03762, arXiv:1706.03762.

A Appendix

A.1 Installing GPU-accelerated version of PyTorch and dependencies

```
pip install torch torchvision torchaudio --index-url https://download.
    pytorch.org/whl/cu118
pip install transformers accelerate bitsandbytes
```

Listing 1: Installs the GPU-accelerated versions

A.2 /upload-thumb API

```
    # Google Cloud Storage setup
    GCS_BUCKET = os.environ['GCS_BUCKET_NAME']
    storage_client = storage.Client()
    bucket = storage_client.bucket(GCS_BUCKET)
```

Listing 2: Google storage bucket

```
@app.route('/upload-thumb', methods=['POST'])
def upload_thumbnail():
    if 'file' not in request.files:
        return jsonify({"error": "No file part"}), 400
    file = request.files['file']
    filename = file.filename  # expected "<id>.png"
    blob = bucket.blob(f'thumbnails/{filename}')
    blob.upload_from_file(file.stream, content_type=file.mimetype)
    thumb_url = blob.public_url
    print(f"[DEBUG] upload_thumbnail succeeded: {thumb_url}")  # Debug
    log
    return jsonify({"thumbUrl": thumb_url}), 200
```

Listing 3: Flask endpoint for uploading a thumbnail to GCS

A.2.1 Expected Request

```
POST /upload-thumb
Content-Type: multipart/form-data
Form field: file=<binary PNG data>
```

Listing 4: Example request to /upload-thumb

A.2.2 Success Response

```
Status: 200 OK
{
  "thumbUrl": "https://storage.googleapis.com/<bucket-name>/thumbnails/
    video123.png"
}
```

Listing 5: Success response from /upload-thumb

A.2.3 Error Response

```
Status: 400 Bad Request
{
  "error": "No file part"
}
```

Listing 6: Error response from /upload-thumb

A.3 /upload-video API

```python
@app.route('/upload-video', methods=['POST'])
def upload_video():
    if 'file' not in request.files:
        return jsonify({"error": "No file part"}), 400
    file = request.files['file']
    filename = file.filename  # expected "<id>.webm"
    blob = bucket.blob(f'videos/{filename}')
    blob.upload_from_file(file.stream, content_type=file.mimetype)
    return jsonify({"videoUrl": blob.public_url}), 200
```

Listing 7: Flask endpoint for uploading a video to GCS

A.3.1 Expected Request

```
POST /upload-video
Content-Type: multipart/form-data
Form field: file=<binary WEBM or MP4 data>
```

Listing 8: Example request to /upload-video

A.3.2 Success Response

```
Status: 200 OK
{
  "videoUrl": "https://storage.googleapis.com/<bucket-name>/videos/
    video123.webm"
}
```

Listing 9: Success response from /upload-video

A.3.3 Error Response

```
Status: 400 Bad Request
{
  "error": "No file part"
}
```

Listing 10: Error response from /upload-video

A.4 /submit-video-to-summarize calls the following function from jobprocessor

```
def submit_video_to_summarize_handler():
    """
    Enqueue a video summarization job by downloading the video,
    extracting audio via ffmpeg, then processing it.
    Expects JSON with 'id', 'title', 'thumbnailUrl', and 'videoUrl'.
    """
    data = request.get_json()
    # Validate payload
    if not data or not all(k in data for k in ("id", "title", "
thumbnailUrl", "videoUrl")):
        return jsonify({"error": "Missing one of id, title, thumbnailUrl,
 videoUrl"}), 400
    job_id = str(uuid.uuid4())
    video_url = data["videoUrl"]
    # Download video to temp file
    video_path = f"/tmp/{job_id}_video"
    resp = requests.get(video_url, stream=True)
    if resp.status_code != 200:
        return jsonify({"error": "Failed to download video"}), resp.
status_code
    with open(video_path, "wb") as vf:
        for chunk in resp.iter_content(chunk_size=8192):
            vf.write(chunk)
    # Extract audio to WAV
    audio_path = f"/tmp/{job_id}.wav"
    ffmpeg.input(video_path).output(
        audio_path, format="wav", acodec="pcm_s16le", ar="16000", ac=1
    ).run(overwrite_output=True)
    os.remove(video_path)
    # Enqueue processing of the audio file
    Thread(target=process_job, args=(audio_path, job_id, data.get("
model_name", "gemini"))).start()
    return jsonify({"job_id": job_id})
```

Listing 11: Flask handler for submitting a video to be summarized

A.4.1 /submit-video-to-summarize API

```
@app.route("/submit-video-to-summarize", methods=["POST"])
def submit_video_to_summarize():
    """
    Enqueue a video summarization job.
    Expects JSON payload with 'id', 'title', 'thumbnailUrl', and '
videoUrl'.
    """
    data = request.get_json()
    print(f"[DEBUG] submit-video-to-summarize called with payload: {data}
")
    # Basic validation
```

```
10      if not data or not all(k in data for k in ("id", "title", "
    thumbnailUrl", "videoUrl")):
11          return jsonify({"error": "Missing one of id, title, thumbnailUrl,
     videoUrl"}), 400
12      # Delegate to job processor
13      return job_processor.submit_video_to_summarize_handler()
```

Listing 12: Flask handler for submitting a video to be summarized

A.4.2 Expected Request

```
1 POST /submit-video-to-summarize
2 Content-Type: application/json
3 {
4   "id": "abc123",
5   "title": "Sample Video",
6   "thumbnailUrl": "https://storage.googleapis.com/bucket/thumbnails/
    abc123.png",
7   "videoUrl": "https://storage.googleapis.com/bucket/videos/abc123.webm",
8   "model_name": "gemini"
9 }
```

Listing 13: Example request to /submit-video-to-summarize

A.4.3 Success Response

```
1 Status: 200 OK
2 {
3   "job_id": "2f1e4c56-17ac-4b3a-88b0-0d7612a92e3f"
4 }
```

Listing 14: Success response from /submit-video-to-summarize

A.4.4 Error Response

```
1 Status: 400 Bad Request
2 {
3   "error": "Missing one of id, title, thumbnailUrl, videoUrl"
4 }
```

Listing 15: Error response from /submit-video-to-summarize

A.5 Job Processor

```
1 import os
2 import uuid
3 from threading import Thread
4 from flask import request, jsonify
5 from transcriber import transcribe_with_whisper
```

```
6 from summarize import summarize_text
7 import redis
8 import hashlib
9
10 import requests
11 import ffmpeg
12
13 r = redis.Redis(host='localhost', port=6379, db=0)
14
15 job_results = {}
16
17 def process_job(file_path, job_id, model_name):
18     try:
19         file_hash = None
20         with open(file_path, "rb") as f:
21             file_hash = hashlib.md5(f.read()).hexdigest()
22         cached_transcript = r.get(f"{file_hash}_transcript")
23         cached_summary = r.get(f"{file_hash}_summary")
24         if cached_summary and cached_transcript:
25             print(f"[DEBUG] Cache hit for job {job_id}")
26             job_results[job_id] = cached_summary.decode()
27             os.remove(file_path)
28             return
29         print(f"[DEBUG] Processing job {job_id} with model: {model_name}"
    )
30         transcript = transcribe_with_whisper(file_path)
31         summary = summarize_text(transcript, model_name)
32         r.set(f"{file_hash}_transcript", transcript, ex=172800)
33         r.set(f"{file_hash}_summary", summary, ex=172800)
34         job_results[job_id] = summary
35         os.remove(file_path)
36     except Exception as e:
37         job_results[job_id] = f"Error: {str(e)}"
38
39 def submit_job_handler():
40     if "file" not in request.files:
41         return jsonify({"error": "No file uploaded"}), 400
42     file = request.files["file"]
43     model_name = request.form.get("model_name", "t5-small")
44     job_id = str(uuid.uuid4())
45     file_path = f"/tmp/{job_id}.mp3"
46     file.save(file_path)
47     Thread(target=process_job, args=(file_path, job_id, model_name)).
    start()
48     return jsonify({"job_id": job_id})
49
50 def job_result_handler(job_id):
51     if job_id in job_results:
52         return jsonify({"summary": job_results[job_id]})
53     return jsonify({"status": "processing"})
54
55 def submit_video_to_summarize_handler():
56     """
```

```
    Enqueue a video summarization job by downloading the video,
    extracting audio via ffmpeg, then processing it.
    Expects JSON with 'id', 'title', 'thumbnailUrl', and 'videoUrl'.
    """
    data = request.get_json()
    # Validate payload
    if not data or not all(k in data for k in ("id", "title", "thumbnailUrl", "videoUrl")):
        return jsonify({"error": "Missing one of id, title, thumbnailUrl, videoUrl"}), 400
    job_id = str(uuid.uuid4())
    video_url = data["videoUrl"]
    # Download video to temp file
    video_path = f"/tmp/{job_id}_video"
    resp = requests.get(video_url, stream=True)
    if resp.status_code != 200:
        return jsonify({"error": "Failed to download video"}), resp.status_code
    with open(video_path, "wb") as vf:
        for chunk in resp.iter_content(chunk_size=8192):
            vf.write(chunk)
    # Extract audio to WAV
    audio_path = f"/tmp/{job_id}.wav"
    ffmpeg.input(video_path).output(
        audio_path, format="wav", acodec="pcm_s16le", ar="16000", ac=1
    ).run(overwrite_output=True)
    os.remove(video_path)
    # Enqueue processing of the audio file
    Thread(target=process_job, args=(audio_path, job_id, data.get("model_name", "gemini"))).start()
    return jsonify({"job_id": job_id})
```

Listing 16: Job processor module for asynchronous transcription and summarization

A.6 Whisper-based transcription

```
import whisper
import os
import torch

# Determine device and load the Whisper model
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"[DEBUG] Loading Whisper model on device: {device}")
model = whisper.load_model("tiny", device=device)  # or "base", "small", etc.

def transcribe_with_whisper(file_path):
    if not os.path.exists(file_path):
        raise FileNotFoundError(f"File not found: {file_path}")

    print(f"[DEBUG] Transcribing file: {file_path}")
    result = model.transcribe(file_path)
```

```
16      print(f"[DEBUG] Whisper raw result object: {result}")
17      transcript = result["text"]
18      print(f"[DEBUG] Transcription complete. Word count: {len(transcript.
        split())}")
19      return transcript
```

Listing 17: Whisper-based transcription function using Hugging Face

A.7 t5-small transformer model

```
1  def summarize_t5_small(text):
2      text = text.strip()
3      words = text.split()
4      chunk_size = 400
5      chunks = [words[i:i + chunk_size] for i in range(0, len(words),
       chunk_size)]
6
7      summaries = []
8      for chunk in chunks:
9          chunk_text = " ".join(chunk)
10         result = summarizer(chunk_text, max_length=130, min_length=30,
       do_sample=False)
11         summaries.append(result[0]['summary_text'])
12
13     final_summary = " ".join(summaries)
14     return final_summary
```

Listing 18: Summarization using t5-small model with chunking

A.8 Summarization using Gemini model

```
1  def summarizer_gemini(text):
2      text = text.strip()
3      from google import genai
4      import json
5      print(f"[DEBUG] About to summarize content using Gemini summarizer...
       ")
6
7      genai_client = genai.Client(
8          vertexai=True,
9          project="secure-garden-460600-u4",
10         location="us-east4"
11     )
12
13     prompt = CHAPTERIZE_PROMPT_TEMPLATE.format(content=text)
14
15     print(f"[DEBUG] Prompt for Gemini summarizer:\n{prompt}")
16
17     chat = genai_client.chats.create(model="gemini-2.0-flash-001")
18     response = chat.send_message(prompt)
19     result = response.text.strip()
```

```
20      # Remove Markdown-style ```json or ``` wrappers if present
21      result = re.sub(r"^```(?:json)?\n", "", result)
22      result = re.sub(r"\n```$", "", result)
23      print(f"[DEBUG] Gemini summarizer response:\n{result}")
24
25      try:
26          json_data = json.loads(result)
27          return json.dumps(json_data, indent=2)
28      except json.JSONDecodeError as e:
29          raise RuntimeError(f"[ERROR] Failed to parse Gemini summary JSON:
        {e}")
```

Listing 19: Summarization using Gemini model via Vertex AI

A.9 MindMap generator

```
1  def generate_mindmap_mistral(summary_text):
2      from transformers import AutoTokenizer, AutoModelForCausalLM,
       pipeline, BitsAndBytesConfig
3      prompt = PROMPT_TEMPLATE.format(summary=summary_text)
4      print(f"[DEBUG] Prompt for Mistral model:\n{prompt}")
5      return
6
7
8  def generate_mindmap_gemini(summary_text):
9      from google import genai
10     from google.genai import types
11
12     genai_client = genai.Client(
13         vertexai=True,
14         project="secure-garden-460600-u4",
15         location="us-east4",
16     )
17
18     prompt = PROMPT_TEMPLATE.format(summary=summary_text)
19     print(f"[DEBUG] Prompt for Gemini GenAI SDK:\n{prompt}")
20
21     chat = genai_client.chats.create(model="gemini-2.0-flash-001")
22     response = chat.send_message(prompt)
23     result = response.text.strip()
24     print(f"[DEBUG] Gemini GenAI SDK response:\n{result}")
25
26     # Updated regex to support deeply nested JSON with narration fields
       in central, branches, and points
27     json_match = re.search(r'(\{.*"central".*"branches".*\}[\s\n]*)',
       result, re.DOTALL)
28     if not json_match:
29         raise ValueError("Mind map JSON not found in Gemini model output.
       ")
30     json_str = json_match.group(1)
31     print(f"[DEBUG] Extracted mind map JSON:\n{json_str}")
32     try:
```

```
33         return json.loads(json_str)
34     except Exception as e:
35         raise RuntimeError(f"[ERROR] Failed to parse Gemini model output:
     {e}")
```

Listing 20: Mind map generation using Gemini and Mistral

A.10 Simple Json Output used to create mindmap


```
1  {
2    "central": {
3      "label": "U+2600 Morning Routine",
4      "narration": "This mind map summarizes the benefits of establishing a
     good morning routine."
5    },
6    "branches": [
7      {
8        "label": "U+1F50A Audio Test & Intro",
9        "narration": "This branch represents the initial audio check and
   brief introduction to the video's theme.",
10       "points": [
11         {
12           "label": "U+1F3A4 Audio Check",
13           "narration": "Confirms the speaker is testing their audio
   before starting the main topic."
14         },
15         {
16           "label": "U+23F1 Brief Topic Intro",
17           "narration": "Introduces the core topic of the video within
   approximately 10 seconds."
18         }
19       ]
20     },
21     {
22       "label": "U+23F0 Morning Work Benefits",
23       "narration": "This section focuses on the advantages of working
   during the morning hours.",
24       "points": [
25         {
26           "label": "U+1F305 Early Wake-Up Impact",
27           "narration": "Discusses the positive effects of waking up early
    and its consistency."
28         },
29         {
30           "label": "U+1F4AA Health Advantages",
31           "narration": "Highlights the significant health benefits
   derived from a morning routine."
32         },
33         {
34           "label": "U+1F501 Atomic Habits",
35           "narration": "Compares the consistency of waking up early to
   the concept of atomic habits."
```

```
        }
      ]
    },
    {
      "label": "U+1F31E Sunlight Exposure",
      "narration": "This branch describes the positive impacts of getting sunlight in the morning.",
      "points": [
        {
          "label": "U+1F60E Good Effects",
          "narration": "Getting morning sunlight has very good effects."
        }
      ]
    },
    {
      "label": "U+1F44B Conclusion",
      "narration": "This section represents the end of the discussion.",
      "points": [
        {
          "label": "U+1F3AC Ending the Video",
          "narration": "Indicates the speaker's intention to conclude the video."
        }
      ]
    }
  ]
}
```

Listing 21: Mind map JSON with emoji labels replaced by Unicode names

A.11 Sample Json for Chaptertized summary

```
[
  {
    "chapterTitle": "Testing Audio and Introduction",
    "startTime": "00:00",
    "chapterSummary": "The speaker starts by stating they are testing their audio. They then transition into providing a brief description of a topic, aiming for around 10 seconds."
  },
  {
    "chapterTitle": "Benefits of Working in the Morning",
    "startTime": "00:04",
    "chapterSummary": "The speaker discusses the benefits of working in the morning. They mention the positive impact of waking up early and doing so consistently, drawing a parallel to the concept of atomic habits. They emphasize the significant benefits on one's health from this practice."
  },
  {
    "chapterTitle": "Sunlight Exposure and Conclusion",
    "startTime": "00:14",
```

```
    "chapterSummary": "The speaker highlights the advantages of getting
    sunlight exposure first thing in the morning, suggesting it has very
    good effects. The speaker then indicates they will end the video."
  }
]
```


Listing 22: Chapter-based summary JSON output

```python
@app.route("/save-summary", methods=["POST"])
def save_summary():
    """
    Save summary entry as JSON under key summary:<id> in Redis.
    Expects JSON with at least 'id' field.
    """
    entry = request.get_json()
    # Do not persist thumbnail in Redis
    # entry.pop("thumbnail", None)
    summary_id = entry.get("id")
    if not summary_id:
        return jsonify({"error": "Missing 'id'"}), 400
    key = f"summary:{summary_id}"
    redis_client.set(key, json.dumps(entry))
    return jsonify({"status": "saved", "id": summary_id}), 201
```

Listing 23: Save summary metadata to Redis

```python
@app.route("/list-summaries", methods=["GET"])
def list_summaries():
    """
    List all saved summary entries from Redis.
    """
    keys = redis_client.keys("summary:*")
    entries = []
    for key in keys:
        data = redis_client.get(key)
        entry = json.loads(data)
        entries.append(entry)
    return jsonify(entries), 200
```

Listing 24: List all saved summaries from Redis

```python
# New route: /download-youtube-and-submit
@app.route("/download-youtube-and-submit", methods=["POST"])
def download_youtube_and_submit():
    """
    Accepts a YouTube URL, validates and downloads the first 7 minutes,
    uploads the video to GCS, and returns metadata for processing.
    """
    data = request.get_json()
    url = data.get("url")

    if not url:
        return jsonify({"error": "No URL provided"}), 400

```

```
    # Validate YouTube URL
    match = re.search(r"(?:v=|youtu\.be/)([a-zA-Z0-9_-]{11})", url)
    if not match:
        return jsonify({"error": "Invalid YouTube URL"}), 400

    video_id = match.group(1)

    try:
        with tempfile.TemporaryDirectory() as tmpdir:
            output_path = os.path.join(tmpdir, f"{video_id}.mp4")
            cookie_path = "youtube_cookies.txt"
            ydl_opts = {
                'format': 'bestvideo[ext=mp4]+bestaudio[ext=m4a]/mp4',
                'outtmpl': output_path,
                'quiet': True,
                'merge_output_format': 'mp4',
                'download_sections': '*00:00:00-00:07:00',  # <--
    download only first 7 minutes
}

            if os.path.exists(cookie_path):
                cloud_logger.info(" Using cookiefile for authentication."
    )
                ydl_opts['cookiefile'] = cookie_path
            else:
                cloud_logger.info(" No cookiefile found. Proceeding
    without cookies.")

            with yt_dlp.YoutubeDL(ydl_opts) as ydl:
                ydl.download([url])

            if not os.path.exists(output_path):
                raise Exception("Video not downloaded")

            # Upload to GCS
            blob = bucket.blob(f'videos/{video_id}.mp4')
            with open(output_path, "rb") as f:
                blob.upload_from_file(f, content_type="video/mp4")

            video_url = blob.public_url

            # Prepare minimal metadata
            payload = {
                "id": video_id,
                "title": f"YouTube_{video_id}",
                "thumbnailUrl": f"https://img.youtube.com/vi/{video_id
    }/0.jpg",
                "videoUrl": video_url,
            }

            return jsonify(payload), 200

    except Exception as e:
```

```
63         cloud_logger.error(f"[ERROR] YouTube download failed: {str(e)}")
64         return jsonify({"error": f"YouTube download failed: {str(e)}"}),
    500
```

Listing 25: api for sending a youtube url to summarize

```
1  gunicorn
2  flask
3  flask-cors
4  git+https://github.com/openai/whisper.git
5  yt-dlp
6  youtube-transcript-api
7  requests
8  torch
9  transformers
10 accelerate
11 bitsandbytes
12 llama-cpp-python
13 huggingface_hub
14 google-cloud-aiplatform
15 redis>=4.5
16 google-cloud-storage
17 # For video processing
18 ffmpeg-python
19 # For GPU acceleration, install with:
20 # pip install llama-cpp-python --extra-index-url=https://jllllll.github.
    io/llama-cpp-python-cuBLAS-wheels/AVX2/cu122
```

Listing 26: Python backend dependencies for video summarization system


```
1  {
2    "id": "1d0c9aba-ce5e-452e-9501-44fe23a806a0",
3    "title": "AquaticCreatures16s.mp4",
4    "thumbnailUrl": "https://storage.googleapis.com/video-summarizer-assets
     /thumbnails/1d0c9aba-ce5e-452e-9501-44fe23a806a0.png",
5    "videoUrl": "https://storage.googleapis.com/video-summarizer-assets/
     videos/1d0c9aba-ce5e-452e-9501-44fe23a806a0.webm",
6    "summaryJson": [
7      {
8        "chapterTitle": "Chapter 1: Humanitarian Connection",
9        "chapterSummary": "This chapter introduces a connection to the
     Samaritans, a group known for providing emotional support. It mentions
      a spokesperson, suggesting a possible commentary or statement related
      to their work. The briefness of the information leaves the nature of
     the connection ambiguous."
10     },
11     {
12       "chapterTitle": "Chapter 2: Athlete Introduction",
13       "chapterSummary": "This chapter introduces Samuel Sanchez,
     identified as a member of the U.S. National Team. The lack of further
     context leaves his sport unspecified, but establishes his status as a
     national-level athlete. The link between this individual and the
     previous chapter is currently unclear based solely on the provided
     text."
```

```
14     }
15   ],
16   "mindmapJson": {
17     "central": {
18       "label": " Summary Overview",
19       "narration": "This mind map represents a summary of two chapters
     with seemingly disparate topics."
20     },
21     "branches": [
22       {
23         "label": " Humanitarian Connection",
24         "narration": "This branch focuses on the first chapter, which
     hints at a connection to humanitarian efforts.",
25         "points": [
26           {
27             "label": " The Samaritans",
28             "narration": "This subpoint mentions the Samaritans, a group
     known for emotional support, suggesting a focus on mental health."
29           },
30           {
31             "label": " Spokesperson",
32             "narration": "This subpoint highlights the presence of a
     spokesperson, indicating a potential statement or commentary."
33           }
34         ]
35       },
36       {
37         "label": " Athlete Introduction",
38         "narration": "This branch describes the second chapter,
     introducing an athlete of national caliber.",
39         "points": [
40           {
41             "label": " Samuel Sanchez",
42             "narration": "This point introduces Samuel Sanchez as a key
     figure in the second chapter."
43           },
44           {
45             "label": " U.S. National Team",
46             "narration": "This point specifies Samuel Sanchez's
     affiliation with the U.S. National Team, indicating his status as a
     national athlete."
47           }
48         ]
49       },
50       {
51         "label": " Unclear Link",
52         "narration": "This branch highlights the ambiguous connection
     between the two chapters.",
53         "points": [
54           {
55             "label": " Disparate Topics",
56             "narration": "This point emphasizes the seemingly unrelated
     nature of the humanitarian group and the athlete."
```

```
          },
          {
            "label": "Missing Context",
            "narration": "This point acknowledges the lack of context to understand the connection between the topics."
          }
        ]
      }
    ]
  }
}
```

Listing 27: Sample cached summary object stored in Redis

A.12 video-summarizer-frontend/pages/index.tsx

```
import { useRouter } from 'next/router';
import { v4 as uuidv4 } from 'uuid';
import { useRef, useState, useEffect } from 'react';
import { useVideoSummarizer } from '../hooks/useVideoSummarizer';
import Link from 'next/link';
import { FLASK_BACKEND_DOWNLOAD_YOUTUBE } from '../lib/config';

export default function Home() {
  const { summaries, summarize } = useVideoSummarizer();
  const fileInputRef = useRef<HTMLInputElement>(null);
  const router = useRouter();

  const [mediaRecorder, setMediaRecorder] = useState<MediaRecorder | null>(null);
  const [recordedChunks, setRecordedChunks] = useState<Blob[]>([]);
  const [isRecording, setIsRecording] = useState(false);
  const [showPreview, setShowPreview] = useState(false);
  const [stream, setStream] = useState<MediaStream | null>(null);

  const handleUploadClick = () => {
    fileInputRef.current?.click();
  };

  const handleFileChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    const file = e.target.files?.[0];
    if (file) {
      const id = uuidv4();
      const url = URL.createObjectURL(file);
      // Navigate to detail page, passing video URL and title via query
      router.push({
        pathname: `/video/${id}`,
        query: { videoUrl: url, title: file.name }
      });
      e.target.value = '';
    }
  };
```

```
36
37  const handleStartRecording = async () => {
38    const mediaStream = await navigator.mediaDevices.getUserMedia({ video
    : true, audio: true });
39    setStream(mediaStream);
40    setShowPreview(true);
41    const recorder = new MediaRecorder(mediaStream);
42    setMediaRecorder(recorder);
43    setRecordedChunks([]);
44    recorder.ondataavailable = event => {
45      if (event.data.size > 0) {
46        setRecordedChunks(prev => [...prev, event.data]);
47      }
48    };
49    recorder.start();
50    setIsRecording(true);
51  };
52
53  const handleStopRecording = () => {
54    mediaRecorder?.stop();
55    stream?.getTracks().forEach(track => track.stop());
56    setIsRecording(false);
57    setShowPreview(false);
58  };
59
60  useEffect(() => {
61    if (!isRecording && recordedChunks.length > 0) {
62      const blob = new Blob(recordedChunks, { type: 'video/webm' });
63      const file = new File([blob], `recording-${Date.now()}.webm`, {
    type: 'video/webm' });
64      const id = uuidv4();
65      const url = URL.createObjectURL(file);
66      router.push({
67        pathname: `/video/${id}`,
68        query: { videoUrl: url, title: file.name }
69      });
70    }
71  }, [isRecording, recordedChunks]);
72
73  return (
74    <div className="bg-black text-white min-h-screen flex flex-col items-
    start justify-start">
75      <header className="w-full bg-gray-900 py-4 text-center aphasia-
    style">
76        <h1 className="text-4xl font-bold font-sans tracking-tight
    leading-tight text-white">
77          ReMindMap <span className="text-gray-300">(Simplifies the
    content)</span>
78        </h1>
79      </header>
80
81
82      <input
```

```
          type="file"
          accept="video/*"
          ref={fileInputRef}
          className="hidden"
          onChange={handleFileChange}
        />

        <main className="aphasia-style p-4 w-full">
          {/* Action cards */}
          <div className="flex justify-center flex-wrap gap-4 mb-6">
            {/* Upload card */}
            <div
              className="w-48 flex flex-col items-center bg-gray-800 p-6
    rounded-lg cursor-pointer aphasia-style"
              onClick={handleUploadClick}
            >
              <span className="text-3xl mb-2">      </span>
              <h3 className="text-lg font-medium">Upload</h3>
              <p className="text-sm text-gray-400">Upload Audio & Video to
    simplify</p>
            </div>
            {/* Record card */}
            <div
              className="w-48 flex flex-col items-center bg-gray-800 p-6
    rounded-lg cursor-pointer aphasia-style"
              onClick={isRecording ? handleStopRecording :
    handleStartRecording}
            >
              <span className="text-3xl mb-2">{isRecording ? '      ' : '
          '}</span>
              <h3 className="text-lg font-medium">{isRecording ? 'Stop' : '
    Record'}</h3>
              <p className="text-sm text-gray-400">{isRecording ? 'Stop
    Recording' : 'Record Video to simplify'}</p>
            </div>
            {/* YouTube URL input card */}
            {/*
            <div className="w-96 flex flex-col items-center bg-gray-800 p-6
     rounded-lg aphasia-style">
              <span className="text-3xl mb-2">      </span>
              <h3 className="text-lg font-medium mb-2">YouTube URL</h3>
              <input
                type="text"
                placeholder="Paste YouTube URL"
                className="text-black w-full p-2 rounded mb-2"
                onKeyDown={async (e) => {
                  if (e.key === 'Enter') {
                    const input = (e.target as HTMLInputElement).value;
                    const isValidYoutubeUrl = /^(https?\:\/\/)?(www\.
    youtube\.com|youtu\.?be)\/.+$/.test(input);
                    if (!isValidYoutubeUrl) {
                      alert('Invalid YouTube URL');
                      return;
```

```
127                 }
128 
129                 try {
130                   const res = await fetch(
    FLASK_BACKEND_DOWNLOAD_YOUTUBE, {
131                     method: 'POST',
132                     headers: { 'Content-Type': 'application/json' },
133                     body: JSON.stringify({ url: input })
134                   });
135                   const data = await res.json();
136                   if (data.success && data.videoUrl && data.title &&
    data.id) {
137                     router.push({
138                       pathname: `/video/${data.id}`,
139                       query: { videoUrl: data.videoUrl, title: data.
    title }
140                     });
141                   } else {
142                     alert('Failed to process video');
143                   }
144                 } catch (err) {
145                   console.error(err);
146                   alert('Error processing YouTube URL');
147                 }
148               }
149             }}
150           />
151           <p className="text-sm text-gray-400 text-center">Press Enter
    to submit. Only first 7 minutes will be used.</p>
152         </div>
153         */}
154       </div>
155       {showPreview && stream && (
156         <div className="fixed top-20 right-4 bg-black border border-
    white p-2 rounded z-50">
157           <video
158             autoPlay
159             muted
160             playsInline
161             ref={video => {
162               if (video && stream) {
163                 video.srcObject = stream;
164               }
165             }}
166             className="w-64 h-36 object-cover rounded"
167           />
168           <p className="text-xs text-center mt-1">Recording Preview</p>
169         </div>
170       )}
171       <h2 className="text-2xl font-semibold aphasia-style mb-4">
    Summarized Videos</h2>
172       {summaries.length === 0 ? (
173         <p className="text-gray-500">No summaries yet.</p>
```

```
174         ) : (
175           <div className="grid grid-cols-2 md:grid-cols-3 lg:grid-cols-4
    gap-4">
176             {summaries.map(s => (
177               <Link
178                 key={s.id}
179                 href={`/video/${s.id}`}
180                 className="flex flex-col items-center space-y-2 bg-gray
    -800 p-4 rounded cursor-pointer"
181               >
182                 <img
183                   src={s.thumbnailUrl}
184                   alt={s.title}
185                   className="w-full h-32 object-cover rounded"
186                 />
187                 <p className="text-sm aphasia-style text-center">{s.title
    }</p>
188               </Link>
189             ))}
190           </div>
191         )}
192       </main>
193       {/* Footer */}
194       <footer className="w-full py-4 text-center aphasia-style text-sm bg
    -gray-900">
195           Dwitee Krishna Panda [MIT license]( If the server is
    inaccessible, it might be in sleep mode to conserve energy. Please
    feel free to reach out at dwitee@gmail.com for access.)
196       </footer>
197     </div>
198   );
199 }
```

Listing 28: index.tsx: Main Frontend Page

A.13 video-summarizer-frontend/pages/video/[id].tsx

```
1 import { useRouter } from 'next/router';
2 import { useEffect, useRef, useState } from 'react';
3 import { Network } from 'vis-network/standalone/umd/vis-network.min.js';
4 import { DataSet } from 'vis-data';
5 import { useVideoSummarizer } from '../../hooks/useVideoSummarizer';
6
7 function getDescendants(nodeId: string, edgesData: DataSet<any>): string
    [] {
8   const descendants: string[] = [];
9   const queue: string[] = [nodeId];
10   while (queue.length) {
11     const current = queue.shift()!;
12     const childrenEdges = edgesData.get({ filter: e => e.from === current
     });
13     childrenEdges.forEach(e => {
```

```
14       descendants.push(e.to);
15       queue.push(e.to);
16     });
17   }
18   return descendants.filter(d => d !== nodeId);
19 }
20
21 export default function VideoDetail() {
22   const router = useRouter();
23   const { id, title: queryTitle } = router.query as { id: string; title?:
       string };
24   // Safely retrieve videoUrl from query (could be array)
25   const rawQueryVideoUrl = Array.isArray(router.query.videoUrl)
26     ? router.query.videoUrl[0]
27     : router.query.videoUrl;
28   const [videoSrc, setVideoSrc] = useState<string>(rawQueryVideoUrl || ''
     );
29   const { videoRef, summaries, summarize } = useVideoSummarizer();
30   const [isSummarizing, setIsSummarizing] = useState(false);
31   const [radialLayout, setRadialLayout] = useState(false);
32   const [narrationEnabled, setNarrationEnabled] = useState(false);
33   const [mapFullscreen, setMapFullscreen] = useState(false);
34   const [narrationRate, setNarrationRate] = useState(0.5);
35   const [narrationVolume, setNarrationVolume] = useState(1);
36   const [narratedEmoji, setNarratedEmoji] = useState<string | null>(null)
     ;
37   const [currentNarrationText, setCurrentNarrationText] = useState<string
     >('');
38   const [fontSize, setFontSize] = useState(34);
39   const summary = summaries.find(s => s.id === id);
40
41   // Once router populates the query, update videoSrc
42   useEffect(() => {
43     if (rawQueryVideoUrl) {
44       setVideoSrc(rawQueryVideoUrl);
45     }
46   }, [rawQueryVideoUrl]);
47
48   const isNew = !!rawQueryVideoUrl && !summary;
49
50   const displayTitle = (queryTitle as string) || summary?.title || '';
51   const visRef = useRef<HTMLDivElement>(null);
52
53   const handleSummarize = async () => {
54     if (!videoSrc) return;
55     setIsSummarizing(true);
56     try {
57       // Debug: log the videoSrc before fetching
58       console.log('Fetching videoSrc:', videoSrc);
59       const blob = await fetch(videoSrc).then(res => res.blob());
60       const file = new File([blob], displayTitle || 'video.mp4', { type:
     blob.type });
61       await summarize(file, displayTitle, id as string);
```

```
62      } finally {
63        setIsSummarizing(false);
64      }
65    };
66
67    // helper to convert "HH:MM:SS" or "MM:SS" into seconds
68    const parseTime = (ts: string) => {
69      const parts = ts.split(':').map(Number);
70      return parts
71        .reverse()
72        .reduce((acc, val, i) =>
73          acc + val * (i === 0 ? 1 : i === 1 ? 60 : 3600),
74        0);
75    };
76
77    useEffect(() => {
78      if (summary?.mindmapJson && visRef.current) {
79        // clear existing canvas so full-screen container can initialize
80        visRef.current.innerHTML = '';
81        // -- Mindmap styling configuration --
82        const centralShape = 'box';
83        const centralColor = 'mediumpurple';
84
85        const branchShape = radialLayout ? 'box' : 'ellipse';
86        const branchColors = radialLayout
87          ? ['#2196f3', '#2196f3', '#2196f3', '#2196f3', '#2196f3']
88          : ['deeppink', 'tomato', 'orange', 'limegreen', 'deepskyblue'];
89
90        const pointShape = radialLayout ? 'box' : 'ellipse';
91        const pointColor = radialLayout ? '#4caf50' : 'lightyellow';
92
93        const nodesArray: Array<{ id: string; label: string; shape: string;
       color: string; baseLabel?: string; hidden?: boolean }> = [
94          {
95            id: 'central',
96            label: summary.mindmapJson.central.label,
97            shape: centralShape,
98            color: centralColor
99          }
100       ];
101       const edgesArray: Array<{ id: string; from: string; to: string;
      hidden?: boolean }> = [];
102
103       summary.mindmapJson.branches.forEach((branch: any, i: number) => {
104         const branchId = `branch_${i}`;
105         nodesArray.push({
106           id: branchId,
107           baseLabel: branch.label,
108           label: radialLayout
109             ? `${branch.label}        `
110             : branch.label,
111           shape: branchShape,
112           color: branchColors[i % branchColors.length]
```

```
113         });
114         edgesArray.push({ id: `central_${branchId}`, from: 'central', to:
      branchId });
115
116         branch.points.forEach((pt: any, j: number) => {
117           const pointId = `${branchId}_point_${j}`;
118           nodesArray.push({
119             id: pointId,
120             label: pt.label,
121             shape: pointShape,
122             color: pointColor,
123             hidden: radialLayout // hide only in radial mode
124           });
125           edgesArray.push({
126             id: `${branchId}_${pointId}`,
127             from: branchId,
128             to: pointId,
129             hidden: radialLayout // hide edge only in radial mode
130           });
131         });
132       });
133
134       const nodesData = new DataSet<{
135         id: string;
136         label: string;
137         shape: string;
138         color: string;
139         hidden?: boolean;
140         baseLabel?: string;
141         // allow pinning by axis
142         fixed?: boolean | { x: boolean; y: boolean };
143         font?: { size: number; face: string; bold: boolean };
144       }>(
145         nodesArray.map(node => ({
146           ...node,
147           font: {
148             size: 16,
149             face: 'arial',
150             bold: false,
151           }
152         }))
153       );
154       const edgesData = new DataSet<{
155         id: string;
156         from: string;
157         to: string;
158         hidden?: boolean;
159       }>(edgesArray);
160
161       // --- Narration (Text-to-Speech) ---
162       const synth = window.speechSynthesis;
163       let cancelled = false;
164
```

```
165     const playNarration = async () => {
166       if (!synth) return;
167
168       // Highlight the node while narrating, update color only (not
    font)
169       const speak = async (nodeId: string, text: string) => {
170         return new Promise<void>((resolve) => {
171           const utterance = new window.SpeechSynthesisUtterance(text);
172           utterance.rate = narrationRate;
173           utterance.volume = narrationVolume;
174           utterance.onstart = () => {
175             if (cancelled) return;
176             // Extract emoji from label (not text) and always set a
    fallback if none
177             const labelText = nodesData.get(nodeId)?.label ?? '';
178             const emojiRegex = /^([\u{1F300}-\u{1F6FF}]|[\u{1F900}-\u{1
    F9FF}]|[\u{2600}-\u{26FF}]|[\u{2700}-\u{27BF}])/u;
179             const matchedEmoji = labelText.match(emojiRegex);
180             const selectedEmoji = matchedEmoji && matchedEmoji.length >
     0 ? matchedEmoji[0] : '          ';
181             setNarratedEmoji(selectedEmoji);
182             setCurrentNarrationText(text);
183             // Set color only
184             nodesData.update({
185               id: nodeId,
186               color: 'gold',
187             });
188           };
189           utterance.onend = () => {
190             if (cancelled) return;
191             // Remove color only
192             nodesData.update({
193               id: nodeId,
194               color: undefined,
195             });
196             setNarratedEmoji(null);
197             setCurrentNarrationText('');
198             resolve();
199           };
200           synth.speak(utterance);
201         });
202       };
203
204       // Start from central
205       const central = summary.mindmapJson.central;
206       if (central.narration && !cancelled) {
207         await speak('central', central.narration);
208       }
209
210       for (let i = 0; i < summary.mindmapJson.branches.length; i++) {
211         if (cancelled) break;
212         const branch = summary.mindmapJson.branches[i];
213         const branchId = `branch_${i}`;
```

```
214           if (branch.narration && !cancelled) {
215             await speak(branchId, branch.narration);
216           }
217           for (let j = 0; j < branch.points.length; j++) {
218             if (cancelled) break;
219             const pt = branch.points[j];
220             const pointId = `${branchId}_point_${j}`;
221             if (pt.narration && !cancelled) {
222               await speak(pointId, pt.narration);
223             }
224           }
225         }
226         setCurrentNarrationText('');
227       };
228 
229       if (narrationEnabled) {
230         playNarration();
231       }
232 
233       // Choose physics settings based on layout
234       const physicsOptions = radialLayout
235         ? { enabled: false }
236         : {
237             barnesHut: {
238               gravitationalConstant: -8000,
239               centralGravity: 0.3,
240               springLength: 95
241             },
242             minVelocity: 0.75
243           };
244 
245       const options = {
246         edges: {
247           color: '#8AB6D6',
248           arrows: {
249             to: {
250               enabled: true,
251               type: 'arrow',
252               scaleFactor: 0.5
253             }
254           },
255           smooth: {
256             enabled: true,
257             type: 'curvedCW',
258             forceDirection: 'horizontal',
259             roundness: 0.2
260           }
261         },
262         physics: physicsOptions,
263         layout: radialLayout
264           ? {
265               hierarchical: {
266                 enabled: true,
```

```
267                 direction: 'LR',
268                 sortMethod: 'directed',
269                 nodeSpacing: 100,
270                 levelSeparation: 150,
271                 treeSpacing: 50
272               },
273               improvedLayout: true
274             }
275           : {
276               improvedLayout: true
277             },
278       };
279       const network = new Network(
280         visRef.current,
281         { nodes: nodesData as any, edges: edgesData as any },
282         options
283       );
284
285       if (radialLayout) {
286         network.on('click', params => {
287           if (params.nodes.length > 0) {
288             // Pin this node's x-axis so it doesn't move horizontally; y-
    axis can move
289             const clickedId = params.nodes[0] as string;
290             nodesData.update({ id: clickedId, fixed: { x: true, y: false
    } });
291
292             // If the central node was clicked, toggle all descendants at
     once
293             if (radialLayout && clickedId === 'central') {
294               const allDesc = getDescendants('central', edgesData);
295               if (allDesc.length) {
296                 const anyVisibleAll = allDesc.some(dId => !nodesData.get(
    dId)?.hidden);
297                 // Toggle all other nodes
298                 allDesc.forEach(dId => {
299                   nodesData.update({ id: dId, hidden: anyVisibleAll });
300                 });
301                 // Toggle all edges from central subtree
302                 const allEdges = edgesData.get({ filter: e => allDesc.
    includes(e.to) });
303                 allEdges.forEach(edge => {
304                   edgesData.update({ id: edge.id, hidden: anyVisibleAll
    });
305                 });
306               }
307               return;
308             }
309             const childEdges = edgesData.get({ filter: e => e.from ===
    clickedId });
310             childEdges.forEach(edge => {
311               const childId = edge.to as string;
312               const nodeItem = nodesData.get(childId);
```

```
313             if (nodeItem) {
314               nodesData.update({ id: childId, hidden: !nodeItem.hidden
    });
315             }
316             edgesData.update({ id: edge.id, hidden: !edge.hidden });
317           });
318           // Update branch icon based on any visible child
319           const branchNode = nodesData.get(clickedId);
320           if (branchNode?.baseLabel) {
321             const children = edgesData.get({ filter: e => e.from ===
    clickedId });
322             const anyVisible = children.some(e => {
323               const child = nodesData.get(e.to as string);
324               return child && !child.hidden;
325             });
326             const newLabel = `${branchNode.baseLabel} ${anyVisible ? '
        ' : '       '}`;
327             nodesData.update({ id: clickedId, label: newLabel });
328           }
329         }
330       });
331     }
332
333     // Add node click narration when narrationEnabled is false
334     if (!narrationEnabled) {
335       network.on('click', async (params) => {
336         const clickedId = params.nodes[0];
337         if (!clickedId) return;
338
339         const node = nodesData.get(clickedId);
340         if (!node) return;
341
342         const findNarration = (id: string): string | null => {
343           if (id === 'central') return summary?.mindmapJson?.central?.
    narration || null;
344           const branchMatch = id.match(/^branch_(\d+)$/);
345           if (branchMatch) {
346             const branchIndex = Number(branchMatch[1]);
347             return summary?.mindmapJson?.branches?.[branchIndex]?.
    narration || null;
348           }
349           const pointMatch = id.match(/^branch_(\d+)_point_(\d+)$/);
350           if (pointMatch) {
351             const [_, branchIndex, pointIndex] = pointMatch.map(Number)
    ;
352             return summary?.mindmapJson?.branches?.[branchIndex]?.
    points?.[pointIndex]?.narration || null;
353           }
354           return null;
355         };
356
357         const text = findNarration(clickedId);
358         if (text) {
```

```
359             // Update the narratedEmoji state based on the clicked node's
      label
360             const labelText = typeof (node as any).label === 'string' ? (
    node as any).label : '';
361             const emojiRegex = /^([\u{1F300}-\u{1F6FF}]|[\u{1F900}-\u{1
    F9FF}]|[\u{2600}-\u{26FF}]|[\u{2700}-\u{27BF}])/u;
362             const matchedEmoji = labelText.match(emojiRegex);
363             const selectedEmoji = matchedEmoji && matchedEmoji.length > 0
     ? matchedEmoji[0] : '          ';
364             setNarratedEmoji(selectedEmoji);
365             setCurrentNarrationText(text);
366 
367             // Set color only for clicked node
368             nodesData.update({
369               id: clickedId,
370               color: 'gold',
371             });
372 
373             const utterance = new window.SpeechSynthesisUtterance(text);
374             utterance.rate = narrationRate;
375             utterance.volume = narrationVolume;
376             utterance.onend = () => {
377               // Remove color only for clicked node
378               nodesData.update({
379                 id: clickedId,
380                 color: undefined,
381               });
382               setNarratedEmoji(null);
383               setCurrentNarrationText('');
384             };
385             window.speechSynthesis.speak(utterance);
386           }
387         });
388       }
389 
390       // Cleanup function to cancel narration if component unmounts or
    narrationEnabled changes
391       return () => {
392         cancelled = true;
393         setCurrentNarrationText('');
394         if (synth?.speaking) {
395           synth.cancel();
396         }
397       };
398     }
399   }, [summary, radialLayout, mapFullscreen, narrationEnabled]);
400 
401   const [scrollY, setScrollY] = useState(0);
402 
403   useEffect(() => {
404     const handleScroll = () => {
405       setScrollY(window.scrollY);
406     };
```

```
407      window.addEventListener('scroll', handleScroll);
408      return () => window.removeEventListener('scroll', handleScroll);
409    }, []);
410
411    if (isNew) {
412      return (
413        <div className="bg-black text-white min-h-screen p-4 aphasia-style"
       >
414          <button
415            onClick={handleSummarize}
416            disabled={isSummarizing}
417            className="mb-4 px-4 py-2 bg-blue-600 rounded"
418          >
419            {isSummarizing ? 'Summarizing...' : 'Summarize'}
420          </button>
421          <div
422            className="sticky-video-wrapper sticky top-0 z-40 bg-black mx-
       auto"
423            style={{
424              width: 'clamp(50%, calc(100% - 4rem), 75%)',
425              transition: 'width 0.3s ease',
426              transform: `scale(${1 - Math.min(scrollY / 1000, 0.33)})`,
427              transformOrigin: 'top center'
428            }}
429          >
430            <video
431              ref={videoRef}
432              src={videoSrc || undefined}
433              controls
434              className="w-full h-auto"
435            />
436          </div>
437        </div>
438      );
439    }
440
441    if (!summary) {
442      return (
443        <div className="bg-black text-white min-h-screen p-4 aphasia-style"
       >
444          <p>Summary not found.</p>
445        </div>
446      );
447    }
448
449    return (
450      <div className="bg-black text-white min-h-screen p-4 aphasia-style">
451        <button
452          onClick={() => router.back()}
453          className="mb-4 px-4 py-2 bg-gray-700 rounded"
454        >
455              Back
456        </button>
```

```
457      <div className="flex flex-col lg:flex-row gap-6">
458        {/* Left column: video & summary */}
459        <div className="flex flex-col flex-shrink-0 w-full lg:w-3/5">
460          <div
461            className="sticky-video-wrapper sticky top-0 z-40 bg-black mx
   -auto mb-4"
462            style={{
463              width: 'clamp(50%, calc(100% - 4rem), 75%)',
464              transition: 'width 0.3s ease',
465              transform: `scale(${1 - Math.min(scrollY / 1000, 0.33)})`,
466              transformOrigin: 'top center'
467            }}
468          >
469            <video
470              ref={videoRef}
471              src={videoSrc || summary.videoUrl || undefined}
472              controls
473              className="w-full h-auto"
474            />
475          </div>
476          {summary && (
477            <div className="fixed bottom-6 right-6 z-50 flex flex-col
   items-end text-center">
478              <img
479                src={`https://api.qrserver.com/v1/create-qr-code/?size
   =120x120&data=${encodeURIComponent(
480                  `${typeof window !== 'undefined' ? window.location.
   origin : ''}/video/${id}?fullscreen=true`
481                )}`}
482                alt="QR Code for Mindmap"
483                className="rounded shadow"
484                title="Scan to view mindmap on your mobile"
485              />
486              <p className="text-sm text-gray-400 mt-1">Scan to view
   mindmap on your mobile</p>
487            </div>
488          )}
489          <div className="bg-gray-800 p-6 rounded-lg mb-4">
490            <h2 className="text-2xl font-semibold mb-2">Summary</h2>
491            {summary.summaryJson ? (
492              <div className="space-y-4 aphasia-style">
493                {summary.summaryJson.map((chapter, idx) => {
494                  // jump video to chapter time on click
495                  const handleJump = () => {
496                    if (videoRef.current && chapter.startTime) {
497                      const seconds = parseTime(chapter.startTime);
498                      videoRef.current.currentTime = seconds;
499                      videoRef.current.play();
500                    }
501                  };
502
503                  return (
504                    <section
```

```
505                         key={idx}
506                         onClick={handleJump}
507                         className="bg-gray-700 p-4 rounded cursor-pointer
    hover:bg-gray-600 transition"
508                       >
509                         <div className="flex items-center mb-1">
510                           {chapter.startTime && (
511                             <span className="text-sm text-gray-400 mr-2">
512                               {chapter.startTime}
513                             </span>
514                           )}
515                           <h3 className="text-xl font-bold">
516                             {chapter.chapterTitle}
517                           </h3>
518                         </div>
519                         <p className="text-base">{chapter.chapterSummary}</
    p>
520                       </section>
521                     );
522                   })}
523                 </div>
524               ) : (
525                 <p className="aphasia-style whitespace-pre-wrap">
526                   {summary.summaryText}
527                 </p>
528               )}
529             </div>
530           </div>
531 
532           {/* Right column: mind map */}
533           {summary.mindmapJson && (
534             <>
535               {mapFullscreen ? (
536                 <div className="fixed inset-0 bg-gray-900 z-50 flex flex-
    col">
537                   <div className="flex justify-between items-center p-4
    border-b border-gray-700">
538                     <h2 className="text-2xl font-semibold text-white">Mind
    Map</h2>
539                     <div className="flex items-center space-x-2">
540                       <button
541                         onClick={() => setRadialLayout(prev => !prev)}
542                         className="px-2 py-1 bg-blue-500 text-white rounded
    "
543                       >
544                         {radialLayout ? 'Standard' : 'Radial'}
545                       </button>
546                       <button
547                         onClick={() => setNarrationEnabled(prev => !prev)}
548                         className="px-2 py-1 bg-green-600 text-white
    rounded"
549                       >
550                         {narrationEnabled ? 'Stop Narration' : 'Narrate'}
```

```
551                     </button>
552                     <select
553                       value={narrationRate}
554                       onChange={(e) => setNarrationRate(Number(e.target.
    value))}
555                       className="px-2 py-1 bg-gray-700 text-white rounded
    "
556                     >
557                       <option value={0.25}>0.25x</option>
558                       <option value={0.5}>0.5x</option>
559                       <option value={1}>1x</option>
560                     </select>
561                     <select
562                       value={fontSize}
563                       onChange={(e) => setFontSize(Number(e.target.value)
    )}
564                       className="px-2 py-1 bg-gray-700 text-white rounded
    "
565                       title="Mindmap Font Size"
566                     >
567                       <option value={20}>20px</option>
568                       <option value={24}>24px</option>
569                       <option value={26}>26px</option>
570                       <option value={30}>30px</option>
571                       <option value={34}>34px</option>
572                       <option value={36}>36px</option>
573                     </select>
574                     <button
575                       onClick={() => setMapFullscreen(false)}
576                       className="text-white text-2xl"
577                     >
578 
579                     </button>
580                   </div>
581                 </div>
582                 <div className="flex-grow relative">
583                   {narratedEmoji && (
584                     <div className="absolute top-4 left-4 text-[10rem] z
    -50">
585                       {narratedEmoji}
586                     </div>
587                   )}
588                   <div ref={visRef} className="h-full w-full" />
589                   {summary && (
590                     <div className="absolute bottom-6 right-6 z-50 flex
    flex-col items-end text-center">
591                       <img
592                         src={`https://api.qrserver.com/v1/create-qr-code
    /?size=120x120&data=${encodeURIComponent(
593                           `${typeof window !== 'undefined' ? window.
    location.origin : ''}/video/${id}?fullscreen=true`
594                         )}`}
595                         alt="QR Code for Mindmap"
```

```
596                          className="rounded shadow"
597                          title="Scan to view mindmap on your mobile"
598                        />
599                        <p className="text-sm text-gray-400 mt-1">Scan to
    view mindmap on your mobile</p>
600                      </div>
601                    )}
602                    {/* Narration text overlay for fullscreen mind map */}
603                    {!!currentNarrationText && (
604                      <div
605                        className="absolute bottom-20 left-4 right-4 bg-
    black bg-opacity-70 text-white font-mono p-3 rounded shadow aphasia-
    style"
606                        style={{ fontSize: `${fontSize}px` }}
607                      >
608                        {currentNarrationText}
609                      </div>
610                    )}
611                  </div>
612                  {/* Volume slider for fullscreen mode, static position
    just below the rate select */}
613                  <div className="px-4 pb-4 pt-2 flex items-center justify-
    end space-x-2">
614                    <span role="img" aria-label="Volume" className="mr-2">
          </span>
615                    <input
616                      type="range"
617                      min="0"
618                      max="1"
619                      step="0.01"
620                      value={narrationVolume}
621                      onChange={(e) => setNarrationVolume(Number(e.target.
    value))}
622                      className="w-32"
623                      title="Narration Volume"
624                    />
625                  </div>
626                </div>
627              ) : (
628                <div className="flex-shrink-0 w-full lg:w-2/5" style={{
    minHeight: "700px" }}>
629                  <div className="sticky top-0 z-30">
630                    <div className="bg-gray-800 p-6 rounded-lg">
631                      <div className="flex justify-between items-center mb
    -2">
632                        <h2 className="text-2xl font-semibold">Mind Map</h2
    >
633                        <button
634                          onClick={() => setMapFullscreen(true)}
635                          className="text-white text-xl"
636                        >
637
638                        </button>
```

```
639                     </div>
640                     <div className="text-right mb-2 space-x-2">
641                       <button
642                         onClick={() => setRadialLayout(prev => !prev)}
643                         className="px-2 py-1 bg-blue-500 text-white
    rounded"
644                       >
645                         {radialLayout ? 'Standard' : 'Radial'}
646                       </button>
647                       <button
648                         onClick={() => setNarrationEnabled(prev => !prev)
    }
649                         className="px-2 py-1 bg-green-600 text-white
    rounded"
650                       >
651                         {narrationEnabled ? 'Stop Narration' : 'Narrate'}
652                       </button>
653                       <select
654                         value={narrationRate}
655                         onChange={(e) => setNarrationRate(Number(e.target
    .value))}
656                         className="px-2 py-1 bg-gray-700 text-white
    rounded"
657                       >
658                         <option value={0.25}>0.25x</option>
659                         <option value={0.5}>0.5x</option>
660                         <option value={1}>1x</option>
661                       </select>
662                     </div>
663                     {/* Volume slider just below the layout buttons and
    rate selector */}
664                     <div className="mt-2 flex items-center justify-end
    space-x-2">
665                       <span role="img" aria-label="Volume" className="mr
    -2">      </span>
666                       <input
667                         type="range"
668                         min="0"
669                         max="1"
670                         step="0.01"
671                         value={narrationVolume}
672                         onChange={(e) => setNarrationVolume(Number(e.
    target.value))}
673                         className="w-32"
674                         title="Narration Volume"
675                       />
676                     </div>
677                     <div className="relative">
678                       {narratedEmoji && (
679                         <div className="absolute top-4 left-4 text-7xl z
    -50">
680                           {narratedEmoji}
681                         </div>
```

```
682                     )}
683                     <div ref={visRef} style={{ height: '600px', width:
    '100%' }} />
684                     {/* Narration text display */}
685                     {!!currentNarrationText && (
686                       <div
687                         className="mt-4 p-2 bg-black bg-opacity-70
    rounded text-white font-mono aphasia-style"
688                         style={{ fontSize: `${fontSize}px` }}
689                       >
690                         {currentNarrationText}
691                       </div>
692                     )}
693                     {/* Volume control moved above, removed absolute
    block */}
694                   </div>
695                 </div>
696               </div>
697             </div>
698           )}
699         </>
700       )}
701     </div>
702   </div>
703   );
704 }
```

Listing 29: id.tsx: Dynamic Video Summary Page

A.14 video-summarizer-frontend/lib/config.ts

```
1  export const MIND_MAP_PATH = `/api/mindmap/`;
2  export const FLASK_JOB_SUBMIT        = `/api/submit-job`;
3  export const FLASK_JOB_RESULT        = `/api/job-result/`;
4  export const FLASK_BACKEND_UPLOAD    = `/api/summarize-upload`;
5  export const FLASK_BACKEND_UPLOAD_THUMBNAIL      = `/api/upload-thumb`;
6  export const FLASK_BACKEND_UPLOAD_VIDEO          = `/api/upload-video`;
7  export const FLASK_BACKEND_SUBMIT_VIDEO         = `/api/submit-video-to-
    summarize`;
8  export const FLASK_BACKEND_GENERATE_MINDMAP = `/api/generate-mindmap`;
9  export const FLASK_BACKEND_UPLOAD_MINDMAP   = `/api/upload-mindmap`;
10
11 export const FLASK_BACKEND_SAVE_SUMMARY    = `/api/save-summary`;
12 export const FLASK_BACKEND_LIST_SUMMARIES = `/api/list-summaries`;
13 export const FLASK_BACKEND_DOWNLOAD_YOUTUBE = `/api/download-youtube-and-
    submit`;
```

Listing 30: config.ts for api endpoints

A.15 video-summarizer-frontend/lib/summarizer.ts

```
// lib/summarizer.ts
import { createFFmpeg, fetchFile } from '@ffmpeg/ffmpeg';
import {
  FLASK_BACKEND_UPLOAD,
  FLASK_BACKEND_GENERATE_MINDMAP,
  FLASK_BACKEND_UPLOAD_THUMBNAIL,
  FLASK_BACKEND_UPLOAD_VIDEO,
} from './config';

export async function captureThumbnail(videoEl: HTMLVideoElement): Promise<string> {
  console.log('[DEBUG] captureThumbnail: starting thumbnail capture');
  await new Promise<void>(resolve => {
    if (videoEl.readyState >= 2) resolve();
    else videoEl.addEventListener('loadeddata', () => resolve(), { once: true });
  });
  console.log('[DEBUG] captureThumbnail: video data loaded, capturing frame');
  const canvas = document.createElement('canvas');
  canvas.width = videoEl.videoWidth;
  canvas.height = videoEl.videoHeight;
  canvas.getContext('2d')!.drawImage(videoEl, 0, 0, canvas.width, canvas.height);
  const thumbnail = canvas.toDataURL('image/png');
  console.log(`[DEBUG] captureThumbnail: thumbnail data-length ${thumbnail.length}`);
  return thumbnail;
}

export async function submitForSummarization(
  audio: Blob,
  filename: string
): Promise<string> {
  console.log(`[DEBUG] submitForSummarization: preparing FormData for file "${filename}"`);
  const form = new FormData();
  form.append('file', audio, filename.replace(/\.[^/.]+$/, '.mp3'));
  form.append('model_type', 'gemini');
  console.log('[DEBUG] submitForSummarization: sending request to', FLASK_BACKEND_UPLOAD);
  const res = await fetch(FLASK_BACKEND_UPLOAD, { method: 'POST', body: form });
  console.log('[DEBUG] submitForSummarization: received response, status', res.status);
  const json = await res.json();
  console.log('[DEBUG] submitForSummarization: summary returned:', json.summary);
  return json.summary ?? 'No summary returned';
}

/**
 * Sends the summarized text to the backend to generate a mind map.
```

```
44  * @param summary The summary text.
45  * @param modelType The model type to use for mind map generation.
46  * @returns The mindmap JSON object.
47  */
48 export async function generateMindmapFromSummary(
49   summary: string,
50   modelType: string
51 ): Promise<any> {
52   console.log(`[DEBUG] generateMindmapFromSummary: sending summary with
     modelType ${modelType}`);
53   try {
54     const res = await fetch(FLASK_BACKEND_GENERATE_MINDMAP, {
55       method: 'POST',
56       headers: { 'Content-Type': 'application/json' },
57       body: JSON.stringify({ summary, model_type: modelType })
58     });
59     if (!res.ok) {
60       throw new Error(`Mindmap request failed: ${res.status}`);
61     }
62     const result = await res.json();
63     console.log('[DEBUG] generateMindmapFromSummary: mind map received',
     result);
64     return result.mindmap ?? result;
65   } catch (e: any) {
66     console.error('[DEBUG] generateMindmapFromSummary error:', e);
67     throw e;
68   }
69 }
70
71 /**
72  * Uploads a thumbnail Blob to your backend storage and returns its
     public URL.
73  */
74 export async function uploadThumbnail(id: string, blob: Blob): Promise<
     string> {
75   const form = new FormData();
76   form.append('file', blob, `${id}.png`);
77   const res = await fetch(FLASK_BACKEND_UPLOAD_THUMBNAIL, {
78     method: 'POST',
79     body: form,
80   });
81   if (!res.ok) {
82     const text = await res.text();
83     throw new Error(`Thumbnail upload failed: ${res.status} ${res.
     statusText}: ${text}`);
84   }
85   const { thumbUrl } = await res.json();
86   return thumbUrl;
87 }
88
89 /**
90  * Uploads a video File to your backend storage and returns its public
     URL.
```

```
91  */
92 export async function uploadVideoFile(id: string, file: File): Promise<
     string> {
93   const form = new FormData();
94   form.append('file', file, `${id}.webm`);
95   const res = await fetch(FLASK_BACKEND_UPLOAD_VIDEO, {
96     method: 'POST',
97     body: form,
98   });
99   if (!res.ok) {
100    const text = await res.text();
101    throw new Error(`Video upload failed: ${res.status} ${res.statusText
     }: ${text}`);
102  }
103  const { videoUrl } = await res.json();
104  return videoUrl;
105 }
```

Listing 31: summarizer.ts for helper methods

A.16 video-summarizer-frontend/hook/useVideoSummarizer.ts

```
1 // hooks/useVideoSummarizer.ts
2 import { useState, useRef, useEffect } from 'react';
3 import { captureThumbnail, generateMindmapFromSummary, uploadThumbnail,
     uploadVideoFile } from '../lib/summarizer';
4 import { v4 as uuidv4 } from 'uuid';
5 import { FLASK_JOB_SUBMIT, FLASK_JOB_RESULT } from '../lib/config';
6 import { FLASK_BACKEND_SAVE_SUMMARY, FLASK_BACKEND_LIST_SUMMARIES } from
     '../lib/config';
7 import { FLASK_BACKEND_SUBMIT_VIDEO } from '../lib/config';
8
9 // Supported model types for summarization
10 export enum ModelType {
11   T5Small = 't5-small',
12   Gemini = 'gemini'
13 }
14
15 // Helper: submit MP3 job, return job ID
16 async function submitAudioJob(
17   audio: Blob,
18   filename: string,
19   modelType: ModelType = ModelType.T5Small
20 ): Promise<string> {
21   console.log('[DEBUG] submitAudioJob: URL=', FLASK_JOB_SUBMIT, 'filename
     =', filename, 'modelType=', modelType);
22   const form = new FormData();
23   form.append('file', audio, filename);
24   form.append('model_name', modelType);
25   let res: Response;
26   try {
27     res = await fetch(FLASK_JOB_SUBMIT, { method: 'POST', body: form });
```

```
28      console.log('[DEBUG] submitAudioJob: HTTP status', res.status);
29    } catch (networkError) {
30      console.error('[DEBUG] submitAudioJob: network error', networkError);
31      throw networkError;
32    }
33    if (!res.ok) throw new Error(`Submit job failed: ${res.status}`);
34    const json = await res.json();
35    return json.job_id;
36  }
37
38  /**
39   * Send video metadata (id, title, thumbnailUrl, videoUrl) to backend.
40   */
41  async function submitVideoJob(payload: {
42    id: string;
43    title: string;
44    thumbnailUrl: string;
45    videoUrl: string;
46  }): Promise<string> {
47    console.log('[DEBUG] submitVideoJob: URL=', FLASK_BACKEND_SUBMIT_VIDEO,
        'payload=', payload);
48    const res = await fetch(FLASK_BACKEND_SUBMIT_VIDEO, {
49      method: 'POST',
50      headers: { 'Content-Type': 'application/json' },
51      body: JSON.stringify(payload),
52    });
53    console.log('[DEBUG] submitVideoJob: status', res.status);
54    if (!res.ok) {
55      const text = await res.text();
56      throw new Error(`Submit video failed: ${res.status} ${res.statusText
      }: ${text}`);
57    }
58    const json = await res.json();
59    console.log('[DEBUG] submitVideoJob: received job_id', json.job_id);
60    return json.job_id;
61  }
62
63  // Helper: poll job result until summary is ready
64  async function pollJobResult(jobId: string, maxWait = 600000, interval =
      10000): Promise<string> {
65    const start = Date.now();
66    while (Date.now() - start < maxWait) {
67      await new Promise(r => setTimeout(r, interval));
68      console.log('[DEBUG] pollJobResult: polling URL=', `${
      FLASK_JOB_RESULT}${jobId}`);
69      const res = await fetch(`${FLASK_JOB_RESULT}${jobId}`);
70      console.log('[DEBUG] pollJobResult: status', res.status);
71      if (!res.ok) throw new Error(`Poll job failed: ${res.status}`);
72      const json = await res.json();
73      if (json.summary) return json.summary;
74      if (json.status !== 'processing') return `Unexpected response: ${JSON
      .stringify(json)}`;
75    }
```

```
76   return 'Error: Summary not available in time.';
77 }
78 
79 // Represents each summarized video entry
80 interface SummaryEntry {
81   id: string;
82   title: string;
83   thumbnailUrl: string;
84   videoUrl: string;
85   summaryText?: string;
86   summaryJson?: Array<{
87     chapterTitle: string;
88     startTime?: string;
89     chapterSummary: string;
90   }>;
91   mindmapJson?: any;
92 }
93 
94 export function useVideoSummarizer() {
95   const videoRef = useRef<HTMLVideoElement>(null);
96   const [summaries, setSummaries] = useState<SummaryEntry[]>([]);
97 
98   // On hook initialization, fetch any previously saved summaries
99   useEffect(() => {
100    (async () => {
101      try {
102        console.log('[DEBUG] loadSummaries: fetching saved entries');
103        const res = await fetch(FLASK_BACKEND_LIST_SUMMARIES);
104        const text = await res.text();
105        if (!res.ok) {
106          console.error('[DEBUG] loadSummaries error status:', res.status
    , text);
107          setSummaries([]);
108          return;
109        }
110        let saved: SummaryEntry[];
111        try {
112          saved = JSON.parse(text) as SummaryEntry[];
113        } catch (jsonErr) {
114          console.error('[DEBUG] loadSummaries: invalid JSON', text,
    jsonErr);
115          saved = [];
116        }
117        console.log('[DEBUG] loadSummaries: received', saved);
118        setSummaries(saved);
119      } catch (e) {
120        console.error('[DEBUG] loadSummaries error:', e);
121      }
122    })();
123  }, []);
124 
125  const summarize = async (
126    file: File,
```

```
127     title: string,
128     idParam?: string,
129     modelType: ModelType = ModelType.Gemini
130   ) => {
131     const id = idParam ?? uuidv4();
132     let thumbnailUrl = '';
133     if (videoRef.current) {
134       console.log('[DEBUG] capturing thumbnail');
135       const thumbnailDataUrl = await captureThumbnail(videoRef.current);
136       const thumbBlob = await (await fetch(thumbnailDataUrl)).blob();
137       thumbnailUrl = await uploadThumbnail(id, thumbBlob);
138       console.log('[DEBUG] uploadThumbnail succeeded:', thumbnailUrl);
139     } else {
140       console.log('[DEBUG] videoRef.current not available, skipping
    thumbnail capture');
141     }
142     const videoUrl = await uploadVideoFile(id, file);
143     console.log('[DEBUG] uploadVideoFile succeeded:', videoUrl);
144     console.log('[DEBUG] submitVideoJob payload:', { id, title,
    thumbnailUrl, videoUrl });
145     const videoJobId = await submitVideoJob({ id, title, thumbnailUrl,
    videoUrl });
146
147     // Local holders for final metadata
148     let finalSummaryText: string = 'Generating...';
149     let finalSummaryJson: Array<{ chapterTitle: string; chapterSummary:
    string }> | undefined = undefined;
150     let finalMindmapJson: any = undefined;
151
152     // Optimistically add placeholder to UI
153     setSummaries(prev => [
154       ...prev,
155       { id, title, thumbnailUrl, videoUrl, summaryText: finalSummaryText
    }
156     ]);
157
158     // Poll the video processing job for summary text
159     try {
160       console.log('[DEBUG] summarize: polling result for video job',
    videoJobId);
161       const raw = await pollJobResult(videoJobId);
162       console.log('[DEBUG] useVideoSummarizer: received video summary',
    raw);
163       // Format the raw summary just like we did for audio
164       let formatted: string;
165       if (modelType === ModelType.Gemini) {
166         // parse JSON for Gemini style summary
167         const jsonData = JSON.parse(raw) as Array<{ chapterTitle: string;
     chapterSummary: string }>;
168         const summaryTextString = jsonData
169           .map(item => `${item.chapterTitle}: ${item.chapterSummary.trim
    ()}`)
170           .join('\n\n');
```

```
171         finalSummaryText = summaryTextString;
172         finalSummaryJson = jsonData;
173         setSummaries(prev =>
174           prev.map(e =>
175             e.id === id
176               ? { ...e, summaryText: summaryTextString, summaryJson:
    jsonData }
177               : e
178           )
179         );
180         // generate mindmap
181         try {
182           console.log('[DEBUG] summarize: generating mindmap for id', id)
    ;
183           const mindmap = await generateMindmapFromSummary(
    summaryTextString, modelType);
184           finalMindmapJson = mindmap;
185           setSummaries(prev =>
186             prev.map(e =>
187               e.id === id ? { ...e, mindmapJson: mindmap } : e
188             )
189           );
190         } catch (e) {
191           console.error('[DEBUG] generateMindmapFromSummary error:', e);
192         }
193       } else {
194         formatted = raw
195           .split('. ')
196           .map((line: string) => `    ${line.trim()}`)
197           .join('\n');
198         finalSummaryText = formatted;
199         finalSummaryJson = undefined;
200         setSummaries(prev =>
201           prev.map(e =>
202             e.id === id ? { ...e, summaryText: formatted, summaryJson:
    undefined } : e
203           )
204         );
205         // generate mindmap
206         try {
207           console.log('[DEBUG] summarize: generating mindmap for id', id)
    ;
208           const mindmap = await generateMindmapFromSummary(formatted,
    modelType);
209           finalMindmapJson = mindmap;
210           setSummaries(prev =>
211             prev.map(e =>
212               e.id === id ? { ...e, mindmapJson: mindmap } : e
213             )
214           );
215         } catch (e) {
216           console.error('[DEBUG] generateMindmapFromSummary error:', e);
217         }
```

```
218         }
219       } catch (err) {
220         console.error('[DEBUG] useVideoSummarizer: error', err);
221         setSummaries(prev =>
222           prev.map(e => (e.id === id ? { ...e, summaryText: 'Error
    generating summary' } : e))
223         );
224       }
225 
226       // Persist full metadata after summary and mindmap are ready
227       try {
228         const fullEntry = {
229           id,
230           title,
231           thumbnailUrl,
232           videoUrl,
233           summaryText: finalSummaryText,
234           summaryJson: finalSummaryJson,
235           mindmapJson: finalMindmapJson,
236         };
237         console.log('[DEBUG] saveSummary: saving full entry', fullEntry);
238         const saveRes = await fetch(FLASK_BACKEND_SAVE_SUMMARY, {
239           method: 'POST',
240           headers: { 'Content-Type': 'application/json' },
241           body: JSON.stringify(fullEntry),
242         });
243         if (!saveRes.ok) throw new Error(`Save full entry failed: ${saveRes
    .status}`);
244         console.log('[DEBUG] saveSummary: full entry saved successfully for
     id', id);
245       } catch (saveErr) {
246         console.error('[DEBUG] saveSummary full entry error:', saveErr);
247       }
248     };
249 
250     return { videoRef, summaries, summarize };
251 }
```

Listing 32: .tsx]useVideoSummarizer.ts, all methods are called by [id].tsx

A.17 video-summarizer-frontend/package.json

```
1  {
2    "name": "video-summarizer-frontend",
3    "version": "1.0.0",
4    "scripts": {
5      "dev": "next dev",
6      "build": "next build",
7      "start": "next start",
8      "postinstall": "npx tailwindcss init -p"
9    },
10   "dependencies": {
```

```
11      "@ffmpeg/ffmpeg": "0.10.1",
12      "@ffmpeg/core": "0.10.0",
13      "next": "latest",
14      "react": "latest",
15      "react-dom": "latest",
16      "uuid": "11.1.0",
17      "vis-network": "^9.1.2",
18      "vis-data": "^7.1.10"
19    },
20    "devDependencies": {
21      "@types/node": "24.0.14",
22      "@types/react": "^18.2.14",
23      "@types/uuid": "^9.0.0",
24      "typescript": "^5.0.0",
25      "tailwindcss": "^3.0.0",
26      "postcss": "^8.4.0",
27      "autoprefixer": "^10.4.0"
28    }
29  }
```

Listing 33: package.json, all libraries

A.18 video-summarizer-frontend/global.css

```
1
2  @tailwind base;
3  @tailwind components;
4  @tailwind utilities;
5
6  body {
7    @apply bg-black text-white;
8    margin: 0;
9    padding: 0;
10   font-family: Arial, Calibri, sans-serif;
11   font-size: 18px;      /* Between 14pt (18.7px) and 18pt (24px) */
12   line-height: 1.6;     /* Ample line spacing */
13 }
14
15 /* Aphasia-friendly spacing and sizing */
16 p, li, h1, h2, h3, label {
17   margin: 1em 0;        /* Plenty of space around text */
18 }
19
20 input, button {
21   font-size: 18px;      /* Ensure controls match text size */
22   padding: 0.5em 1em;   /* Space inside buttons/inputs */
23   margin: 0.5em 0;      /* Space around controls */
24 }
25
26 /* Aphasia Style */
27 .aphasia-style {
28   font-family: Arial, Calibri, sans-serif;
```

```
29    font-size: 18px;        /* Equivalent to 14 18pt   */
30    line-height: 1.6;       /* Ample line spacing */
31  }
```

Listing 34: global.css for aphasia specific style

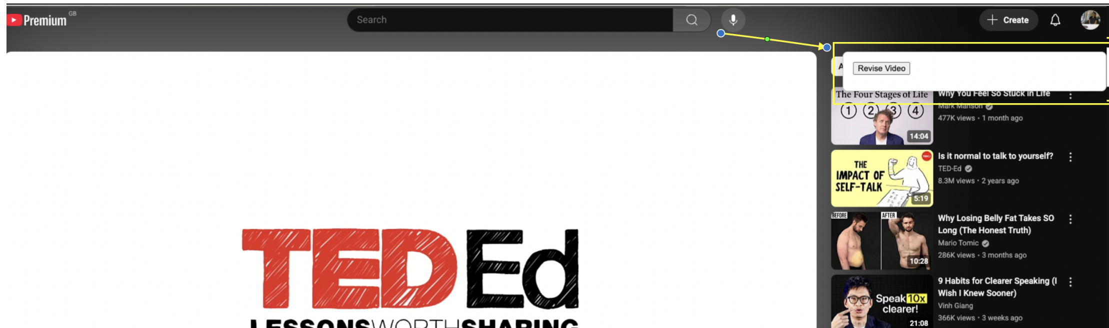

Figure 19: POC of Chrome plugin , check the yellow box

Figure 20: POC of Chrome plugin summary, check the yellow box

Figure 21: Deployment Pipeline using Vercel connected to github repo