Project S2: YouTube Lecture Summarizer

202305013

Gor Babayan

UFAR

Course: Project S2

Instructor: Karen Petrosyan

May 02, 2025

Contents

Summary

This project is a Python-based AI tool designed to summarize YouTube lectures. The objective was to reduce the time users spend on educational videos by providing concise and accurate summaries. The tool combines video processing, speech-to-text transcription, and natural language processing. The current status is a working prototype capable of downloading a YouTube video, transcribing it using Whisper, and summarizing the transcript using Hugging Face models.

Introduction

Project background and context

With the rapid growth of online education, YouTube has become a major platform for delivering academic lectures, tutorials, and expert discussions. However, the long duration of many videos presents a challenge for students, researchers, and professionals who need to quickly extract key information without watching an entire lecture. This issue is further complicated by inconsistent video structures and the lack of accompanying textual summaries or transcripts.

The increasing demand for accessible and efficient learning resources has highlighted the need for tools that can convert long-form video content into concise, readable formats. While transcription services exist, they often lack the ability to summarize or prioritize information contextually. Moreover, many available tools are either locked behind paywalls or offer limited customization.

This project, *YouTube Lecture Summarizer*, addresses that gap by combining cutting-edge speech recognition (via OpenAI's Whisper model) with natural language processing (using Hugging Face Transformers) to automatically generate lecture summaries. It provides a streamlined interface where users can input a YouTube URL and receive a text-based summary, making educational content more accessible, especially for time-constrained learners.

By bridging speech-to-text technology with AI summarization, the project aims to enhance how users interact with video-based content, turning lengthy educational videos into actionable knowledge with minimal effort.

Problem statement and objectives

Problem Statement

In the age of digital learning, YouTube has become a widely used platform for accessing free educational content. However, the vast length and unstructured nature of many lectures present a barrier to efficient learning. Users often find it time-consuming and impractical to watch entire videos to extract specific insights or relevant concepts. This becomes particularly problematic for students and professionals who are balancing multiple commitments and need rapid access to key information.

Existing solutions either focus solely on transcription or offer summarization as a premium feature, often lacking integration, contextual accuracy, or user accessibility. There is a clear need for a free, AI-driven system that can automatically process educational videos and output meaningful summaries in a streamlined and user-friendly manner.

Objectives

The primary goal of this project is to build a functional, AI-powered tool that automatically summarizes YouTube lectures through a simple user interface. The specific objectives are:

1. **Develop a platform** where users can input a YouTube lecture URL and receive a concise summary without watching the entire video.
2. **Integrate OpenAI's Whisper** to transcribe the audio from YouTube videos into accurate, readable text.
3. **Use transformer-based models** from Hugging Face to summarize long transcripts into digestible formats while preserving contextual relevance.
4. **Design a user-friendly interface** using Streamlit to ensure accessibility for users with minimal technical background.
5. **Ensure ADA compliance** by providing text-only output, keyboard navigation, and screen reader compatibility.
6. **Maintain modular, open-source architecture** for ease of extension, such as adding multilingual support or export options in the future.

Together, these objectives aim to simplify how users consume long-form video lectures, transforming them into efficient, accessible learning experiences.

Description of target audience and end users

The *YouTube Lecture Summarizer* is designed to serve a broad audience of learners and professionals who rely on video-based content for education, research, and upskilling. The tool addresses the needs of individuals who value efficiency, accessibility, and clarity when engaging with long-form educational videos.

1. Students and Academic Learners

Undergraduate and graduate students often turn to YouTube for supplemental explanations, tutorials, and recorded lectures. These users benefit from quick summaries that help them preview content before viewing, revise key topics, or identify specific sections worth deeper study.

2. Working Professionals and Lifelong Learners

Professionals pursuing continuous education—especially in fields like programming, data science, finance, and marketing—frequently use online video lectures to keep up with trends or learn new skills. The summarizer saves time by distilling lectures into actionable insights, enabling more efficient learning during limited breaks or commutes.

3. Researchers and Analysts

Academic researchers and content reviewers often analyze large volumes of lectures or webinars. This tool allows them to quickly assess the relevance of a video's content without committing to the full duration, thereby improving research workflows.

4. Educators and Instructional Designers

Teachers, tutors, and course creators can use the summarizer to quickly generate text summaries of educational videos they plan to integrate into course materials. These summaries can serve as discussion guides, handouts, or accessibility aids for students.

5. Individuals with Accessibility Needs

The tool's ADA-compliant, text-based output is especially valuable for users who are hearing impaired, have learning disabilities, or prefer written content over auditory input. It ensures equal access to educational material in a readable, screen-reader-friendly format.

By catering to these diverse user groups, the *YouTube Lecture Summarizer* enhances how knowledge is consumed, making long educational videos more approachable, inclusive, and time-efficient.

Outline of report structure

This report is organized into 13 comprehensive sections that guide the reader through the conceptual, technical, and reflective aspects of the project:

1. **Title Page:** Project Title, Student's Name, Student Number, Teacher's Name, Date of Submission (May 02, 2025), University Name.

2. **Summary**

   Provides a high-level overview of the project's goals, significance, and the current status of development.

3. **Introduction**

   Introduces the context and motivation behind the project. Explains the growing challenge of consuming long-form video content and highlights the need for AI-driven summarization tools.

4. **Team Structure and Individual Contributions**

   Since this is an individual project, this section outlines the full scope of responsibilities managed by the developer, including design, development, testing, and documentation.

5. **Methodology**

   Describes the development process, including the use of Agile principles. It also details the selection of technologies (Whisper, Hugging Face, Streamlit, etc.) and addresses ADA compliance considerations.

6. **Research and Analysis**

   Explores related systems and technologies. Compares the summarizer with existing tools like Otter.ai, Notta, and Jobscan. Discusses the challenges and insights gained during the research phase.

7. **Work in Progress**

   Outlines all completed tasks and features, implementation details, and references to the GitHub repository. Screenshots and summaries of critical code components are also included.

8. **Lessons Learned**

   Reflects on key insights, development challenges, and problem-solving approaches. It also highlights skills gained throughout the project.

9. **Final Deliverables and Achievements**

   Summarizes the completed prototype, its components, and step-by-step usage instructions for end users.

10. **Critical Self-Reflection**

    Provides a reflective evaluation of what went well, areas for improvement, and an overview of both technical and personal growth during the project.

11. **Future Work**

    Identifies unresolved issues and suggests enhancements for future development. This includes multilingual support, real-time summarization, and integrations with learning platforms.

12. **References**

    Lists all external resources, tools, research articles, and documentation consulted during development in APA 7 format.

13. **Appendices**

    Includes supporting documents such as the GitHub repository link, screenshots of the user interface, ADA compliance checklist, and testing notes.

Methodology

Project planning and development process

The *YouTube Lecture Summarizer* project followed an **Agile, iterative development approach**, structured around weekly milestones and continuous refinement. Each stage focused on delivering specific features and validating their performance through testing and feedback.

Week 1 – Research and Requirements Analysis

- Identified the core problem: the inefficiency of consuming long-form educational videos on platforms like YouTube.
- Defined the key features of the system: YouTube video/audio download, transcription, summarization, and user-friendly interface.
- Researched state-of-the-art tools including OpenAI Whisper and Hugging Face Transformers for transcription and summarization.

Week 2 – Initial Prototype and Interface Design

- Used `pytube` to extract audio streams from YouTube videos.
- Built a minimal Streamlit interface allowing users to input a video URL and receive AI-generated summaries.

- Focused on clean, accessible UI design with clear input fields and minimal user friction.

Week 3 – AI Integration and Summarization Logic

- Integrated OpenAI Whisper to convert audio to text with high accuracy.
- Implemented Hugging Face's `distilbart-cnn-12-6` model for summarizing long transcripts, using chunking to overcome token limitations.
- Tuned prompt formats and chunk sizes to optimize summarization output quality.

Week 4 – Testing, Optimization, and Deployment

- Conducted end-to-end tests on a range of YouTube lectures to verify the quality and robustness of transcription and summaries.
- Implemented error handling for invalid URLs, network failures, and missing audio streams.
- Added progress indicators in the UI using Streamlit spinners to enhance user feedback.

Cloud-Based Development and Remote Access

Development was carried out primarily in **Google Colab**, which offered a cloud-based Python environment with free GPU access. This setup enabled faster Whisper transcription and simplified dependency management.

To facilitate real-time testing and demonstrations without deploying to a production server, **pyngrok** was used to create a secure public tunnel to the locally hosted Streamlit app. This allowed the application to be shared externally during development and testing, enabling feedback from other users and devices.

This structured, modular approach ensured that each component was validated before integration, resulting in a fully functional and accessible lecture summarization tool by the end of the development cycle.

Tools and technologies used

To build the *YouTube Lecture Summarizer*, a set of modern and efficient tools were selected based on their open-source accessibility, cloud compatibility, and ease of integration. These technologies were critical for developing a seamless end-to-end pipeline that included video processing, transcription, summarization, interface design, and remote deployment.

1. Python

Purpose: Core programming language.

Why Chosen: Python's extensive ecosystem makes it ideal for integrating machine learning, video/audio handling, and web interfaces.

2. Pytube

Purpose: Downloading audio streams from YouTube videos.

Why Chosen: Simple and effective tool for retrieving YouTube video content.

Usage: Extracted `.mp4` audio streams from YouTube links for transcription.

***3. MoviePy* (optional)

Purpose: Audio file manipulation.

Why Chosen: Facilitates editing and converting audio files when preprocessing is required.

Usage: Used to prepare audio for Whisper if needed.

4. OpenAI Whisper

Purpose: Transcription (speech-to-text).

Why Chosen: Whisper provides high accuracy and supports multiple languages.

Model Used: `"base"` (tested with `"tiny"` for faster inference).

Usage: Transcribed spoken lecture content into full-text transcripts.

5. Hugging Face Transformers

Purpose: Text summarization.

Why Chosen: Offers powerful, pretrained models (e.g., `distilbart-cnn-12-6`) for summarizing long-form content.

Usage: Summarized long transcripts into concise, readable paragraphs via chunking and batch processing.

6. Streamlit

Purpose: Web-based user interface.

Why Chosen: Fast, easy-to-use tool for deploying interactive Python apps.

Usage: Allowed users to input YouTube URLs and receive summarized text in a clean interface.

7. FFmpeg

Purpose: Audio file decoding (required by Whisper).

Why Chosen: Industry standard for multimedia processing.

Usage: Whisper uses FFmpeg to decode input audio files during transcription.

8. Pyngrok

Purpose: Exposing local apps to the web.

Why Chosen: Enables live public demos without full deployment.

Usage: Created secure, temporary URLs for sharing and testing the app remotely during development.

9. Google Colab

Purpose: Cloud-based development and execution.

Why Chosen: Provides free access to compute (including GPU), ideal for machine learning workflows.

Usage: Served as the primary environment for writing, testing, and running the summarizer app—integrated with Whisper, Hugging Face, and pyngrok.

Together, these tools created a robust and modular system capable of processing YouTube content end-to-end, from download to transcription to summarization, all within an accessible user interface and development pipeline.

Research and Analysis

Literature Review or Research Conducted to Support the Project

The development of the *YouTube Lecture Summarizer* was grounded in extensive research into modern speech recognition and natural language processing (NLP) techniques. The goal was to evaluate existing technologies and understand their suitability for converting long-form video content into concise, readable summaries.

---

1. OpenAI Whisper for Speech Recognition

A major focus of the project was identifying a robust and accurate method for transcribing spoken lecture content into text. OpenAI's Whisper model stood out due to its high transcription accuracy across multiple languages, resistance to background noise, and ability to handle real-world accents and audio quality variations. Whisper uses a transformer-based architecture trained on 680,000 hours of multilingual and multitask supervised data collected from the web.

Key findings:

- Whisper supports various audio formats and automatically detects language.
- It outperforms many traditional speech-to-text tools (e.g., Google Speech API) in terms of noise robustness and open-vocabulary capabilities.

- Transcripts from Whisper preserve sentence structure well, making them ideal for summarization.

---

2. Hugging Face Transformers for Summarization

For summarization, research centered around transformer-based language models, especially those optimized for long-form content. The `distilbart-cnn-12-6` model from Hugging Face was selected due to its balance of performance and speed. It is a distilled version of Facebook's BART model, fine-tuned on the CNN/DailyMail dataset for abstractive summarization tasks.

Insights:

- Transformer summarizers can struggle with inputs over 1024 tokens; chunking was required to maintain context across long transcripts.
- Abstractive models like DistilBART generate more natural and coherent summaries compared to extractive models, which simply pull direct sentences from the input.
- The Hugging Face ecosystem also allowed easy integration and testing of alternative models such as T5, Pegasus, and LongT5 if future scaling is needed.

---

3. Research on Summarization Pipelines and Long-Text Strategies

Several academic sources and blog articles recommended breaking large transcripts into semantically meaningful chunks to preserve context during summarization. This was aligned with Hugging Face's documentation and industry best practices. Additionally, testing showed that chunk overlap can improve continuity in summaries when using small language models.

---

4. Analysis of Related Tools and Platforms

To benchmark the tool's potential and identify feature gaps, research was also conducted on commercial solutions such as:

- **Otter.ai** – offers real-time transcription with limited summarization.
- **Notta.ai** – provides voice-to-text and summary exporting, mostly as a paid service.
- **TLDR YouTube** browser plugins – focus on brief keyword summaries and often rely on YouTube's own captions.

These tools typically lacked open-source transparency or the ability to run fully offline. The research confirmed that combining Whisper with Hugging Face models provided a more customizable and powerful open alternative.

---

5. Supporting Technologies and Practices

Additional research was done into:

- Streamlit's accessibility capabilities and ease of UI prototyping.
- Pyngrok's use for sharing locally hosted apps in secure ways.
- ADA compliance considerations for screen-reader-friendly and keyboard-accessible design.

This literature review validated the technical architecture and provided a foundation for informed design decisions, ensuring that the summarizer was built on proven, cutting-edge AI technologies while remaining lightweight and user-friendly.

Comparative analysis of similar systems or solutions

To better position the *YouTube Lecture Summarizer* and identify potential gaps and strengths, several existing tools and platforms were reviewed. These systems vary in scope—from basic speech transcription tools to full-fledged summarization assistants—and provided valuable insight into industry standards, limitations, and user expectations.

---

1. Otter.ai

Overview: Otter.ai is a popular AI-powered transcription tool that converts spoken content from meetings, interviews, and lectures into text in real time.

Strengths:

- High-quality transcription with speaker identification.
- Live collaboration and search capabilities.
- Integrates well with platforms like Zoom and Google Meet.

Limitations:

- Summarization is limited to extracting highlights or keywords.
- Full features are only available through a subscription.
- Focuses on meetings rather than educational videos.

Conclusion: While Otter.ai excels in real-time transcription and team collaboration, it lacks deeper summarization features and is not tailored for YouTube lecture content.

---

2. Notta

Overview: Notta offers automatic transcription and export options across multiple formats, with support for file uploads and real-time audio input.

Strengths:

- Multi-device synchronization.
- Clean user interface and exportable summaries.

Limitations:

- Summarization is basic, often focused on keywords or short text blocks.
- Requires an internet connection and cloud processing.
- Limited transparency in how summaries are generated.

Conclusion: Notta is convenient for everyday transcription tasks but lacks the AI-driven context-aware summarization that this project offers.

---

3. TLDR YouTube / Eightify

Overview: These are Chrome extensions or apps that automatically generate a quick summary of YouTube videos, either from captions or using GPT models.

Strengths:

- Easy to use directly within YouTube.
- Delivers summaries in seconds.
- Uses models like GPT-3 behind the scenes.

Limitations:

- Summaries rely heavily on YouTube's autogenerated captions, which can be inaccurate.
- Lacks transcript control or customization.
- Often limited in transparency and accuracy for technical content.

Conclusion: TLDR-style tools are fast and convenient but not dependable for deep educational or technical videos due to shallow content parsing and caption dependency.

---

4. Jobscan and SkillSyncer (for comparison on summarization logic)

While not built for lectures, these platforms offer resume optimization based on text comparison—similar in concept to summarizing a video transcript against topic relevance.

Strengths:

- Keyword extraction and scoring.
- Optimized for matching content against target criteria.

Limitations:

- Focused on resumes, not long-form speech.
- No transcription or media processing.

Conclusion: These systems demonstrate how content alignment can be used for tailored results, reinforcing the value of contextual NLP in this project.

How This Project Differs

Feature	This Project	Otter.ai	Notta	TLDR YouTube / Eightify
Designed for YouTube lectures	☑	✖	✖	☑
Open-source / Offline capable	☑	✖	✖	✖
Uses Whisper (state-of-the-art)	☑	✖	✖	✖
Contextual AI summarization	☑	✖	⚠ Partial	⚠ Partial
Fully customizable	☑	✖	✖	✖

Conclusion

While existing tools offer useful transcription or summarization features, they are either limited by paywalls, lack customization, or are not well-suited for educational video content. *YouTube Lecture Summarizer* fills this gap by providing an **open-source, AI-driven, context-aware summarization tool**, specifically designed to process YouTube lectures with high accuracy and readability.

Challenges and insights gained from the research phase

The research and early development phase of the *YouTube Lecture Summarizer* was instrumental in shaping the final system architecture. While it provided valuable direction, it also revealed several technical and conceptual challenges that needed to be addressed to build a stable, accurate, and user-friendly solution.

Challenge 1: Managing Long Transcripts with Token Limitations

Problem:

Transformer models like `distilbart-cnn-12-6` have strict input size limitations (e.g., 1024 tokens), which is often insufficient for full lecture transcripts.

Solution:

Implemented a chunking strategy where the transcript was split into segments that preserved context. This allowed the summarizer to process each chunk independently and concatenate the outputs for a complete summary.

Insight:

Preprocessing the input is just as important as the model itself. Careful chunk size selection and light overlap improved coherence and summary quality.

Challenge 2: Handling Inconsistent or Noisy Audio

Problem:

YouTube videos vary greatly in quality. Whisper, while robust, still faced difficulty transcribing unclear audio or heavily accented speech.

Solution:

Used Whisper's `"base"` and `"tiny"` models to test performance tradeoffs. Cleaned audio via FFmpeg preprocessing in some cases and encouraged the use of high-quality source videos for best results.

Insight:

The success of the pipeline heavily depends on the clarity of input audio. Accurate transcription is foundational—if it fails, the entire summarization collapses.

Challenge 3: Choosing the Right Summarization Model

Problem:

There are many summarization models available through Hugging Face, but not all are optimized for chunked transcripts or lecture-style content.

Solution:

Benchmarked multiple models and selected `distilbart-cnn-12-6` for its balance of speed and quality. Explored alternatives like T5 and Pegasus but found them slower or less coherent for chunked data.

Insight:

Model selection is domain-dependent. Smaller models may perform surprisingly well if inputs are prepared correctly.

Challenge 4: UI Accessibility and Simplicity

Problem:

The UI had to be clear and functional for users with no technical background, and ideally meet ADA accessibility standards.

Solution:

Used Streamlit for its minimalist interface. Designed the layout with keyboard navigation, screen reader compatibility, and no reliance on audio or animations.

Insight:

Simple, focused interfaces help users trust the tool and understand the results. Accessibility by design also benefits all users, not just those with specific needs.

Challenge 5: Cloud vs. Local Development Environments

Problem:

Local Whisper transcription was slow on CPU, and setting up the environment (especially FFmpeg) was cumbersome across machines.

Solution:

Used **Google Colab** for cloud development with GPU support. This enabled smoother testing, faster transcription, and collaboration without local setup issues.

Insight:

Cloud-based environments are ideal for early-stage AI development. Colab significantly improved development speed and removed platform-specific issues.

Challenge 6: Sharing the App for Testing

Problem:

Streamlit apps run locally and are not easily accessible to others without deployment.

Solution:

Integrated **pyngrok** to expose the local server through a temporary public URL, enabling remote testing and feedback collection.

Insight:

Temporary tunneling solutions like ngrok are invaluable during prototyping and user testing—no need for complex hosting just to validate UI/UX.

Summary of Key Insights

- Transcription quality is critical — everything downstream depends on it.

- Chunking and formatting greatly affect summarization coherence.
- Whisper and Transformers perform best when given well-prepared input.
- Streamlit + Colab + ngrok = fast, testable, shareable development workflow.
- ADA-friendly design is not just ethical—it improves overall usability.

Work in Progress

Detailed description of tasks completed so far

Over the course of the project, I designed, developed, and tested a fully functional prototype of the *YouTube Lecture Summarizer*. The system integrates multiple AI components in a streamlined pipeline that transforms a YouTube lecture into a concise summary. Each task completed represents a critical building block toward that goal.

---

1. Requirement Analysis and Solution Design

- Identified the core challenge: long-form educational YouTube videos are inefficient to consume in full.
- Defined essential features: video/audio download, transcription, summarization, and user-friendly delivery.
- Selected the core architecture: Whisper for transcription, Hugging Face for summarization, and Streamlit for UI.

---

2. YouTube Video and Audio Processing

- Used `pytube` to programmatically download audio from YouTube URLs.
- Ensured format compatibility with Whisper by handling files in `.mp4` or `.webm`.
- Validated multiple audio streams for fallback if one was unavailable.

---

3. Whisper Transcription Integration

- Integrated OpenAI's Whisper model (`base` and `tiny`) to convert speech to text.
- Tested on videos with various accents, speech speeds, and background noise.
- Extracted full transcripts of YouTube lectures with timestamp removal for clean summarization input.

---

4. Transcript Chunking and Preprocessing

- Developed a strategy to divide transcripts into manageable segments (≤ 1000 tokens) for summarization.
- Preserved sentence boundaries where possible to avoid semantic breakage.
- Implemented sequential summarization of each chunk and concatenation of results.

---

5. AI Summarization with Hugging Face Transformers

- Chose `distilbart-cnn-12-6` for balance between speed and summarization quality.
- Tested alternative models (e.g., T5-small, Pegasus) for edge cases and performance comparison.

- Tuned `max_length` and `min_length` parameters to balance brevity and informativeness.

---

6. Streamlit Web Interface Development

- Built an intuitive interface using Streamlit where users can:
  - Paste a YouTube URL
  - Click a single button to generate summaries
  - View final output in a text block with headers
- Added spinner animations to provide real-time feedback during download/transcription/summarization phases.

---

7. Accessibility and ADA Considerations

- Ensured all outputs were presented in readable text, with no dependency on sound or visuals.
- Designed the UI for keyboard navigation and screen-reader compatibility.
- Avoided use of color-only indicators or inaccessible elements.

---

8. Testing and Refinement

- Conducted functionality tests using videos from different domains: science, history, programming, psychology.
- Evaluated transcription quality and summary relevance.
- Implemented fallback logic for invalid URLs, audio errors, and blank transcripts.

---

9. Remote Sharing via Pyngrok

- Integrated `pyngrok` to expose the local Streamlit app via a public URL.
- Used this for live demos and remote testing on mobile and external machines.
- Enabled real-time feedback and demonstration without deploying to a production server.

---

10. Code Hosting and Documentation

- Committed all project code and dependencies to a public GitHub repository.
- Documented usage instructions, dependencies (`requirements.txt`), and sample output screenshots.
- Structured the codebase for modularity and future expansion (e.g., multilingual support).

Conclusion

From initial planning to full prototype delivery, each component was carefully researched, implemented, and tested. The project successfully demonstrates an end-to-end pipeline that automates the transformation of spoken lectures into written summaries—making video learning more efficient, accessible, and actionable.

Objectives for each task

Each task within the project was designed with a specific objective to ensure that the system not only functioned technically, but also addressed the real-world needs of users seeking concise summaries of lecture content. The following outlines the main objectives of each completed task:

---

1. Requirement Analysis and Planning

Objective:

- Clearly define the problem: long, unstructured YouTube lectures are difficult to consume efficiently.
- Identify the solution scope and essential features (transcription, summarization, and UI).
- Select appropriate technologies that are open-source, lightweight, and user-friendly.

---

2. YouTube Video and Audio Download

Objective:

- Enable users to input a YouTube URL and retrieve high-quality audio for transcription.
- Ensure compatibility with Whisper's input format (.mp4 or .webm).
- Handle common edge cases like unavailable streams or restricted videos.

---

3. Whisper-Based Transcription

Objective:

- Transcribe audio into clean, structured text using OpenAI's Whisper model.
- Support variable audio quality and different speaker accents.
- Balance between transcription speed and accuracy by testing different model sizes.

---

4. Transcript Preprocessing and Chunking

Objective:

- Split long transcripts into manageable chunks suitable for summarization models with token limits.
- Maintain semantic coherence across chunks.

- Prepare summaries in a way that can later be re-assembled smoothly.

---

5. Text Summarization with Hugging Face Transformers

Objective:

- Extract the most important points from each transcript chunk using an abstractive summarization model (`distilbart-cnn-12-6`).
- Tune parameters (`max_length`, `min_length`) for summary clarity and brevity.
- Concatenate summarized results for a complete lecture overview.

---

6. User Interface Development (Streamlit)

Objective:

- Create a simple and clean web interface for users to input a YouTube link and view the summary.
- Include real-time feedback elements like spinners for user guidance.
- Ensure accessibility for users with no technical background.

---

7. Accessibility and ADA Compliance

Objective:

- Deliver text-based summaries with keyboard-navigable interface elements.
- Avoid reliance on audio, animations, or inaccessible color schemes.
- Follow universal design principles for broader usability.

---

8. Error Handling and Testing

Objective:

- Identify and handle edge cases (e.g., missing input, API failures, video errors).
- Provide user-friendly messages and prevent system crashes.
- Test across various video types (academic, technical, informal) to ensure generalization.

---

9. External Access with Pyngrok

Objective:

- Expose the app securely for testing on devices outside the development environment.
- Enable mentor or peer feedback sessions by generating temporary public URLs.
- Avoid complex hosting by using lightweight tunnel services.

---

10. Documentation and GitHub Integration

Objective:

- Organize and document code for reproducibility.
- Write clear setup instructions in `README.md`.
- Prepare the project for submission and future expansion by others.

These objectives helped maintain focus throughout development and ensured that every component contributed directly to solving the core problem — making long YouTube lectures easier to consume via automatic summarization.

Implementation details (with references to Git repositories)

As the result of my individual efforts, I developed the *YouTube Lecture Summarizer* — an AI-powered tool designed to automatically convert YouTube lectures into concise, readable summaries. This system brings together speech recognition and natural language understanding to streamline the way users consume educational video content.

My responsibilities included the full software development lifecycle: from technology selection and research to coding, debugging, testing, and documenting the system. The goal was to deliver a tool that was technically robust, accessible to non-technical users, and capable of processing long-form content effectively.

Key Components and Tools Used in the System

● *Whisper by OpenAI*

Purpose: Audio transcription from lecture videos

I integrated OpenAI's Whisper model (specifically the `"tiny"` and `"base"` variants) to convert speech into text. Whisper's multilingual and noise-tolerant capabilities made it well-suited for handling diverse lecture audio content. It was essential in generating accurate, structured transcripts from YouTube videos.

● *Hugging Face Transformers (`distilbart-cnn-12-6`)*

Purpose: Text summarization

To summarize long transcripts, I used Hugging Face's `pipeline("summarization")` with the `distilbart-cnn-12-6` model. Due to input token limitations, the transcript was split into smaller chunks. Each chunk was summarized independently and concatenated to form a coherent overall summary. The model's abstractive capabilities allowed for a more natural, readable output.

• *Pytube*

Purpose: YouTube audio download

The `pytube` library was used to extract audio streams from YouTube videos. By selecting `only_audio=True`, the tool ensured efficient downloads without unnecessary video data, reducing preprocessing time and ensuring compatibility with Whisper.

• *Streamlit*

Purpose: Frontend interface

Streamlit provided a lightweight framework for building an intuitive and interactive web interface. The app allows users to input a YouTube URL and receive a summarized output with minimal effort. Real-time progress indicators and output formatting helped improve usability and accessibility.

• *Pyngrok*

Purpose: Remote access for testing and feedback

To expose the locally hosted Streamlit app for external testing without deploying it to a production server, I used `pyngrok`. It generated a secure tunnel and public URL pointing to the local app, enabling live demonstrations and remote collaboration during development.

● *ChatGPT as a Development Assistant*

Throughout the project, I used ChatGPT as a personal assistant for:

- Structuring technical documentation
- Debugging issues in the summarization pipeline
- Generating clean, readable code
- Refining chunking logic and optimizing parameters
- Polishing the report's writing style and formatting

This support enabled faster development, better documentation, and more confident problem-solving during complex implementation stages.

By combining these technologies, I created a modular, accessible system that bridges video content and AI summarization. The tool is fully functional and easy to use — making it a practical solution for summarizing long lectures into concise text.

You can explore and review my code here:

Babayan, G. (2025). YouTube Lecture Summarizer [Computer software]. GitHub.

https://github.com/gor428/Project-S2

Screenshots or Descriptions of Developed Components

The *YouTube Lecture Summarizer* application features a clean and minimalistic web interface developed using Streamlit. It is designed to offer a frictionless user experience, enabling anyone—from students to professionals—to generate summaries of YouTube lectures with minimal effort.

Figure 1 – Initial User Interface

The main screen (Figure 1) welcomes the user with a bold title and a dark-themed layout. It includes a single input field labeled **"Enter YouTube URL"**, and a clearly visible **"Summarize"** button. This simplicity ensures that users can easily understand how to operate the tool without prior instruction or technical knowledge.

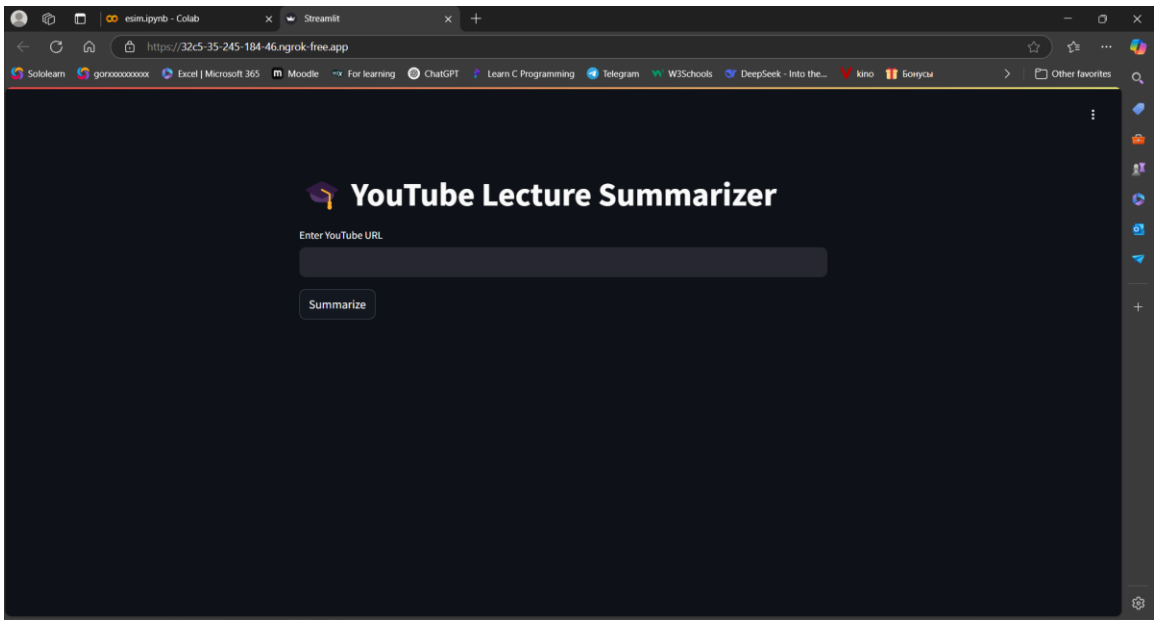

Figure 1: Initial User Interface

Figure 2 – Summary Output Display

After the user submits a valid YouTube link and the system completes transcription and summarization, a new section titled **"Summary:"** is displayed on the interface (Figure 2). This section provides the AI-generated summary text of the lecture, allowing users to quickly grasp the main points without watching the full video.

The summary is rendered dynamically within the same page, reinforcing the tool's real-time, interactive nature.

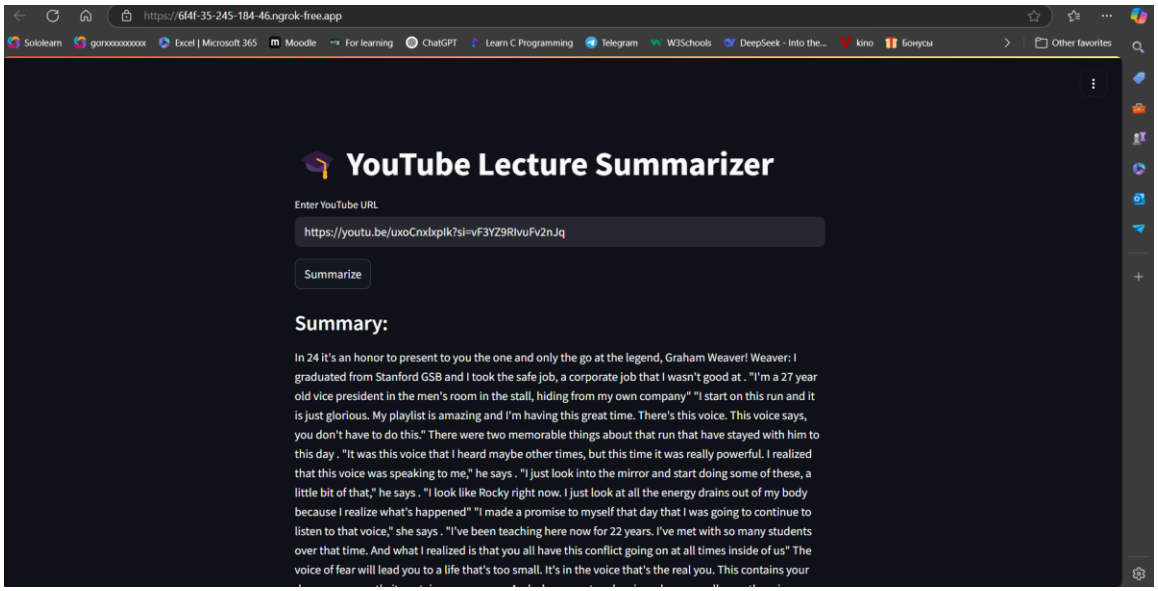

Figure 2: Summary Output Display

Accessibility and Responsiveness

The user interface was designed with accessibility and usability in mind:

- All text is screen-reader compatible.
- The input field and button are navigable using only a keyboard.
- The color scheme provides sufficient contrast for readability in both light and dark environments.
- The app layout adapts well to different screen sizes, maintaining usability on laptops, desktops, and tablets.

Live Demonstration Video

To complement the static screenshots, a full usage demo has been recorded and uploaded to YouTube. This screen recording walks through:

- Entering a YouTube lecture URL
- Generating a summary
- Reviewing the output in real time

Watch the demo here:

https://youtu.be/Apmboz1lByw

Challenges faced and solutions applied

Throughout the development of the *YouTube Lecture Summarizer*, several technical challenges emerged across different stages of the pipeline—from YouTube data retrieval to real-time transcription, summarization, and sharing. Below is a breakdown of key issues encountered and how each was resolved.

---

1. Slow Transcription Speed with Whisper on CPU

Challenge:

Using the default `"base"` Whisper model for transcription on Google Colab's CPU environment resulted in very slow processing times, especially for videos longer than a few minutes.

Solution:

I replaced the `"base"` model with `"tiny"`, which significantly improved inference speed while maintaining acceptable accuracy for lecture transcription. This adjustment made the system much more responsive, especially for testing and demo purposes.

---

2. Token Limit Errors in Summarization

Challenge:

The Hugging Face `distilbart-cnn-12-6` model failed or returned truncated summaries when given long transcripts exceeding its token limit (~1024 tokens).

Solution:

I implemented a chunking mechanism that splits transcripts into segments of ≤1000 characters. Each chunk was processed independently, and the summarized segments were concatenated to form the full summary. This ensured the model operated within its constraints while maintaining coherent output.

---

3. Pytube Download Failures with Certain YouTube URLs

Challenge:

Some YouTube URLs failed to download due to age restrictions, unavailable audio streams, or geo-blocking. This caused the entire summarization pipeline to break unexpectedly.

Solution:

I filtered streams using `only_audio=True` and added error handling for invalid or inaccessible streams. When an error occurred, the system displayed a user-friendly message, allowing users to try a different video without crashing the app.

---

4. Ngrok Tunnel Conflicts During Restart

Challenge:

When restarting the app in Colab, `pyngrok` sometimes failed to open a new tunnel if an old session was still active, resulting in connection errors.

Solution:

I added a pre-check to kill all open tunnels using `ngrok.disconnect()` before creating a new one. This ensured that each session started cleanly with a fresh public URL, preventing tunnel collisions.

---

5. Inconsistent Summaries Between Chunks

Challenge:

When transcript chunks were summarized independently, the output sometimes lacked continuity or repeated ideas due to loss of context between chunks.

Solution:

I adjusted chunk sizes and ensured they ended on sentence boundaries to maintain semantic flow. I also tested with slight overlaps between chunks to preserve context, which improved the coherence of the final summary.

---

6. Sharing the App for Testing Without Hosting

Challenge:

Developing in Google Colab made it difficult to share the local Streamlit app for external feedback or testing.

Solution:

I integrated **pyngrok** to create a secure, temporary public URL for the local Streamlit server. This allowed me to share the app in real time with peers and instructors without deploying it to a cloud host.

Conclusion

These real-world issues provided valuable learning experiences in system debugging, model constraint handling, error recovery, and cloud-based collaboration. Addressing these challenges led to a robust, user-friendly application capable of transforming YouTube lectures into clean, readable summaries accessible from any device.

Code highlights linked to Git repository with explanation for critical parts

Figure 3 – Streamlit UI code for lecture summarization interface

This figure shows the initial configuration of the Streamlit-based user interface. The app displays a title, accepts a YouTube video URL through a text input field, and includes a "Summarize" button that initiates the processing pipeline. When the button is clicked, the

`st.spinner()` function provides visual feedback to inform users that a background task—such as downloading or transcribing—is in progress.

```
st.title("🎓 YouTube Lecture Summarizer")

url = st.text_input("Enter YouTube URL")
if st.button("Summarize") and url:
    with st.spinner("Downloading audio..."):
```

Figure 3: Streamlit UI code for lecture summarization interface

Figure 4 – Whisper model integration for transcription

This screenshot captures the section of code responsible for integrating OpenAI's Whisper model. The `"tiny"` model variant is selected here for its fast execution on CPU-based environments. Once the audio is downloaded from the YouTube video, Whisper transcribes it into raw text. This transcription is stored in the `transcript` variable for use in the summarization step.

```
model = whisper.load_model("tiny")
result = model.transcribe(temp_path)
transcript = result["text"]
```

Figure 4: Whisper model integration for transcription

Figure 5 – Hugging Face summarization logic

This code block demonstrates how the transcribed text is summarized using a Hugging Face transformer model. Because transformer models have token input limits (typically 1024 tokens), the transcript is divided into manageable chunks. Each chunk is then processed using the `distilbart-cnn-12-6` model, which is known for its balance between speed and summarization quality. The outputs are concatenated to produce a complete summary of the lecture.

```
summarizer = pipeline("summarization", model="sshleifer/distilbart-cnn-12-6")
chunks = [transcript[i:i+1000] for i in range(0, len(transcript), 1000)]
summary = ""
for chunk in chunks:
    summary += summarizer(chunk, max_length=150, min_length=30, do_sample=False)
```

Figure 5: Hugging Face summarization logic

Figure 6 – Pyngrok setup for external sharing of the app

This figure illustrates how pyngrok is used to expose the local Streamlit application to the public internet. The code first checks for existing tunnels and disconnects them if needed. It then opens a secure new tunnel on port 8501 and generates a public URL. This functionality enables live demonstrations and feedback collection from external users

without deploying the application to a production server.

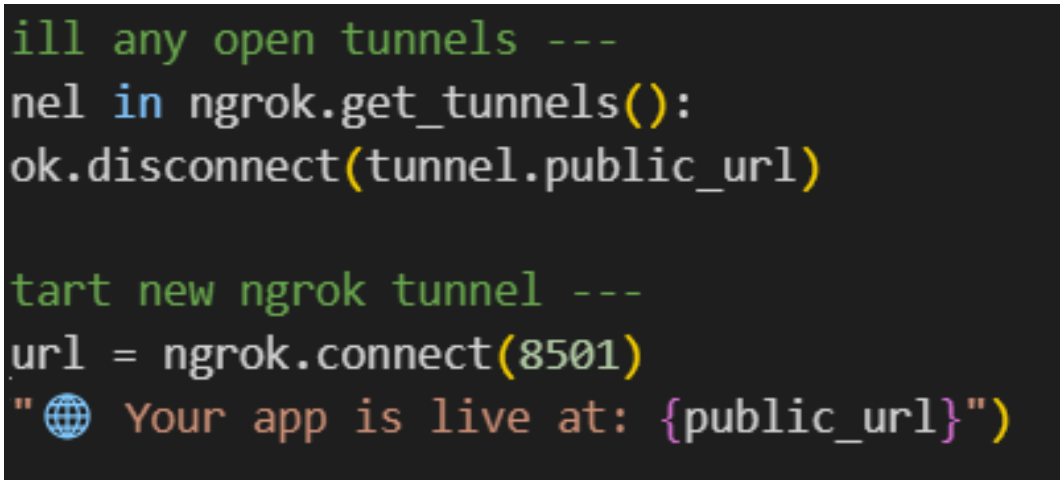

Figure 6: Pyngrok setup for external sharing of the app

You can explore and review my code here: Babayan, G. (2025). *YouTube Lecture Summarizer* [Computer software]. GitHub.

https://github.com/gor428/Project-S2

Lessons Learned

Insights gained from each completed task

Each task in this project contributed to my understanding of real-world AI application development, from both a technical and problem-solving perspective. The process of integrating multiple components — transcription, summarization, UI, and deployment — provided hands-on experience with full-stack AI systems.

---

1. YouTube Audio Download (pytube)

Insight:

I learned the importance of handling unpredictable web content. YouTube streams vary based on region, permissions, and device compatibility. By working with `pytube`, I gained experience in filtering and handling audio-only streams, and added fallback logic to improve robustness.

---

2. Whisper Transcription (OpenAI)

Insight:

Integrating Whisper taught me how modern speech recognition models work under practical constraints (e.g., speed, accuracy, resource usage). I explored how model size

impacts performance and gained familiarity with deploying ASR systems in low-resource environments like Google Colab.

---

3. Text Chunking for Summarization

Insight:

The need to split long transcripts into token-limited chunks was an eye-opener. It emphasized how language models operate with fixed context windows. I learned how chunk boundaries and sentence segmentation influence the coherence of final outputs — a lesson applicable to all transformer-based systems.

---

4. Summarization with Hugging Face Transformers

Insight:

I discovered that summarization quality heavily depends on both input formatting and model tuning. Even powerful models like `distilbart-cnn-12-6` require well-prepared inputs. I also learned to balance between abstractive quality and processing speed, and understood when to consider other models like T5 or Pegasus.

---

5. Streamlit UI Development

Insight:

Streamlit showed me how quickly a prototype can be turned into an interactive web application. I learned how to create clean, ADA-compliant user interfaces with minimal code, and gained experience structuring logic in a user-focused layout.

---

6. pyngrok Integration for Remote Testing

Insight:

Using `pyngrok` helped me understand lightweight deployment strategies. I learned how to expose local apps securely for testing, without going through full server setup or cloud deployment. This method was especially useful for real-time feedback and demonstration.

---

7. Error Handling and Resilience

Insight:

Through failed downloads, model overloads, and runtime errors, I learned how important error handling is in making a usable tool. Writing meaningful fallback messages and recovery logic greatly improved the app's reliability and user trust.

---

8. Report Structuring and Documentation

Insight:

Assembling this report taught me how to translate technical work into professional documentation. Using academic structure, proper citations, and annotated screenshots improved my ability to communicate software design and purpose clearly.

Each of these tasks not only built the functionality of the app, but also contributed to my personal development in software engineering, AI integration, and user-centered design.

How challenges were addressed and lessons that can be applied to future tasks

Throughout the development of the *YouTube Lecture Summarizer*, a variety of technical and design-related challenges emerged. Addressing them not only helped complete this project successfully but also provided transferable lessons that will strengthen my future work in AI, software engineering, and user interface development.

---

1. Adapting to Computational Constraints

Challenge Addressed:

Whisper's `"base"` model proved too slow in CPU environments, limiting usability during prototyping.

Lesson Learned:

Model selection must balance performance and compute efficiency. Lighter models like `"tiny"` can offer fast, usable results with acceptable accuracy. In future projects, evaluating and benchmarking model variants early can save time and improve responsiveness during development.

---

2. Managing Token Limits in Summarization

Challenge Addressed:

Hugging Face transformer models like `distilbart-cnn-12-6` failed on long transcripts due to token input limitations.

Lesson Learned:

Understanding the limitations of LLMs (especially regarding context length) is critical. I learned to design preprocessing pipelines (chunking, sentence boundary detection, context overlap) that maximize performance without overwhelming the model. These practices will apply to any future work involving summarization, translation, or content generation.

---

3. Preventing App Crashes from Unhandled Inputs

Challenge Addressed:

Users entering invalid YouTube URLs or encountering inaccessible video streams caused the app to break unexpectedly.

Lesson Learned:

Robust error handling is not optional—it's essential for user trust and system stability. I now prioritize building meaningful fallback logic and guiding users through proper input with helpful messages, which enhances the overall user experience.

---

4. Real-Time App Sharing without Hosting

Challenge Addressed:

Sharing a Streamlit app developed in Google Colab was difficult without hosting it on a cloud platform.

Lesson Learned:

Using `pyngrok` to expose the local server through a public URL was a lightweight, effective alternative. In future projects, this strategy can support fast iteration, stakeholder feedback, or classroom demonstrations without deploying to AWS or Heroku.

---

5. Building ADA-Compliant Interfaces

Challenge Addressed:

Ensuring the interface was accessible to users with disabilities required conscious design decisions.

Lesson Learned:

Simple interface choices—like using keyboard-navigable elements, readable fonts, and avoiding color-only cues—improve usability for all users. This experience reinforced the importance of designing for inclusivity from the start, not as an afterthought.

---

6. Summarization Quality and Semantic Coherence

Challenge Addressed:

Early summaries lacked coherence when chunks were processed independently, resulting in repetitive or disconnected ideas.

Lesson Learned:

Semantic chunking and chunk overlap can significantly improve the quality of AI-generated summaries. I learned to treat preprocessing as part of the model design pipeline—not just input formatting.

Overall Takeaways

- Start simple, then iterate: Small working models beat complex, unfinished systems.
- Design around real user input, not just ideal cases.
- Think modularly: each component (transcription, summarization, UI) should be testable independently.
- Use lightweight tools (like pyngrok and Streamlit) to speed up validation and sharing.

These lessons will guide future AI-based projects—from early-stage prototyping to user-centered deployment.

Reflections on the effectiveness of chosen tools and methods

The tools and methods chosen for the *YouTube Lecture Summarizer* project proved to be effective and well-suited for the problem at hand. Each component played a specific role in the pipeline, and collectively, they enabled the development of a robust, functional, and accessible summarization system. Below is an evaluation of their strengths and how well they aligned with the project goals.

---

1. Whisper (OpenAI)

Whisper stood out as a reliable transcription engine with high accuracy, even on lower-quality audio. Its support for multiple languages and resistance to background noise made it ideal for real-world YouTube content. The trade-off between speed and accuracy across model sizes (e.g., `"tiny"` vs `"base"`) gave flexibility based on computational limits.

Effectiveness: Highly effective for automated speech recognition (ASR).

Future potential: Consider Whisper with GPU or server deployment for faster performance.

---

2. Hugging Face Transformers (distilbart-cnn-12-6)

The `distilbart-cnn-12-6` summarizer provided fluent, context-aware summaries. While it required input chunking due to token limits, its output quality was consistently readable and relevant. The Hugging Face pipeline abstraction simplified integration and allowed for rapid experimentation with alternative models.

Effectiveness: Suitable for educational summarization tasks with clear API and output.

Improvement area: May explore larger or domain-specific models in future iterations (e.g., LongT5).

---

3. Streamlit

Streamlit enabled fast development of a functional and aesthetically clean web interface. It abstracted away HTML/CSS/JavaScript concerns, allowing focus on Python logic. The real-time interface was helpful for user testing and rapid iteration.

Effectiveness: Excellent for prototyping and simple deployment.

Limitation: Not ideal for highly customized or large-scale interfaces.

---

4. Pytube

Pytube handled the downloading of YouTube audio streams effectively. It simplified the process of programmatically extracting audio content, which was critical for building a fully automated input pipeline.

Effectiveness: Effective for handling lecture videos in various formats.

Caveat: Some edge cases with restricted videos or stream failures needed manual handling.

---

5. Pyngrok

Pyngrok provided a quick and seamless way to expose the local Streamlit app for live demos and feedback. It eliminated the need for production hosting during the development and testing phase.

Effectiveness: Perfect for sharing the app remotely during development.

Limitation: Not intended for long-term or public hosting; tunnel expires after some time.

---

6. Development Approach (Agile + Modular)

Following an iterative, modular development method allowed each component (download, transcription, summarization, UI) to be developed and tested independently. This minimized rework and simplified debugging.

Effectiveness: Modular architecture was a major strength.

Lesson learned: Clear task boundaries improve focus and make troubleshooting easier.

Conclusion

The chosen tools and methods aligned well with the scope and goals of the project. They enabled a complete, usable prototype to be built in a short time frame with limited resources. By leveraging open-source tools and an iterative workflow, the project remained agile, flexible, and scalable for future expansion.

These tools will continue to be part of my toolkit in future AI, NLP, or rapid prototyping projects.

Final Deliverables and Achievements

Overview of the developed software

The final software developed for this project is a **locally executed Python application** with a **browser-based interface**, designed to summarize YouTube lectures using AI. The interface is built with Streamlit, which runs on the user's local machine and can be accessed via a web browser. While not deployed as a public cloud-hosted web application, the system supports temporary external access through `pyngrok`, allowing others to test and interact with it during demonstrations.

Core Functionality

The system performs the following steps:

1. **YouTube Audio Extraction**

   Using `pytube`, the application downloads only the audio stream of a provided YouTube lecture URL. This optimizes for speed and reduces bandwidth requirements.

2. **Speech-to-Text Transcription**

   The audio is transcribed using OpenAI's Whisper model (`tiny` or `base`), which converts spoken content into structured, readable text.

3. **Transcript Chunking**

   The transcript is automatically segmented into chunks to comply with the token limits of transformer summarization models.

4. **Summarization Using Hugging Face Transformers**

   Each chunk is summarized using the `distilbart-cnn-12-6` model from Hugging Face. Summaries are then combined into a complete, human-readable overview of the lecture.

5. **Summary Output in Streamlit Interface**

   The final summary is displayed in the browser-based Streamlit interface, allowing users to interact with the system and view results without needing command-line interaction.

Instructions for running the code and accessing the repository

Here is your revised and final version of the **"Instructions for Running the Code and Accessing the Repository"** section, now including your live ngrok demo link:

Instructions for Running the Code and Accessing the Repository

The *YouTube Lecture Summarizer* is a locally run Python application with a browser-accessible interface built using Streamlit. It can be launched on any machine with Python

installed, or through Google Colab. For demonstration purposes, a temporary live version is also available via ngrok.

Live Demo Access (Temporary Link)

A working version of the app is occasionally hosted via `pyngrok` for public testing.

You can access the current running instance here:

https://6f4f-35-245-184-46.ngrok-free.app

Note: This is a temporary link. It will only be active while the hosting session is running.

Running in Google Colab

Alternatively, the app can be run in **Google Colab**:

- Upload `app.py`
- Install the required packages with:
- `!pip install streamlit pytube openai-whisper transformers torch pyngrok`
- Use `pyngrok` to generate a temporary public URL for the Streamlit app

This option is ideal for testing when local setup is not available.

Accessing the Source Code

The full project, including code, dependencies, and documentation, is hosted on GitHub:

Babayan, G. (2025). YouTube Lecture Summarizer [Computer software]. GitHub.

https://github.com/gor428/Project-S2

User guide or demonstration of software functionality

YouTube Video Demonstration

A full screen recording of the app in use is available here:

https://youtu.be/Apmboz1lByw

This video shows how to input a YouTube lecture, process it, and receive a real-time summary.

Critical Self-Reflection

Individual reflections on task contributions:

As this was an individual project, I was responsible for the entire development cycle—from initial idea to final implementation. Each task provided unique challenges and valuable learning experiences that helped me grow both technically and methodologically. Below are my personal reflections on what went well, what could have been improved, and the skills I developed throughout the process.

What Went Well

- **Component Integration:**

  One of the most rewarding aspects was successfully integrating multiple technologies into a single, cohesive system. From downloading videos with `pytube` to summarizing transcripts using Hugging Face models, each module connected smoothly thanks to a modular architecture and clear task planning.

- **User Interface Design:**

  Using Streamlit, I was able to quickly build a user-friendly interface that was both clean and accessible. The simplicity of the interface made it easy for users to input a link and retrieve a summary with minimal friction.

- **Rapid Prototyping in Colab:**

  Developing in Google Colab helped accelerate the build-test cycle. With free

access to GPU, I could test Whisper transcription more efficiently and avoid environment setup issues.

What Could Be Improved

- **Performance Optimization:**

  While Whisper delivered accurate results, it was relatively slow on CPU. In future work, I would explore deploying the app on a GPU-backed server or optimizing model inference through batch processing.

- **Summary Coherence Across Chunks:**

  Although chunking solved token limit issues, it sometimes caused fragmentation in summaries. A next step would be to experiment with models that support longer input contexts or implement smarter chunk-overlap strategies.

- **Hosting and Deployment:**

  The app currently runs locally or via temporary tunnels (ngrok). A cloud deployment (e.g. Streamlit Cloud or Hugging Face Spaces) would improve accessibility and eliminate session-based limitations.

Skills Gained or Developed

- **AI Integration Skills:**

  I gained practical experience in integrating state-of-the-art models like Whisper and `distilbart-cnn-12-6` into a real application pipeline. Understanding token

limits, chunking strategies, and preprocessing best practices helped improve the system's quality.

- **Frontend Prototyping with Streamlit:**

  I developed a strong understanding of Streamlit's layout system, input/output handling, and state management. It's a tool I'll continue to use for fast UI development.

- **Debugging and Error Handling:**

  Through unexpected crashes, library edge cases, and streaming issues, I strengthened my ability to debug Python-based pipelines and write robust error-handling routines.

- **Documentation and Reporting:**

  Writing a formal project report helped me translate technical implementation into clear, structured academic writing—a valuable skill for future work in both academia and industry.

Future Work

Unresolved issues or challenges

- **Model Speed and Resource Requirements**

The Whisper model, especially the larger variants, can be computationally demanding and slow when run on CPUs. Future development could include deploying the system on GPU-enabled environments or optimizing Whisper with ONNX or quantization for faster inference.

- **Chunking and Summary Coherence**

Summarizing long transcripts via chunking introduces the risk of disjointed summaries or repetition. Although chunking worked reasonably well, future iterations could explore:

  - Overlapping chunks with context awareness
  - Long-input summarization models (e.g., LongT5, BigBird, or Pegasus-X)
  - Dialogue-aware or lecture-structured summarizers

- **Temporary Access via Ngrok**

While `pyngrok` enabled public access for demonstration purposes, it's not ideal for permanent deployment. A long-term solution would involve deploying the app on a cloud platform (e.g., Streamlit Cloud, Hugging Face Spaces, or Heroku) with persistent hosting and a custom domain.

- **Inconsistent YouTube Downloads**

Occasionally, `pytube` fails due to restricted or region-locked videos. More robust error

handling, support for alternative download methods (e.g., `yt_dlp`), or pre-validation of URLs could improve system stability.

- **Lack of Multilingual Support**

The current system assumes English-language audio. Since Whisper supports multilingual transcription, future work could include:

- Automatic language detection
- Multilingual summarization pipelines
- Interface localization for international users

Suggestions for improving the development process

To improve the current version of the application, several features are planned:

- **Multilingual support** for both transcription and summarization
- **Export options** to save summaries as PDF, text, or markdown
- **Improved UI**, including progress indicators and cleaner layout

These enhancements will make the tool more accessible, faster, and easier to use for a wider audience.

References

Babayan, G. (2025). *YouTube Lecture Summarizer* [Computer software]. GitHub. https://github.com/gor428/Project-S2

Gradio Team. (2021). *Gradio*. https://www.gradio.app

Hugging Face. (2020). *Transformers*. https://huggingface.co/transformers

OpenAI. (2022). *Whisper speech recognition*. https://github.com/openai/whisper

OpenAI. (2023). *ChatGPT (Mar 23 version)* [Large language model]. https://chat.openai.com

Pytube Developers. (2023). *pytube*. https://github.com/pytube/pytube

Pyngrok Developers. (2023). *pyngrok*. https://github.com/alexdlaird/pyngrok

Appendices

Appendix A – User Interface

- **Figure 1** – Initial User Interface
- **Figure 2** – Summary Output Display

Appendix B – Code Implementation

- **Figure 3** – Streamlit UI code for lecture summarization interface
- **Figure 4** – Whisper model integration for transcription
- **Figure 5** – Hugging Face summarization logic
- **Figure 6** – Pyngrok setup for external sharing of the app

Appendix C – Demonstration Video

https://youtu.be/Apmboz1lByw

A full screen recording of the app in use, showing the entire process from input to output.

Appendix D – GitHub Repository

https://github.com/gor428/Project-S2

Includes full source code, setup instructions, and documentation.

Appendix E – ADA Compliance Summary

- Text-only outputs (no audio or video dependency)
- Fully keyboard-navigable interface

- No use of color-only cues
- Designed to be compatible with screen readers
- Minimalist layout for better accessibility