Smart Subtitle Generator

Rapport Final

Rédigé par :

Ammar KASBAOUI
Ilias ISSAF

Supervisé par :

Hakim Hafidi, Yasser Aderghal, Hamza Gamouh

Université Internationale de Rabat
Salé, Morocco

2024

Contents

Chapter 1

Introduction

1.1 Context and Challenges

In today's digital landscape, video content has become a dominant medium for communication, education, and entertainment. However, engaging effectively with video content often presents significant challenges. These include language barriers, a lack of accessibility for individuals with hearing impairments, and difficulties in recalling or retrieving specific details from lengthy videos. Such challenges can greatly limit accessibility and productivity, particularly for non-native speakers and individuals with disabilities.

1.2 Objective

This project aims to address these challenges by developing an AI-driven application named the Smart Subtitle Generator. The application is designed to enhance the video consumption experience by offering two key functionalities: automated subtitle generation and interactive querying of video content. By leveraging advanced technologies such as speech-to-text processing, multilingual translation, and retrieval-augmented generation (RAG), paired with a vectorial database, the application ensures both usability and accessibility for a wide range of users thanks to an interactive graphical user interface.

1.3 Report Structure

This report provides a detailed account of the methodologies, architectural designs, and implementation steps employed in creating the Smart Subtitle Generator. It begins with an analysis of the background and project requirements, followed by an in-depth discussion of system design and development. The final sections evaluate the results, assess the system's performance, and propose avenues for future enhancement.

1.4 Significance

This introduction highlights the relevance and importance of the Smart Subtitle Generator in addressing critical issues in video accessibility and interaction. By leveraging AI technologies, the project aspires to create a tool that makes video content more inclusive and practical for diverse audiences, particularly in academic and professional contexts. The application promises to serve as a bridge between accessibility and functionality.

Chapter 2

Background

2.1 Contextual Foundation

The Smart Subtitle Generator project is rooted in cutting-edge advancements in artificial intelligence, particularly in natural language processing (NLP) and computer vision. As video content increasingly dominates communication, education, and entertainment, the demand for tools that enhance accessibility and usability has grown exponentially. This project integrates multiple AI technologies to address these needs effectively, providing an innovative approach to subtitle generation and content querying.

2.2 Relevant Prior Work

Significant progress has been made in developing technologies that improve video accessibility:

1. **Speech-to-Text Conversion:** Platforms like YouTube and Zoom employ speech-to-text systems powered by models such as Google's Speech API and OpenAI's Whisper. These systems form the foundation of automated subtitle generation by transcribing spoken language into text.
2. **Multilingual Translation:** NLP models, including those from Hugging Face and Helsinki-NLP, have advanced real-time text translation capabilities. These tools are essential for generating multilingual subtitles, enabling global accessibility.
3. **Retrieval-Augmented Generation (RAG):** RAG systems combine embeddings stored in vector databases with generative language models, enabling precise and context-aware queries. This allows users to interact with video content dynamically and effectively.

2.3 Challenges in Existing Solutions

Despite advancements, existing tools still face notable limitations:

- **Accuracy:** Variability in accents, background noise, and specialized vocabulary can challenge transcription systems.

- **Real-Time Performance:** Many systems lack the capacity for seamless, real-time multilingual processing.
- **Interactivity:** Few solutions support direct, interactive querying of video content, limiting user engagement.

2.4 Project Differentiation

The Smart Subtitle Generator stands out by addressing these challenges through:

1. **Advanced AI Models:** Utilizing OpenAI Whisper for transcription, Helsinki-NLP for translation, and vector databases for efficient content querying.
2. **Integrated Pipeline:** Seamlessly combining subtitle generation and interactive querying into a cohesive application.
3. **User-Centric Design:** Prioritizing accessibility and usability to deliver an inclusive and functional user experience.

This chapter underscores the project's relevance within the broader context of AI advancements and accessibility solutions. By tackling existing challenges and leveraging state-of-the-art technologies, the Smart Subtitle Generator sets the stage for an innovative tool designed to meet the needs of diverse audiences. The following chapters will delve deeper into the design, implementation, and evaluation of this application.

Chapter 3

Requirements Capture

3.1 Introduction

The development of the Smart Subtitle Generator is built upon the foundational insights from previous analyses, focusing on resolving critical accessibility challenges in video content. This chapter delves into the systematic process of identifying and capturing user requirements to ensure that the application's design and functionalities align with practical needs. By clearly defining necessary and desirable features, this phase establishes a roadmap for the subsequent stages of development.

3.2 Identifying User Needs

The core objective of the Smart Subtitle Generator is to overcome key obstacles in video accessibility. To achieve this, a detailed analysis of the target user groups and their requirements was conducted:

1. **Non-Native Speakers:** Require accurate multilingual subtitles to better understand content in unfamiliar languages.

2. **Individuals with Hearing Impairments:** Depend on precise and timely subtitles to access auditory information effectively.

3. **General Content Consumers:** Seek efficient tools to retrieve specific information from videos without the need for exhaustive rewatching.

These insights have informed the functional and non-functional requirements, ensuring that the solution is comprehensive and user-focused.

3.3 Functional Requirements

The project defines several key functionalities to address user needs effectively:

1. **Automated Subtitle Generation:**

    - Accurate transcription of audio content into text.
    - Compatibility with various video formats and seamless audio extraction.

2. **Multilingual Translation:**
    - Support for a wide range of target languages to cater to global audiences.
    - Contextually accurate translations that retain the original meaning.
3. **Interactive Content Querying:**
    - Capability to dynamically query specific video content using natural language.
    - Context-aware responses generated through advanced AI embeddings.

3.4 Non-Functional Requirements

Beyond core functionalities, the application must meet several additional criteria to ensure reliability and usability:

1. **Performance:**
    - Real-time processing for transcription, translation, and queries.
    - Low latency to provide users with a seamless experience.
2. **Scalability:**
    - Capacity to support large datasets and multiple concurrent users.
    - A modular architecture to accommodate future expansions and improvements.
3. **Accessibility:**
    - Intuitive and user-friendly interface design to accommodate users with varying technical expertise.
    - Full compliance with accessibility standards to support individuals with disabilities.

3.5 Conclusion

The requirements capture phase serves as a critical step in aligning the project's objectives with user expectations. By addressing both functional and non-functional needs, the Smart Subtitle Generator is designed to deliver a reliable and inclusive solution. These requirements guide the design and implementation processes, ensuring the final application achieves its intended impact and functionality. The next chapter will elaborate on the architectural and design choices made to fulfill these objectives.

Chapter 4

Analysis and Design

4.1 Introduction

Building on the requirements established in the previous chapter, this section presents an in-depth analysis of the Smart Subtitle Generator's system architecture and design. This chapter aims to articulate the key architectural decisions, design principles, and modifications encountered during the development process. It also highlights the rationale for significant design choices and explores how the system evolved to address emerging challenges.

4.2 System Architecture Overview

The architecture of the Smart Subtitle Generator is designed to be modular, scalable, and maintainable, integrating AI technologies to achieve its objectives. The system consists of the following primary components:

1. **Frontend Interface:**
   - A dynamic web-based interface built with Flask, offering intuitive and seamless user interactions.
   - Features include video or URL upload options, language selection for subtitles, and processed video download capabilities.
2. **Backend Services:**
   - Implements core functionalities such as video processing, speech transcription, multilingual translation, and content-based embedding queries.
   - Integrates AI APIs like OpenAI Whisper for transcription and Helsinki-NLP for translation.
3. **Database Layer:**
   - A vector database is employed for storing and retrieving embeddings, enabling efficient and contextually accurate queries.
4. **Processing Pipeline:**

- Sequentially handles tasks including video download, audio extraction, transcription, translation, and embedding subtitles.
- Leverages robust tools such as ffmpeg for media processing and yt-dlp for video downloading.

4.3 Evolution of Design

The system design underwent significant evolution, driven by practical challenges:

1. **Initial Vision:**
    - Initially conceptualized as a desktop application, the design was pivoted to a web-based solution to enhance accessibility and reach.
2. **Challenges and Adaptations:**
    - *Latency in Real-Time Processing:* Early iterations encountered delays related to models loading time; this issue was handled by storing the models in cache memory.
    - *Language Expansion:* The need for extensive multilingual support led to the integration of dynamic model-loading capabilities.
3. **Design Trade-offs:**
    - The primary trade-off involved limited computational resources and storage. As a result, the application had to be scaled down to a single-user application, focusing on optimizing performance and usability within these constraints.

4.4 Key Design Choices

1. **Modularity:**
    - A modular approach ensures that components can be developed and tested independently, enabling easier maintenance and future enhancements.
    - This design facilitates adding new languages, expanding features, or integrating advanced AI models without significant architectural changes.
2. **User-Centric and Accessible Design:**
    - The frontend design emphasizes simplicity and responsiveness, ensuring usability across diverse user groups.
3. **Adoption of Open-Source Tools:**
    - Leveraging open-source technologies such as Whisper, Helsinki-NLP, and ffmpeg minimized development costs while delivering robust and reliable performance.

4.5 Conclusion

The design of the Smart Subtitle Generator system reflects a careful balance between functionality, performance, and user accessibility. Despite inevitable trade-offs, the chosen architecture and design principles align with the project's overarching goals. This modular and adaptable system lays a strong foundation for ongoing improvements and feature expansions. The next chapter will detail the implementation phase, elucidating how these design decisions were brought to life in practice.

Chapter 5

Implementation

5.1 Introduction

The implementation of the Smart Subtitle Generator translates the system design into a functional application, focusing on realizing its core features through efficient coding practices and AI integration. This chapter highlights the key components, interesting aspects, and challenges encountered during the development process. While complete code listings are omitted for brevity, annotated explanations of significant parts are provided to clarify the logic and decision-making.

5.2 Key Components of Implementation

5.2.1 Video Processing Module

The video processing module serves as the backbone of the application, responsible for handling input videos and preparing them for subsequent tasks:

- **Audio Extraction:** Utilizing `ffmpeg` to extract audio from video files efficiently while maintaining quality.
- **Video Download:** Employing `yt-dlp` to support URL-based video input, ensuring compatibility with various platforms like YouTube.

Code Highlight: Audio Extraction Function

```
import ffmpeg

def extract_audio(video_file_path, audio_output_path="audio.mp3"):
    try:
        ffmpeg.input(video_file_path).output(audio_output_path).run()
        print(f"Audio extracted successfully: {audio_output_path}")
        return audio_output_path
    except ffmpeg.Error as e:
        print(f"Error extracting audio: {e}")
        return None
```

This function demonstrates robust handling of audio extraction, ensuring compatibility with diverse video formats.

5.2.2 Speech-to-Text Transcription

Transcription is powered by OpenAI Whisper, chosen for its accuracy and ability to handle various languages and accents:

- **Model Integration:** The pre-trained Whisper model for transcription tasks and translation to English.
- **Timed Segments:** Transcription results are broken into time-stamped segments for precise subtitle synchronization.

5.2.3 Multilingual Translation

Translation tasks leverage Helsinki-NLP's OPUS-MT models:

- **Dynamic Model Loading:** Language-specific models are loaded on-demand to optimize memory usage.

5.2.4 Subtitle Integration

Subtitles are generated in `.srt` format and merged with the original video using `ffmpeg`:

- **Formatting:** Subtitles include sequential numbering, time ranges, and translated text.
- **Embedding Subtitles:** The `add_subtitles` function embeds subtitles directly into video files for end-user convenience.

5.2.5 Interactive Querying

Queries are managed through a combination of vector embeddings and retrieval-augmented generation (RAG):

- **Embedding Storage:** A vector database (e.g., Pinecone) stores video embeddings for efficient retrieval.
- **Dynamic Queries:** User inputs are converted into embeddings and compared against stored vectors to retrieve relevant information through similarity search.

5.3 Challenges and Solutions

- **Real-Time Performance:**
  - Initial latency in transcription and translation was mitigated through loading models from cache rather than downloading them repeatedly.

5.4 Visualizing the Implementation

Below is a simplified block diagram illustrating the logical flow of the application:

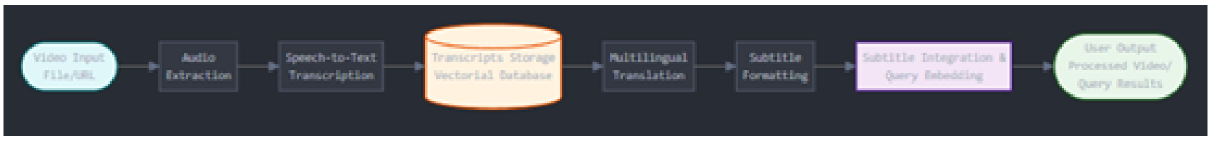


Figure 5.1: Block Diagram

5.5 Conclusion

The implementation phase successfully realized the key functionalities of the Smart Subtitle Generator, overcoming challenges through robust design and efficient coding practices. The next chapter will evaluate the system's performance, providing insights into its effectiveness and areas for improvement.

Chapter 6

Results and Discussion

6.1 Introduction

This chapter presents the outcomes of the Smart Subtitle Generator project, focusing on the functionalities achieved and the challenges overcome during its development. It discusses the system's capabilities in meeting its objectives and highlights its practical applications. Visual aids, such as screenshots, can be included to better illustrate key aspects of the implementation.

6.2 Functional Achievements

6.2.1 Subtitle Generation

- **Accuracy:** The system provides precise transcription of video audio into text, leveraging advanced AI models to handle varied accents and speaking speeds.
- **Multilingual Support:** Subtitles are generated in multiple languages, enhancing accessibility for a diverse audience.

6.2.2 Interactive Querying

- **Contextual Relevance:** The querying module retrieves meaningful, context-aware responses from video content using embedding-based search techniques.
- **Dynamic Search:** Users can interact with video data seamlessly, asking specific questions and receiving accurate responses.

6.2.3 User Interface

- **Usability:** The web-based interface is intuitive and responsive, allowing users to easily upload videos, select subtitle languages, and download processed content.
- **Real-Time Feedback:** While processing videos, the interface provides real-time status updates to improve the user experience.

6.3 Visual Demonstrations

6.3.1 Subtitle Generator

Test Case 1: Upload a Video

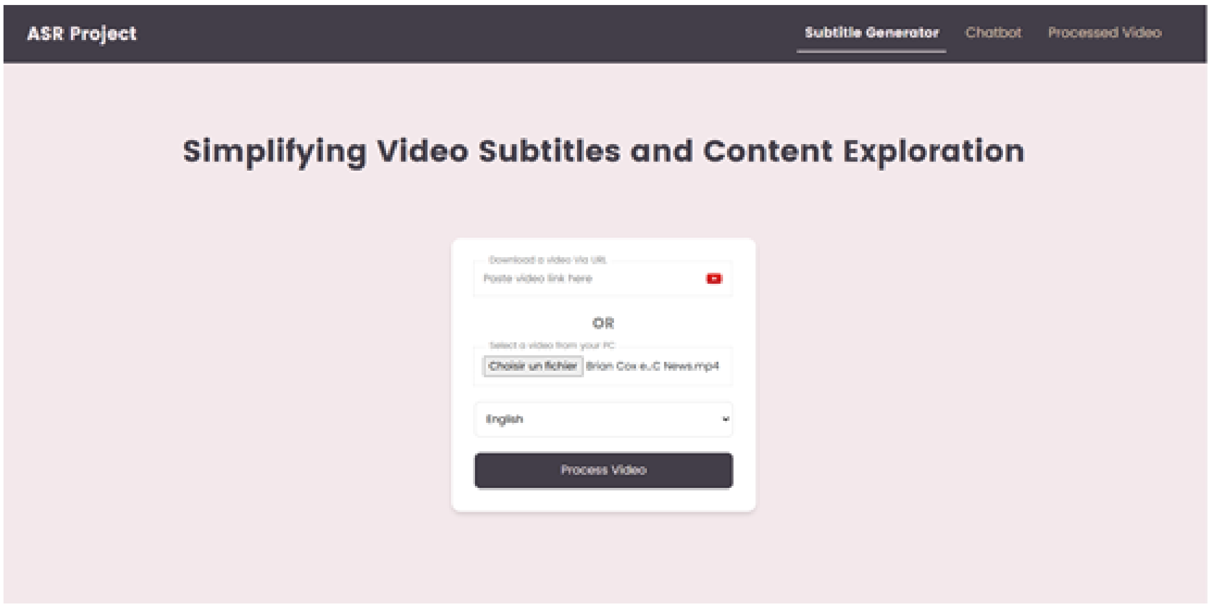


Figure 6.1: Uploading a video file in the application.

Output:

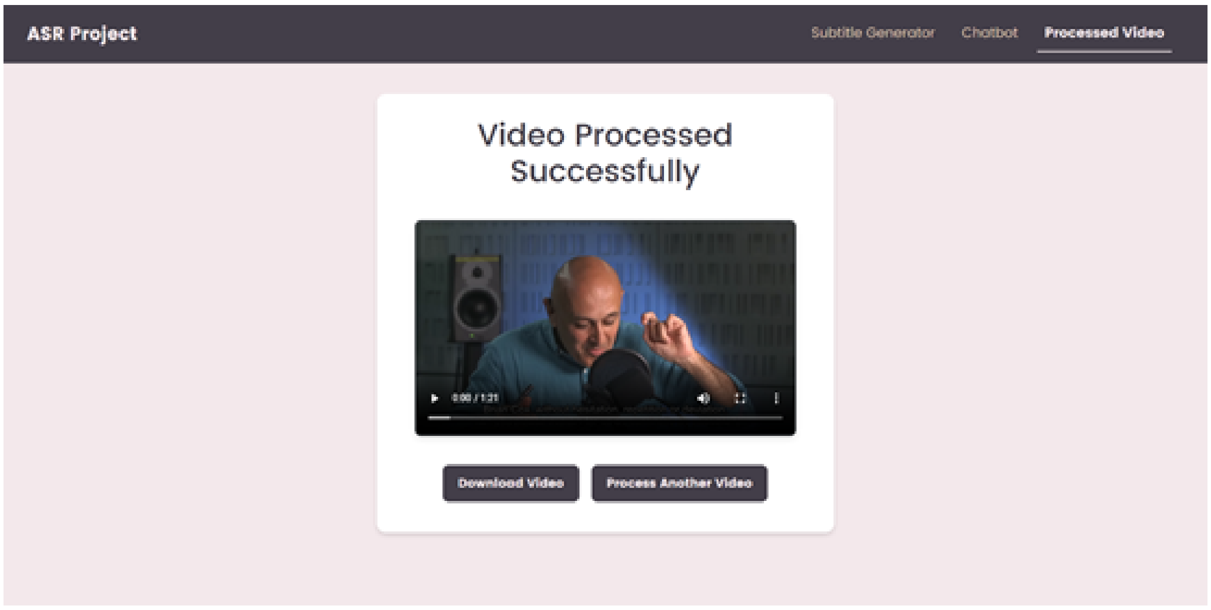


Figure 6.2: Example of generated subtitles.

Test Case 2: Download Video via URL

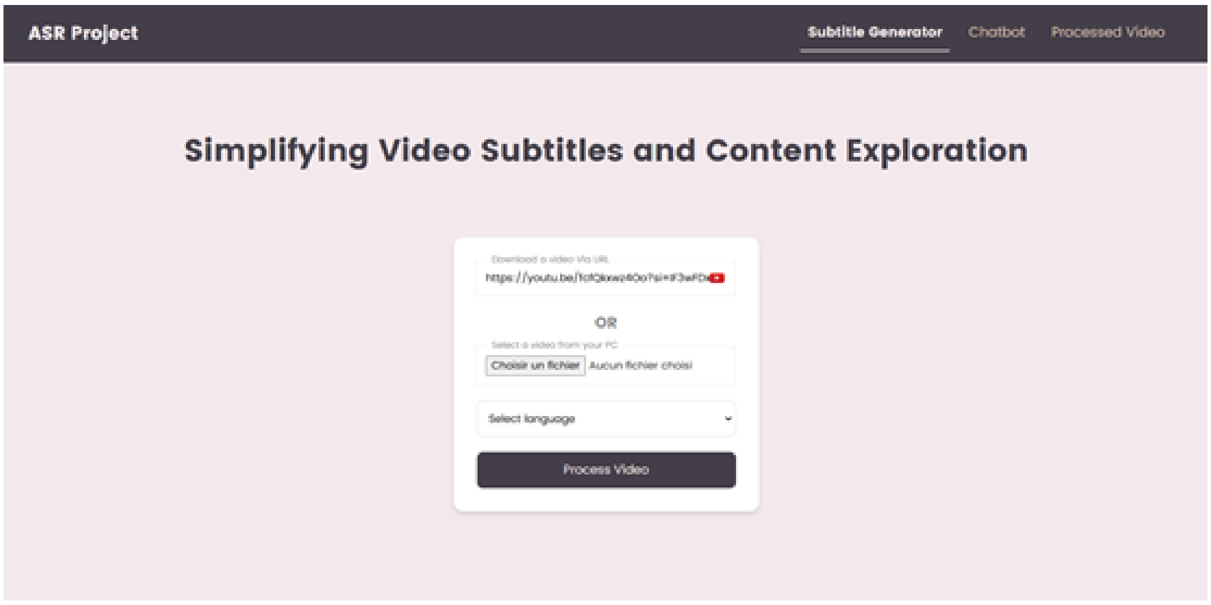


Figure 6.3: Downloading a video via URL.

Output:

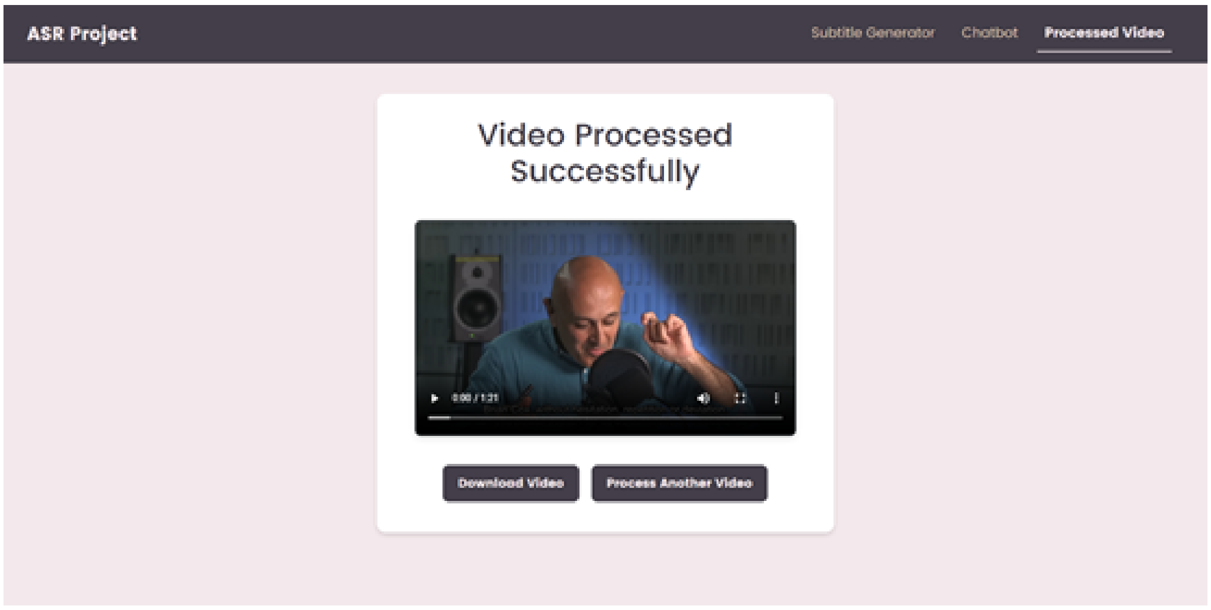
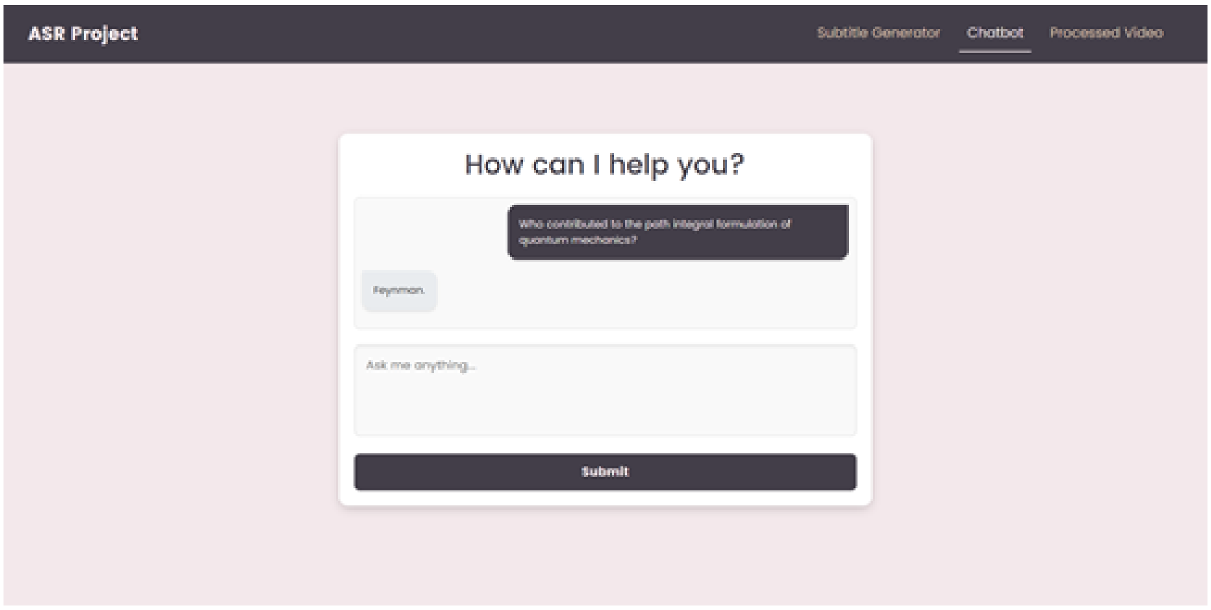


Figure 6.4: Video downloaded with integrated subtitles.

6.3.2 Chatbot

Test Case 3: Single Prompt Submission

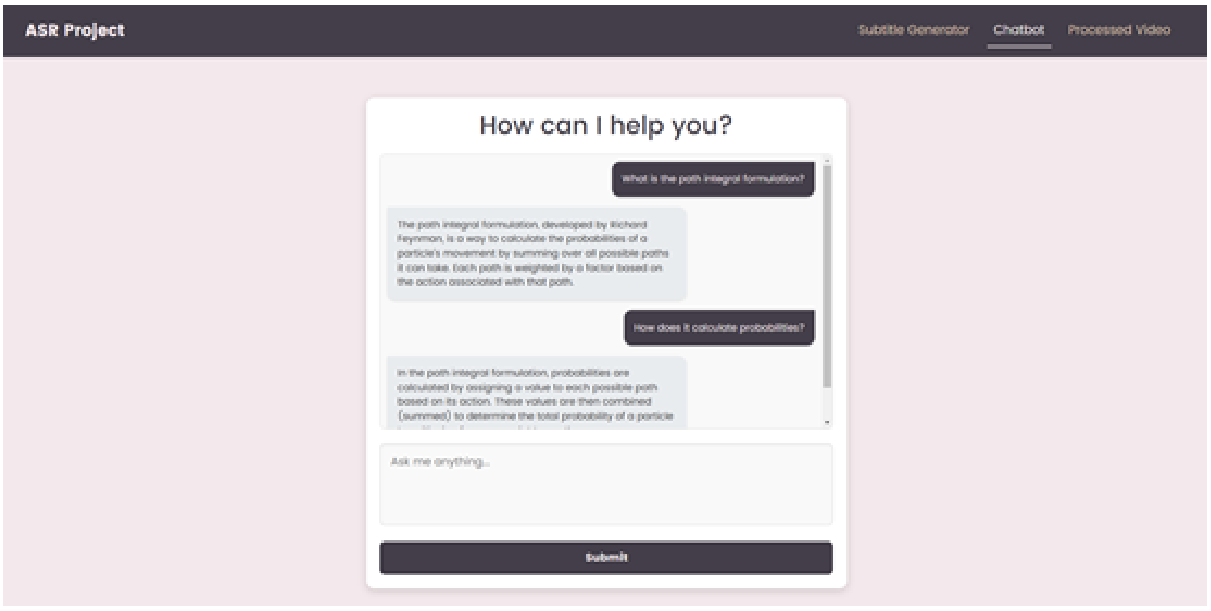


Who contributed to the path integral formulation of quantum mechanics?

Feynman.

Figure 6.5: Single prompt submission in the chatbot.

Test Case 4: Multi-turn Conversation



What is the path integral formulation?

The path integral formulation, developed by Richard Feynman, is a way to calculate the probabilities of a particle's movement by summing over all possible paths it can take. Each path is weighted by a factor based on the action associated with that path.

How does it calculate probabilities?

In the path integral formulation, probabilities are calculated by assigning a value to each possible path based on its action. These values are then combined (summed) to determine the total probability of a particle

Figure 6.6: Example of a multi-turn conversation.

6.3.3 User Interface Testing

Navbar Navigation

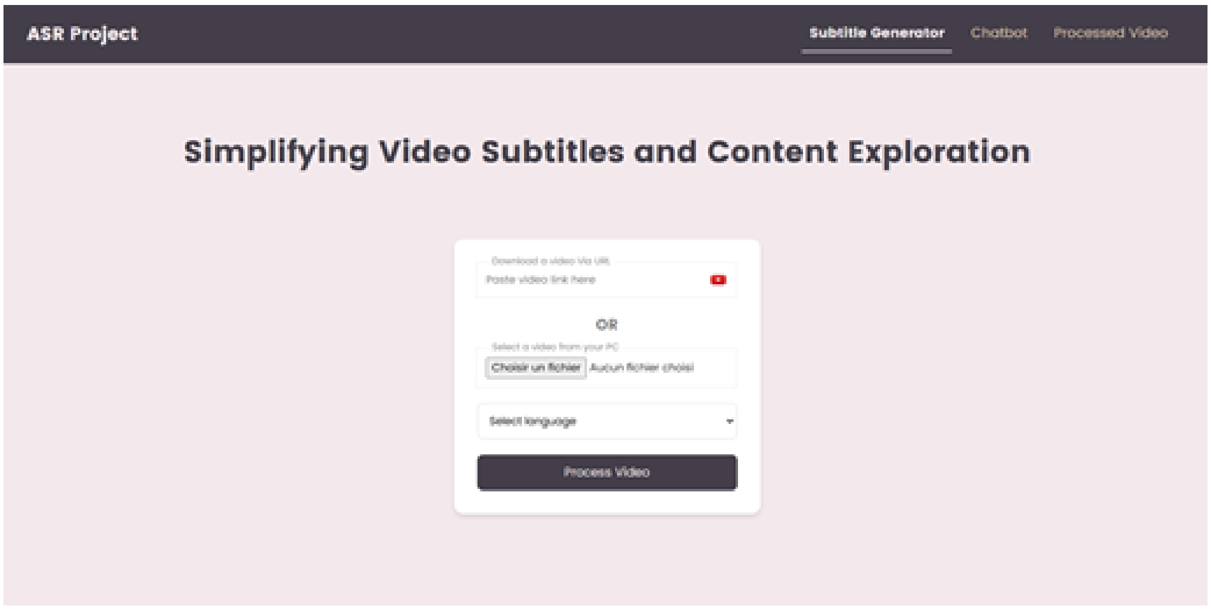


Figure 6.7: Subtitle Generator.

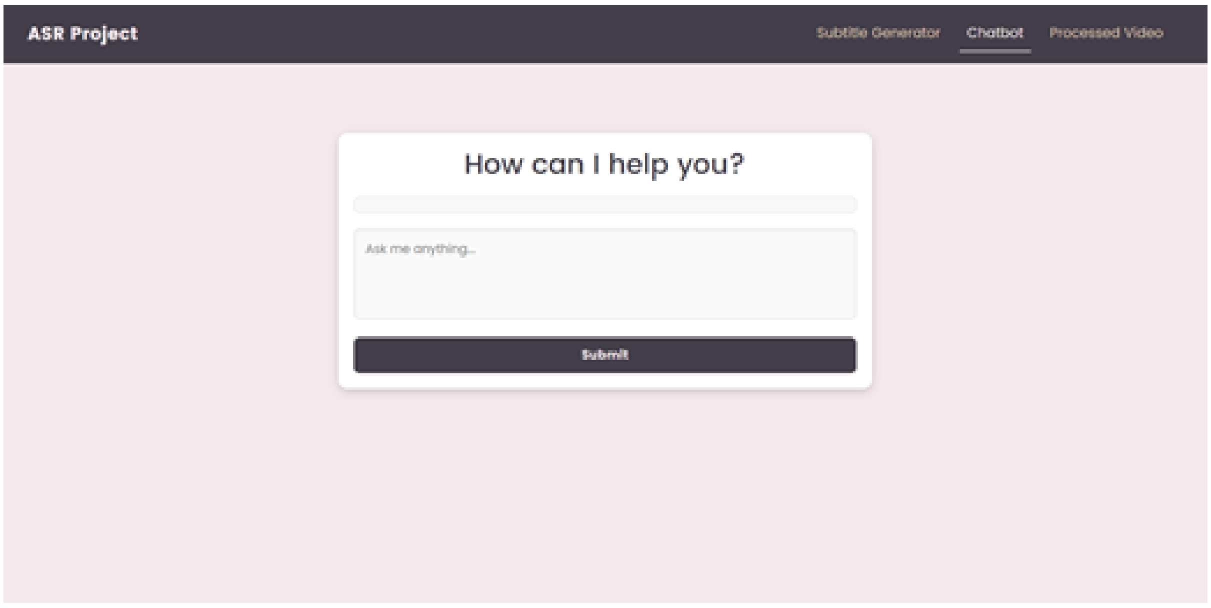


Figure 6.8: Chatbot.

Loading Screens

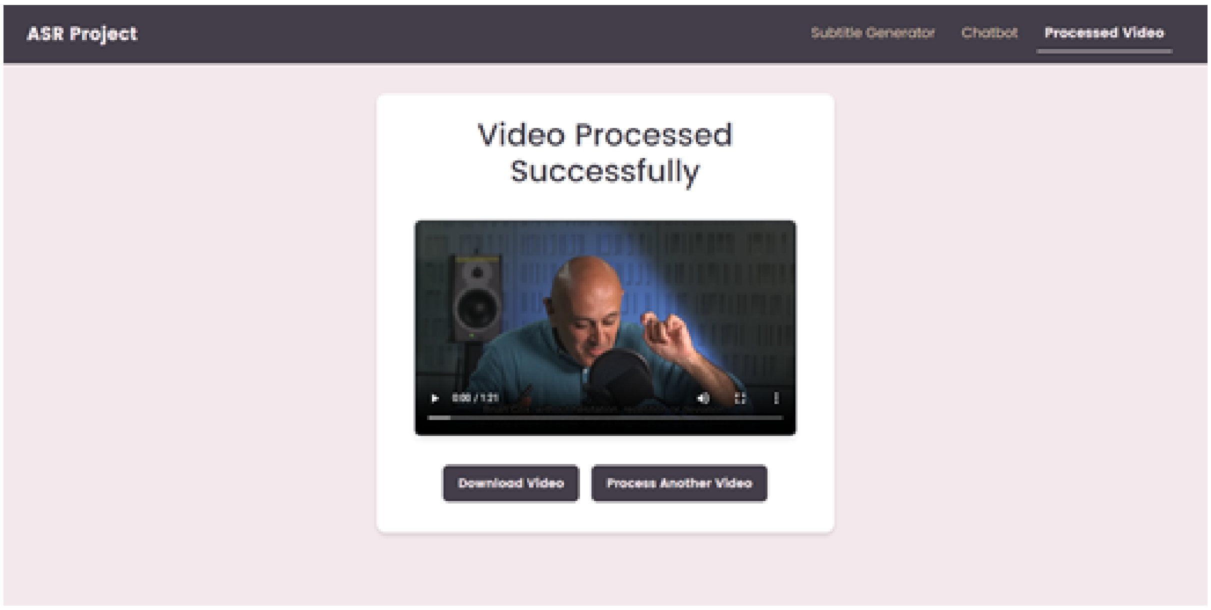


Figure 6.9: Processed Video.

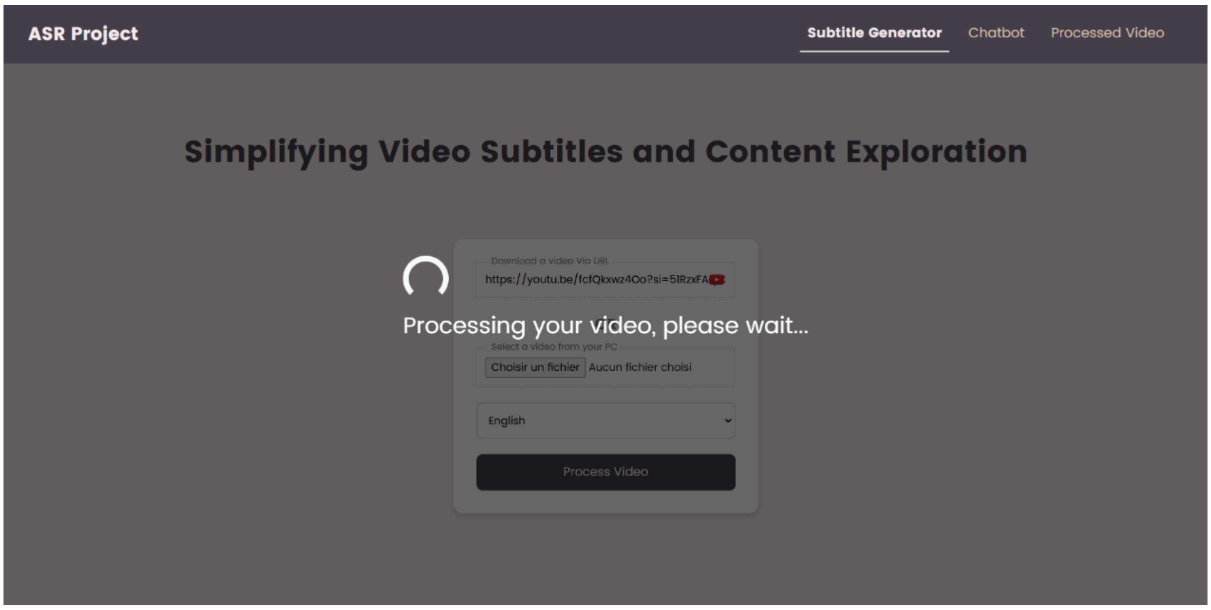

ASR Project
Subtitle Generator
Chatbot
Processed Video
Simplifying Video Subtitles and Content Exploration
Download a video Via URL
https://youtu.be/fcfQkxw24Oo?si=5lRzxFA
Processing your video, please wait...
Select a video from your PC
Choisir un fichier
Aucun fichier choisi
English
Process Video

Figure 6.10: Loading screen.

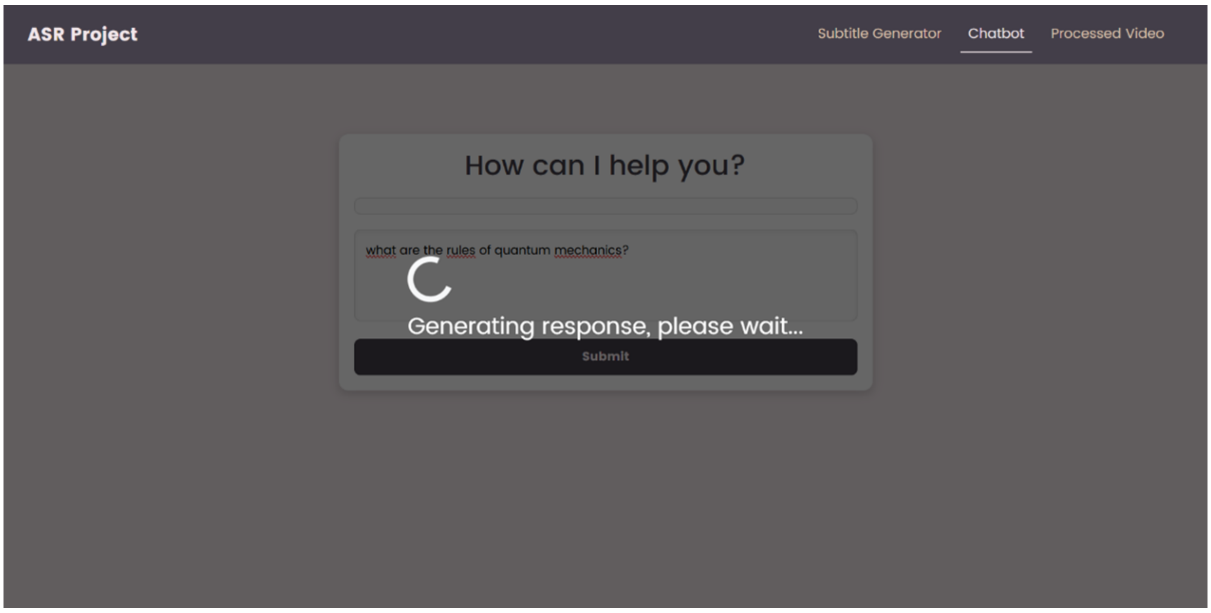


Figure 6.11: Loading screen.

6.4 Conclusion

The Smart Subtitle Generator has successfully delivered its intended functionalities, addressing core challenges in video accessibility. Through accurate transcription, effective translation, and an intuitive interface, the system provides a robust tool for users to engage with video content in a meaningful way. These results lay a solid foundation for future enhancements and broader applications. The next chapter will focus on evaluating the system's performance in various use cases and scenarios.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

The Smart Subtitle Generator project has successfully addressed its primary objectives by providing a robust solution to key challenges in video accessibility. The application integrates advanced AI technologies to deliver automated subtitle generation, multilingual translation, and interactive querying features. These functionalities enhance the video consumption experience, making it more inclusive and user-friendly for diverse audiences.

7.2 Key Achievements

1. **Automated Subtitle Generation:**
    - Precise transcription of video audio into text using advanced speech-to-text technologies.
    - Seamless generation of subtitles across multiple languages.
2. **Interactive Content Querying:**
    - Efficient and context-aware querying enabled by a vector database and embedding-based retrieval.
    - Dynamic and intuitive interactions for users to extract specific information from video content.
3. **User-Centric Design:**
    - Intuitive web-based interface ensuring ease of use and accessibility.
    - Real-time feedback during video processing to improve user experience.

7.3 Challenges Overcome

Some limitations were encountered, primarily related to resource constraints and scalability. The system was developed as a single-user application due to limited computational power, and some low-priority features, such as advanced error correction, were deferred.

7.4 Future Work

To build on the foundation laid by this project, several opportunities for future enhancements have been identified:

1. **Scalability:**
   - Expand the system's architecture to support multiple concurrent users.
   - Deploy the application on cloud-based infrastructure for improved performance and availability.
2. **Enhanced Features:**
   - Integrate advanced error correction mechanisms for transcription and translation.
   - Add support for additional languages and domain-specific vocabularies.
3. **User Feedback Integration:**
   - Implement a feedback loop to gather user input and improve system functionalities iteratively.
4. **Performance Optimization:**
   - Further reduce processing times by adopting more efficient AI models and task-handling techniques.
5. **Broader Accessibility:**
   - Develop mobile and desktop versions of the application to reach a wider user base.
   - Incorporate features tailored to specific user groups, such as individuals with visual impairments.

7.5 Final Remarks

The Smart Subtitle Generator has demonstrated the potential of AI-driven solutions in addressing real-world accessibility challenges. While there is room for growth and enhancement, the project provides a solid framework for future developments. By continuing to refine and expand its capabilities, the application can make significant contributions to improving video accessibility and user engagement in both academic and professional contexts.

Bibliography

[1] Brown, T., et al. (2021). *Language Models are Few-Shot Learners.* Proceedings of the Advances in Neural Information Processing Systems (NeurIPS). Available at: https://openai.com/research/whisper

[2] Tiedemann, J., Thottingal, S. (2020). *OPUS-MT: Building Open Translation Services for the World.* Proceedings of the 22nd Annual Conference of the European Association for Machine Translation. Available at: https://huggingface.co/Helsinki-NLP

[3] FFmpeg Developers. (2023). *FFmpeg: A Complete, Cross-Platform Solution to Record, Convert and Stream Audio and Video.* Available at: https://ffmpeg.org

[4] yt-dlp Contributors. (2023). *yt-dlp: A Youtube-dl Fork with Additional Features and Fixes.* Available at: https://github.com/yt-dlp/yt-dlp

[5] OpenAI API. *OpenAI. (2023). "OpenAI API Documentation."* Available at: https://platform.openai.com/docs/

Appendix A

Code Embed

This appendix provides key code excerpts demonstrating the implementation of core functionalities in the Smart Subtitle Generator. These snippets focus on initializing the embedding system, generating and storing embeddings, performing similarity searches, and processing prompts with contextual enhancements using a language model.

A.1 Initializing Pinecone

The following code initializes Pinecone for managing vector embeddings and sets up the namespace and index used for storing embeddings.

```
from pinecone.grpc import PineconeGRPC as Pinecone
pc = Pinecone(api_key="YOUR_API_KEY")
index_name = "transcript-index"
namespace = "transcript-namespace"
```

This setup is foundational for interacting with the Pinecone vector database, ensuring proper organization and retrieval of embeddings.

A.2 Generating and Storing Embeddings

This function generates embeddings for a given text input and stores them in the Pinecone index.

```
def generate_and_store_embedding(text):
    embedding = pc.inference.embed(
        model="multilingual-e5-large",
        inputs=[text],
        parameters={"input_type": "passage", "truncate": "END"}
    )

    index = pc.Index(index_name)
    record = {
        "id": f"vec-{int(time.time())}",
        "values": embedding[0].values,
        "metadata": {"text": text}
    }
    index.upsert(vectors=[record], namespace=namespace)
```

This function integrates a multilingual embedding model and uses Pinecone's API to store the generated embeddings alongside metadata for later retrieval.

A.3 Similarity Search

The following code performs a similarity search to retrieve the most relevant embeddings based on a query.

```
def similarity_search(query_text, top_k=1):
    query_embedding = pc.inference.embed(
        model="multilingual-e5-large",
        inputs=[query_text],
        parameters={"input_type": "query"}
    )

    index = pc.Index(index_name)
    results = index.query(
        namespace=namespace,
        vector=query_embedding[0].values,
        top_k=top_k,
        include_metadata=True
    )
    return results['matches']
```

This function converts the input query into an embedding and searches the Pinecone index to retrieve the top relevant matches, including their metadata.

A.4 Processing Prompts with Context

This function integrates the similarity search results with a language model to enhance responses with relevant contextual information.

```
def process_prompt(prompt):
    similar_content = similarity_search(prompt)
    retrieved_texts = [match["metadata"]["text"] for match in similar_content]
    context = "\n".join(retrieved_texts)

    enhanced_prompt = f"""
    ### Context:
    {context}
    ### Question:
    {prompt}
    ### Answer:
    """
    response = generate_response(enhanced_prompt)
    return response
```

This function retrieves similar content from the Pinecone index, concatenates it into a context section, and uses it to construct a prompt for a language model, thereby enhancing the relevance and accuracy of the generated response.

A.5 Summary

These code snippets illustrate essential steps in embedding management and their integration with a language model. By combining these functionalities, the Smart Subtitle Generator achieves efficient storage, retrieval, and context-aware response generation. For further details, refer to the full codebase in the project's repository.