Lecture Summarization and Question-Answering System using Video-to-Speech-to-Text

Akshay Chavan, Gaurav Tejwani, Ananya Asthana

Foundations of Artificial Intelligence

Rose Sloan

Abstract

We present StudyBuddy, an AI-powered system that generates lecture summaries, structured notes, and provides intelligent question-answering capabilities from educational videos. The system first extracts audio from uploaded videos using MoviePy, then transcribes the speech using OpenAI's Whisper model. Key sentences are paraphrased and scored using cosine similarity to generate concise summaries through the NLTK library and T5 Tokenizer. Then detailed lecture notes are created using OpenAI's GPT-4o mini model with prompt engineering. The processed outputs are indexed for semantic retrieval, enabling a question-answering module that uses FLAN-T5-Large and Instructor XL embeddings to deliver contextually accurate responses to user queries. A responsive web interface, developed with Next.js and Flask, supports video uploads and presents transcripts, summaries, notes, and real-time answers within a streamlined user experience. The system has been evaluated for output quality, showing strong alignment between user queries and retrieved content. StudyBuddy simplifies lecture review by automating time-consuming tasks like note-taking and content comprehension. It offers a scalable, accessible solution for students navigating large volumes of educational content, and demonstrates the effectiveness of combining modern language models with traditional NLP techniques for academic support.

Introduction

For our project, we focused on addressing the challenge of improving the learning experience for students by helping them efficiently capture and understand key concepts from lecture content. The traditional process of watching lengthy lectures can be time-consuming, especially when students need to revisit information quickly for review or exam preparation. We aimed to create a tool that could automatically summarize lecture content, saving time, enhancing comprehension and improving learning experience.

A major source of inspiration for our project was the system described in the paper ‘HumSum: A Personalized Lecture Summarization Tool for Humanities Students Using LLMs’. HumSum is a web application specifically designed for humanities students that summarizes transcripts of lectures according to individual user preferences. It makes use of Streamlit for the user interface and uses Langchain to connect with OpenAI’s GPT model for the summarization tasks. To handle large transcripts that could exceed the token limits of large language models, HumSum utilized Langchain’s 'Map Reduce Document Chain' approach, where the transcript was divided into smaller chunks within the token limit, summarized individually based on user preferences, and then recombined into a final summary.

Building on this foundation, our project ‘StudyBuddy’ extends and generalizes the idea to support students across all disciplines, not just humanities. While HumSum accepted transcript documents as input, we expanded the functionality to allow users to input video lectures directly. Our application internally converted these videos into transcripts before performing summarization. This improvement enabled students to quickly capture the main points and

critical ideas from lengthy video lectures without needing to watch them in full, improving study efficiency.

Additionally, we added a Question-and-Answer module into our system. After generating the summary, users could interact with the summarized content through a chat interface, asking questions and receiving responses based on the summarized material. This feature further enhanced the active learning experience by supporting clarification and deeper engagement with the material.

Thus, StudyBuddy not only builds upon the work outlined in the HumSum project but also broadens its applicability and adds new interactive capabilities, all with the goal of making learning more accessible and efficient.

Methods

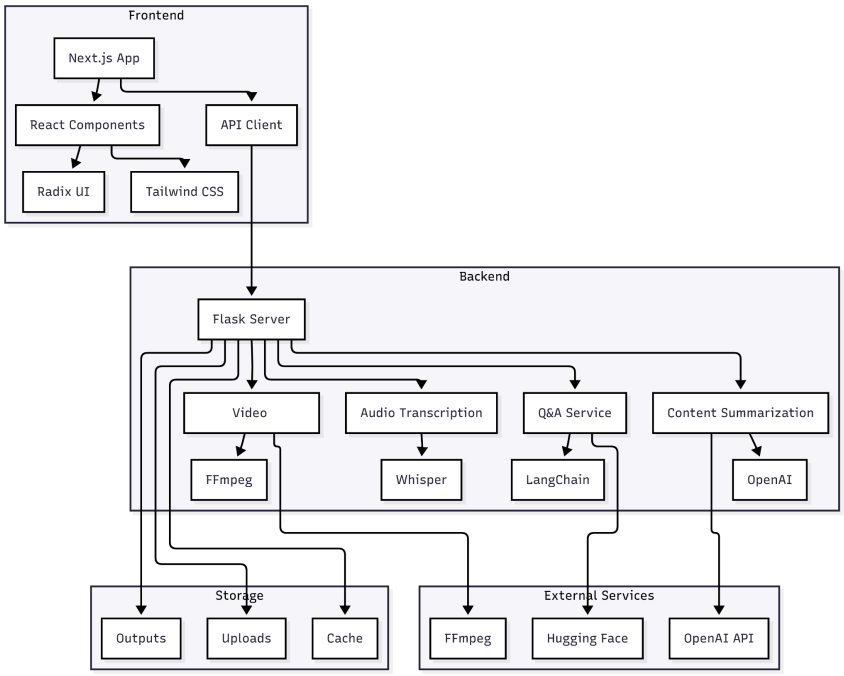

Fig 1: Architecture Diagram

Our approach to building StudyBuddy follows a modular pipeline architecture that processes educational videos end-to-end to produce lecture transcripts, summaries, structured notes, and supports semantic question answering. The system is composed of several stages, each employing widely used AI tools and NLP models.

The first stage extracts audio from uploaded video files using the MoviePy library in Python. The extracted audio is transcribed using OpenAI's Whisper model, chosen for its strong performance in multilingual and domain-specific transcription tasks. The resulting transcript is segmented into sentences, paraphrased for clarity, and scored using cosine similarity between each sentence and the lecture title. High-ranking paraphrased sentences are selected to create concise summaries. We used NLTK for tokenization and sentence parsing, and the Hugging Face T5 tokenizer for compatibility with transformer-based models.

To generate lecture notes, the full transcript is passed to OpenAI's GPT-4o mini via prompt engineering techniques designed to elicit structured, section-wise notes from dense text. The final outputs, transcript, summary, and notes, are indexed using Instructor XL embeddings for semantic search, and a question-answering module based on FLAN-T5-Large returns contextually relevant answers to user queries.

We evaluated our system using a combination of functional testing and qualitative human judgment. Each module was tested independently to ensure it produced valid and complete outputs. To evaluate accuracy and relevance, we manually reviewed generated summaries and notes by comparing them against the source video content. We began by evaluating our paraphrased sentences and validating whether the selected ones are relevant to our lecture topic. Evaluation criteria included coherence, completeness, and factual alignment. Additionally, if our

video had an existing lecture transcript we used it as an informal baseline for comparison. We also conducted informal user testing with peers who interacted with the web interface and provided feedback on usability and content quality. We contacted 3 peers who each selected a lecture video from YouTube. We downloaded the 3 videos and provided them as inputs to our model. The results of this activity are mentioned in the Results section.

First, we created the following questions to be rated on a scale of 1–5 (1 being "Very Unsatisfied" and 5 being "Very Satisfied"):

1. How satisfied are you with StudyBuddy's user interface and ease of use?

2. How would you rate the quality of the lecture notes/ summary generated by StudyBuddy?

3. How would you rate the Question-and-Answer (Q&A) module?

4. How likely are you to continue using StudyBuddy in the future?

Additionally, to evaluate specific aspects of the summarization and Q&A modules, we included the following True/False/Somewhat questions:

5. Were you satisfied with the length of the generated lecture notes/ summary ?

6. Were you satisfied with the tone and writing style of the lecture notes/ summary ?

7. Did the lecture notes/ summary effectively capture all the important information of the lecture?

8. Would you prefer further personalization options for the generated summary?

9. Did the Q&A module accurately answer your questions?

10. Were you satisfied with the length of the answers provided by the Q&A module?

This feedback allowed us to evaluate StudyBuddy's usability, summarization quality, and Q&A accuracy from a user perspective, and helped us identify areas for further improvement.

All models and libraries used were sourced from open-source or publicly accessible APIs. Whisper and GPT-4o mini were accessed through OpenAI's API. Instructor XL and FLAN-T5-Large were accessed via the Hugging Face Transformers library. The front-end was developed in Next.js, and the back-end was built with Flask.

Results

Here we present answers to all the 10 questions from the Methods section asked to all 3 participants, and then proceed with showing the summarization of our inferences and results.

Questions	**Peer A** **(Lecture title: Overview of Reinforcement Learning Approaches and Techniques)**	**Peer B** **(Lecture title: Object-oriented programming)**	**Peer C** **(Lecture title: Discover the History of English)**
Q1	5	5	4
Q2	5	4	4
Q3	4	4	3
Q4	5	5	5
Q5	True	True	True
Q6	True	True	Somewhat
Q7	True	Somewhat	True

Q8	True	True	True
Q9	Somewhat	True	False
Q10	True	True	Somewhat

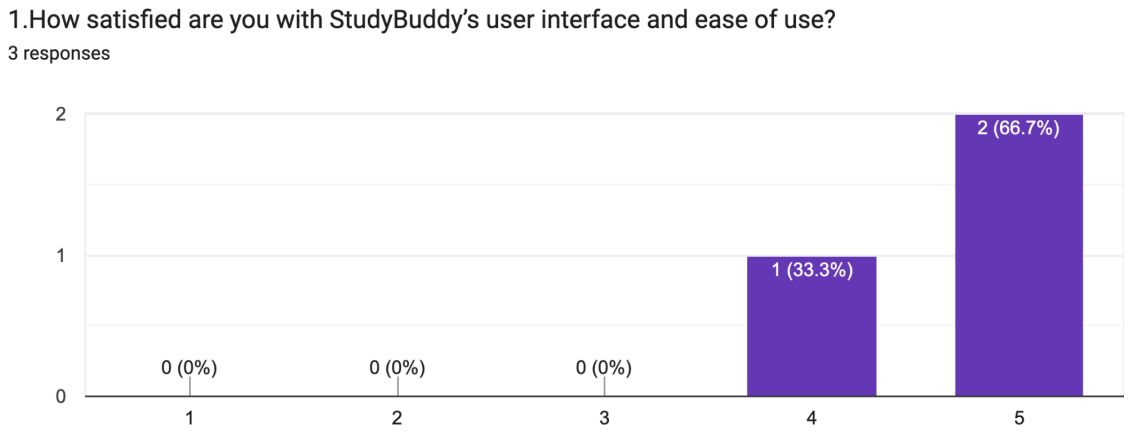

Fig 2: Question 1 Statistics

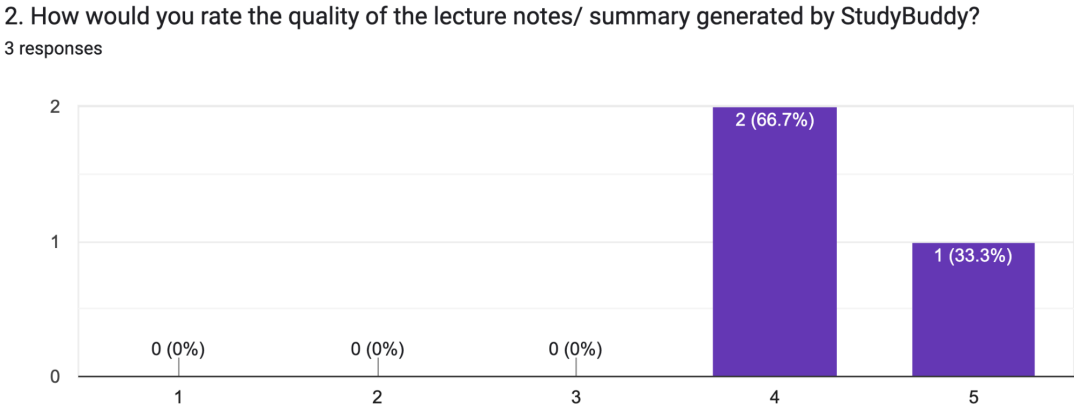

Fig 3: Question 2 Statistics

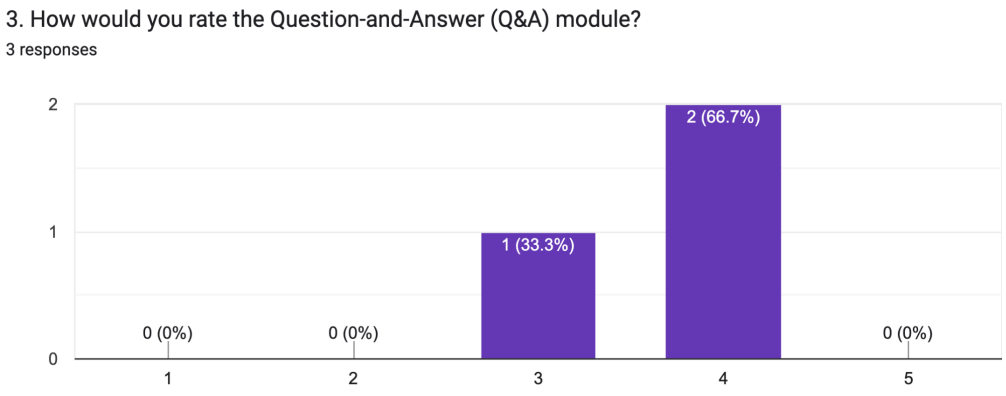

Fig 4: Question 3 Statistics

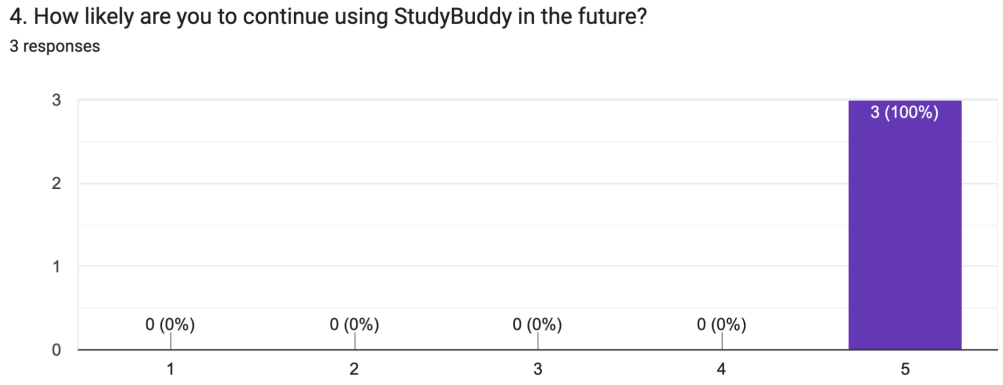

Fig 5: Question 4 Statistics

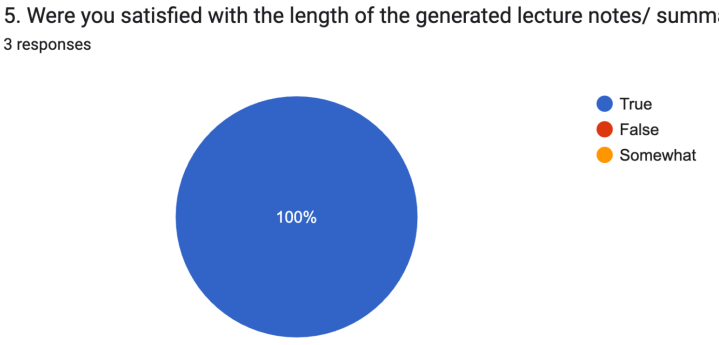

Fig 6: Question 5 Statistics

6. Were you satisfied with the tone and writing style of the lecture notes/ summary ?

3 responses

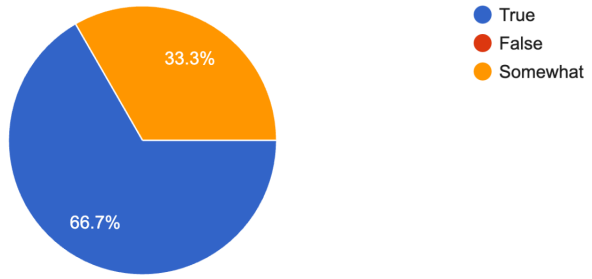

Fig 7: Question 6 Statistics

7. Did the lecture notes/ summary effectively capture all the important information of the lecture?

3 responses

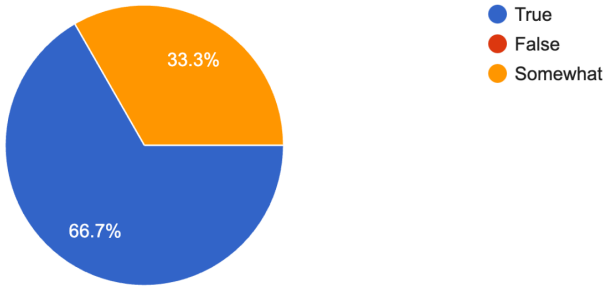

Fig 8: Question 7 Statistics

8. Would you prefer further personalization options for the generated summary?

3 responses

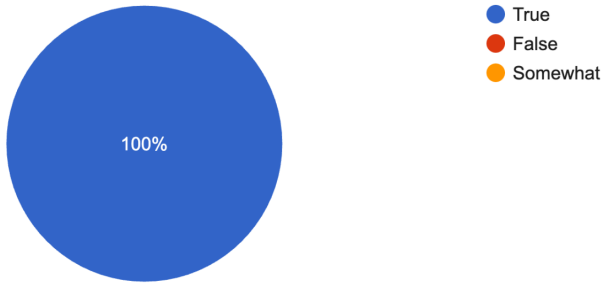

Fig 9: Question 8 Statistics

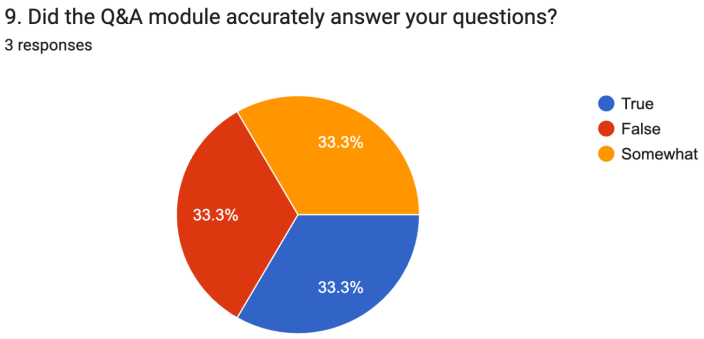

Fig 10: Question 9 Statistics

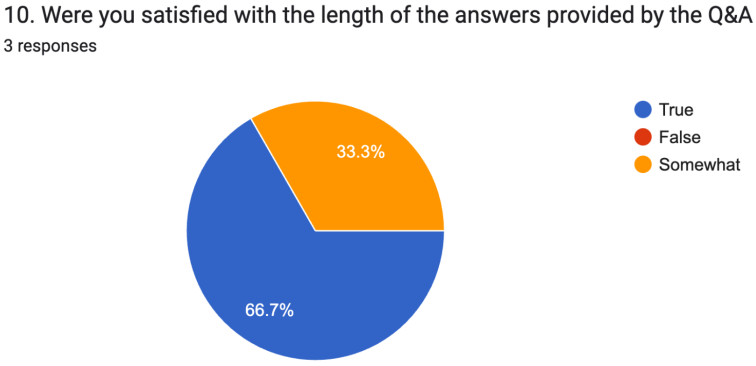

Fig 11: Question 10 Statistics

From our methods of evaluation and the results of the peer-based feedback process, we reached the following evaluation about the various modules of our project:

Module Name	Evaluation	Comments
User Interface	Very Satisfied	Easy to use and intuitive
Lecture Transcript	Very Satisfied	Transcript appeared accurate and similar to actual lecture content
Summary/Lecture Notes	Satisfied	Users found it useful, however since every user has a different learning style and prefers a different tone, their experiences varied
Q&A Module	Satisfied	Accurate answers provided for the most part

Conclusions

We have identified that the strengths of our approach lie in the way we generate the transcript and the paraphrased summary along with lecture notes. Our Q&A module is also quite strong, with participants finding relevant answers most of the time. The way we can improve it however is by using a better LLM model with more parameters such that it can understand the context better to provide better answers. This can include Google Gemini 2.5 Pro for example, or Llama 4 Scout.

A potential improvement we see towards our methodology is to follow an approach we discovered in the paper we referenced - HumSum: A Personalized Lecture Summarization Tool for Humanities Students Using LLMs, described in the introduction section. Our project could also take some inspiration from it and allow users to customize the notes and Q&A material they get. The HumSum paper introduced several personalization options for summarization, including customization of length, tone, complexity, and format. These features allowed users to tailor summaries to their individual learning preferences, significantly enhancing the learning experience. Recognizing that different individuals have different learning styles, this level of customization proved to be a valuable aspect of the tool. Inspired by this approach, we plan to add similar personalization features into StudyBuddy to improve its adaptability and effectiveness across a wider range of users.

In support of this enhancement, we conducted a survey with three users. As mentioned in the Methods section, Question 8 asked, “Would you prefer further personalization options for the generated summary?” All three participants responded True, as reported in the Results section.

This feedback, together with the implementation outlined in the HumSum project, highlights the importance of offering tailored summarization options.

Works cited

[1] Zahra Kolagar, Alessandra Zarcone (2024). HumSum: A Personalized Lecture Summarization Tool for Humanities Students Using LLMs

[2] Libraries and Licence

Library	License
Flask	BSD-3-Clause
Flask-CORS	MIT
OpenAI	MIT
LangChain	MIT
Hugging Face Hub	Apache-2.0
Sentence Transformers	Apache-2.0
Transformers	Apache-2.0
PyTorch	BSD-3-Clause
FAISS-CPU	MIT
Instructor Embedding	Apache-2.0
NLTK	Apache-2.0
SentencePiece	Apache-2.0
Tiktoken	MIT
Whisper	MIT
MoviePy	MIT
FFmpeg-python	Apache-2.0
imageio	BSD-2-Clause
NumPy	BSD-3-Clause
Altair	BSD-3-Clause

Python-dotenv	BSD-3-Clause
Requests	Apache-2.0
tqdm	MIT/MPL-2.0

Appendix

Appendix A: Lecture Summarization via Cosine Similarity and Transformers

```python
def generate_summary(title, document, num_sentences=5):
    sentences = nltk.sent_tokenize(document)

    if len(sentences) == 0:
        return "No valid sentences to summarize."

    title_embedding = model.encode(title, convert_to_tensor=True)
    sentence_embeddings = model.encode(sentences, convert_to_tensor=True)

    similarity_scores = util.pytorch_cos_sim(title_embedding, sentence_embeddings)[0]

    ranked_sentences = sorted(
        [(score.item(), sentence) for score, sentence in zip(similarity_scores, sentences)],
        key=lambda x: x[0],
        reverse=True
    )

    selected_sentences = [sentence for i, sentence in ranked_sentences[:num_sentences]]
    paraphrased_sentences = [paraphrase(sentence) for sentence in selected_sentences]

    print("\n=== Paraphrased Sentences ===")
    for idx, sent in enumerate(paraphrased_sentences, 1):
        print(f"{idx}. {sent}")

    if paraphrased_sentences:
        input_text = "summarize: " + " ".join(paraphrased_sentences)
        input_ids = tokenizer.encode(input_text, return_tensors="pt", max_length=2048,
truncation=True)
        outputs = t5_model.generate(
            input_ids,
            max_length=1024,
            num_beams=4,
            early_stopping=False,
            do_sample=True
        )
        summary = tokenizer.decode(outputs[0], skip_special_tokens=True).strip()

        print("\n=== Generated Summary ===")
        print(summary)
    else:
        summary = "Summary generation failed due to insufficient data."

    return summary
```

Fig 12: Sentence scoring for Paraphrasing and Summarization

The generate_summary() function performs automated lecture summarization by combining semantic sentence ranking with transformer-based paraphrasing and abstraction. It

begins by tokenizing the lecture transcript into individual sentences and then computes cosine similarity scores between each sentence and the lecture title using embeddings from the all-MiniLM-L6-v2 SentenceTransformer model. This semantic comparison ensures that the most relevant sentences are selected, rather than simply the first or longest ones. The top-ranked sentences are paraphrased using a T5 model to enhance fluency and variability, and the resulting content is passed once more through the T5 model to produce a final abstractive summary. This multi-stage approach improves alignment with the lecture topic and ensures the generated summary is both contextually relevant and linguistically coherent.

Appendix B: AI-Powered Note Generation

```python
def generate_notes(text):
    """Generate structured notes from text using OpenAI's GPT model"""
    try:
        # Create chat completion
        response = openai.ChatCompletion.create(
            model="gpt-4o-mini",
            messages=[
                {"role": "system", "content": "You are an AI that generates detailed, structured, and
accurate lecture notes from transcriptions. Minimum 2-3 page response is required. The format must be
markdown that can be embedded into a website. Add proper line breaks and bullet points for lists,
subtopics, and lines to look it good. You may add information that is not present in the transcription,
but ensure it is relevant and accurate."},
                {"role": "user", "content": f"Generate detailed and structured lecture notes from the
following transcription:\n{text}\n\nPlease follow these guidelines:\n- Organize the notes into clear
sections (e.g., Introduction, Key Concepts, Examples, Summary).\n- Include definitions, explanations,
and key points made by the lecturer.\n- Ensure the notes are comprehensive, accurate, and coherent.\n-
Break down complex ideas into simpler terms.\n- Use bullet points for lists and subtopics.\n- If
possible, highlight any key takeaways or important conclusions.\n- Maintain the authenticity of the
information provided in the transcription."}
            ]
        )

        # Extract the generated notes from the response
        notes = response.choices[0].message['content']

        return notes
    except Exception as e:
        print(f"Error generating notes: {str(e)}")
        return str(e)
```

Fig 13: Automated Note Generation using LLM

The generate_notes() function, shown above, automates the transformation of raw lecture transcripts into structured and coherent study notes using OpenAI's GPT-4o mini language model. This function accepts the full transcription text along with the corresponding summary as input, and connects to the OpenAI API using proper authentication credentials. It then sends a carefully crafted prompt that includes formatting instructions, such as requests for section headings, bullet points, and concise phrasing, to guide the model's output. The AI-generated response is parsed and returned in markdown format to ensure readability and structure.

To maintain reliability, the function also incorporates basic error handling to catch and report any issues encountered during communication with the API. This automated process replaces the traditionally manual task of organizing lecture content into readable notes, thereby streamlining the workflow and enhancing the efficiency of note generation.