

A MAJOR PROJECT REPORT on
AI-Powered Content Generation & Speech Synthesis with RAG and TTS
Submitted to Manipal University Jaipur
Towards the partial fulfillment for the Award of the Degree of
MASTER OF COMPUTER APPLICATIONS
2023-2025

by
Jagriti
23FS20MCA00023



MANIPAL UNIVERSITY
JAIPUR

Department of Computer Applications
School of AIML, IoT&IS, CCE, DS and Computer Applications
Faculty of Science, Technology and Architecture
Manipal University Jaipur
Jaipur-303007
Rajasthan, India

May 2025

DEPARTMENT OF COMPUTER APPLICATION
MANIPAL UNIVERSITY JAIPUR, JAIPUR-303007 (RAJASTHAN) INDIA

Date- 16-05-2025

CERTIFICATE

This is to certify that the project titled **AI-Powered Content Generation & Speech Synthesis with RAG and TTS** is a record of the Bonafide work done by **Jagriti (Reg No. 23FS20MCA00023)** submitted in partial fulfilment of the requirements for the award of the Degree of MASTER OF COMPUTER APPLICATIONS of **Manipal University Jaipur**, during the academic year **2023-2025**.

Dr. Devershi Pallavi Bhatt

*Professor & Project Guide, Dept of Computer Application
Manipal University Jaipur*

Dr. Shilpa Sharma

*HOD, Dept of Computer Application
Manipal University Jaipur*



CLIPO AI PRIVATE LIMITED
Regd. Address: B32/48 K-2-A-11 3RD FLOOR, KUBER HOUSE, SAKET NAGAR,
Durgakund, Varanasi- 221005, Uttar Pradesh
CIN: U62099UP2024PTC200261
Email: contact@clipo.pro | mobile: 9935666589

May 20, 2025

To Whomsoever It May Concern

This is to certify that **Ms. Jagriti** has successfully completed her internship with **Clipo AI Private Limited** as a **Data Science Intern** during the period from **January 20, 2025 to May 20, 2025**.

During this time, she worked on AI and machine learning projects, particularly focusing on **Text-to-Speech (TTS)** and **Auto-Reframe systems**. Her contributions included developing and fine-tuning TTS models, implementing vision-based reframing logic for video content, and performing prompt engineering to enhance AI model performance. She also handled preprocessing and analysis of large datasets to support these applications effectively.

Throughout her internship, Ms. Jagriti consistently demonstrated a high degree of **responsibility**, **dedication**, and **eagerness to learn**. Her strong **analytical thinking**, **collaborative spirit**, and **attention to detail** made her a valuable member of the team.

We wish her continued success and all the very best in her future endeavors.

Yours Sincerely,
For Clipo AI Private Limited

Abhishek Rai
Chief Executive Officer

ACKNOWLEDGMENTS

I would like to express my sincere and heartfelt gratitude to my project guide, **Dr. Devershi Pallavi Bhatt, Professor, Department of Computer Applications, Manipal University Jaipur**, for her invaluable guidance, constant encouragement, and unwavering support throughout the course of this project, *AI-Powered Content Generation & Speech Synthesis with RAG and TTS*. Her mentorship, technical insights, and continuous motivation played a crucial role in the successful completion of this work.

I am also deeply thankful to **Dr. Shilpa Sharma, Head of the Department, Department of Computer Applications, Manipal University Jaipur**, for providing a supportive academic environment and consistent encouragement during the entire duration of the project.

I extend my gratitude to **Dr. Amit Hirawat, Department of Computer Applications, Manipal University Jaipur** whose academic expertise and inputs enriched my understanding and approach to the project.

I am especially thankful to the team at Clipo AI, where I had the privilege of interning during this project. I am especially thankful to the team at Clipo AI, where I had the privilege of interning during this project. I am grateful to **Mr. Abhisek Rai**, my supervisor at Clipo AI, for his expert guidance, continuous motivation, and trust in my abilities. His insights on AI-powered voice synthesis and video intelligence significantly shaped the direction of my work. I also thank my colleagues and fellow interns for their constant collaboration and feedback, which helped me improve and grow professionally.

Furthermore, I thank my peers and the **Department of Computer Applications** for providing the necessary resources and a conducive environment to carry out this project effectively.

This project has been a significant learning experience, and I am truly grateful to all those who guided and supported me throughout this journey.

Thank you.

Submitted By:
Jagriti

(Reg No. 23FS20MCA00023)

ABSTRACT

In today's fast-paced digital content landscape, the demand for AI-powered tools that streamline audio and video production is rapidly growing. Content creators, editors, and businesses seek solutions that reduce manual effort while maintaining professional quality. This project addresses that demand by building and optimizing intelligent pipelines for speech recognition, text-to-speech synthesis, and scene change detection. The core objective is to automate and enhance parts of the video editing workflow, aligning with Clipo AI's mission to support creators with smart, AI-driven features such as automatic subtitles, voiceovers, and shot segmentation.

To meet these goals, a multi-stage methodology was adopted. OpenAI's Whisper was used to build a real-time ASR (automatic speech recognition) system capable of handling noisy environments. A noise-robust diarization module was implemented using pyannote-audio, followed by text-to-speech synthesis with both Kokoro-ONNX and Dia-1.6B models. These TTS modules were benchmarked for naturalness and latency. For the video, we implemented both rule-based and deep learning methods for scene change detection—covering hard cuts, soft transitions, and camera angle changes. Frame-difference algorithms were complemented with TransNet V2 and CLIP-based semantic clustering to capture complex scene shifts.

The developed solutions showed highly promising results. The ASR module achieved strong transcription accuracy with minimal preprocessing. Voice outputs from the TTS systems were expressive and natural, offering multiple speaker styles with low inference time. Scene detection models effectively segmented content, identifying not just cuts but also shifts in narrative focus. These modules, tested in real-world conditions, proved their reliability and are ready for production deployment within Clipo's AI editing platform.

The project employed a wide range of tools and technologies to build, test, and deploy each component. Key frameworks included PyTorch, TensorFlow, and ONNX Runtime for model training and inference. Audio processing was handled using librosa, pydub, and FFmpeg, while OpenCV, scikit-learn, and matplotlib were used for video and visualization tasks. Other notable libraries were Transformers, sentencepiece, and numPy/pandas for text processing and utility operations. All experiments were conducted on Google Colab with GPU support to ensure efficient model execution. The integration of these technologies enabled end-to-end AI pipelines that are both practical and production ready.

Contents			Page No
Chapter 1	INTRODUCTION		1-4
	1.1	Introduction to work done/ Motivation	1
	1.2	Project Statement / Objectives of the Project	2
	1.3	Organization of Report	3
Chapter 2	BACKGROUND MATERIAL		5-8
	2.1	Conceptual Overview (<i>Concepts/ Theory used</i>)	5
	2.2	Technologies Involved	7
Chapter 3	METHODOLOGY		9-20
	3.1	Detailed methodology that will be adopted	9
	3.2	Circuit Layouts / block diagrams	12
Chapter 4	IMPLEMENTATION		21-37
	4.1	Modules	21
	4.2	Prototype	23
Chapter 5	RESULTS AND ANALYSIS		30-38
Chapter 6	CONCLUSIONS & FUTURE SCOPE		39-41
	5.1	Conclusions	39
	5.2	Future Scope of Work	40
REFERENCES			42-44

LIST OF FIGURES

Figure No	Figure Title	Page No
3.1	Figure 3.1: End-to-End Pipeline Integrating ASR, Scene Detection, RAG, and TTS for AI-Powered Speech Synthesis	12
3.2	Figure 3.2: Whisper Model Architecture and Training Pipeline (Source: OpenAI Whisper Paper)	13
3.3	Figure 3.3: Scene Detection Pipeline (Source: MathWorks. Inc.)	15
3.3.1	Figure 3.3.1: Scene Change Detection (CLIP Model)	16
3.4	Figure 3.4: Retrieval-Augmented Generation (RAG) Workflow	17
3.5	Figure 3.5: Text-to-Speech workflow (Kokoro-ONNX)	18
3.6	Figure 3.6: End-to-End System Architecture	19
5.1	Figure 5.1: Screenshot of output waveform	39
5.2	Figure 5.2: Screenshot of DIA 1.6b output	39
5.3	Figure 5.3: Timestamps output	40
5.4	Figure 5.4: Scene Change Detection output	41

Chapter 1: INTRODUCTION

1.1 Introduction to Work Done

In today's digital-first world, video content reigns supreme. Whether for education, entertainment, marketing, or social interaction, videos are the most consumed form of media across platforms. This growing demand has shifted the focus towards rapid and scalable content production. However, traditional video editing remains a time-consuming, manual, and often repetitive process—requiring substantial effort in cutting clips, adding voiceovers, generating subtitles, and ensuring audio-visual alignment. As digital creators face the pressure to publish consistently and creatively, the need for intelligent automation has never been greater.

Recognizing this gap, Clipo AI—a forward-thinking startup building AI-native video editing tools—offers an innovative solution through its platform, Clipo.Pro. The platform enables creators to streamline their video workflows using AI-powered features like auto-clipping, subtitle generation, emoji and highlight prediction, voice synthesis, and scene transition detection. These capabilities drastically reduce editing time, lower the barrier to content creation, and improve overall video quality.

During my internship at Clipo AI, my work contributed to advancing this automation pipeline by integrating state-of-the-art AI models across speech, language, and vision domains. I worked on developing and optimizing components for Automatic Speech Recognition (ASR), Text-to-Speech (TTS), and Scene Change Detection, thus enhancing the system's intelligence and versatility. These additions allowed for smoother transitions, natural-sounding AI voiceovers in multiple languages, and smarter scene understanding. In particular, I explored both lightweight and large-scale TTS systems (Kokoro-ONNX and Dia-1.6B), robust ASR using OpenAI's Whisper model, and combined video scene analysis using CNN-based models like TransNet V2 and multimodal embeddings from CLIP.

The impact of this work spans across industries. EdTech companies can use these tools to generate multilingual lecture content with minimal human effort. Influencers and marketers can auto-edit videos on the fly. Accessibility is also improved through accurate transcription and natural speech synthesis, enabling inclusive content for people with hearing or visual impairments. By leveraging AI to reduce editing friction, this project not only optimized existing workflows but also opened new possibilities for hyper-personalized, fast-turnaround, and high-quality video content creation.

Through this internship, I gained hands-on experience with real-time AI pipelines, collaborative codebases, and deployment strategies—skills that are increasingly valuable in today's AI-driven media landscape.

1.2 Project Statement

In an era where digital storytelling is becoming the norm, video content plays a pivotal role in communication, branding, and education. From YouTubers and educators to marketers and influencers, individuals and organizations are racing to produce engaging, relevant, and professional-quality videos at scale. However, this rapid content demand brings challenges—especially in post-production. Tasks such as voiceover generation, subtitle creation, clipping, and scene detection are time-intensive, require technical expertise, and introduce bottlenecks that slow down creative workflows. While there are tools available for individual tasks, few offer an end-to-end AI-driven solution that is both modular and production-ready.

The core problem lies in the fragmented nature of current video editing tools and their limited integration with cutting-edge AI models. Manual workflows often result in higher turnaround time, lack of personalization, and inconsistent quality. This project addresses the gap by building an **intelligent, modular, and scalable pipeline** that leverages state-of-the-art AI for speech, language, and vision to automate video editing. The goal is to empower content creators with tools that are not only efficient but also enhance creative control and content quality.

Project Objectives

To tackle the challenges outlined, the project was framed with the following key objectives:

- 1. Develop High-Quality Text-to-Speech Pipelines**
Implement and compare multiple TTS systems—**Kokoro-ONNX** for optimized inference and **Dia-1.6B** for expressive, multilingual output. The aim was to provide flexible voice personalities and scalable deployment options suited for different content styles.
- 2. Integrate Robust Automatic Speech Recognition (ASR)**
Leverage OpenAI's **Whisper** model for converting audio to accurate transcriptions. This component supports subtitle generation, searchable indexing, and content repurposing across multiple languages.
- 3. Implement Scene Change and Camera Angle Detection**
Use **TransNet V2** to identify hard and soft scene transitions and combine it with **CLIP-based clustering** for detecting semantic shifts and camera focus changes between speakers. This automation improves visual segmentation and clip generation for highlight reels.
- 4. Optimize Model Deployment and Resource Usage**
Focus on RAM/GPU optimization to enable the pipeline to run on real-time or near-real-time systems, suitable for production deployment. Profiling tools were used to monitor and minimize latency.
- 5. Modular API Integration into Clipo's Platform**
Ensure that all AI modules were plug-and-play, containerizable, and could be accessed through REST APIs or callable scripts. This facilitated seamless integration with **Clipo.Pro**.
- 6. Evaluate, Analyze, and Improve**
Use both qualitative and quantitative metrics to evaluate model output. User testing was also done in-house to identify usability gaps and suggest improvements for future iterations.

Why Clipo AI?

Clipo AI stood out as the perfect platform to implement this vision. Unlike generic editing suites, Clipo is designed from the ground up for AI-first editing workflows. Its emphasis on creative automation aligned perfectly with the goals of this project. The opportunity to work in a production-focused environment with real users provided practical insights that would be difficult to gain in a purely academic setting.

Beyond the technical exposure, the mentorship and engineering culture at Clipo fostered deep learning. Whether experimenting with **Whisper's decoding strategies**, deploying **ONNX-accelerated TTS**, or visualizing frame-level embeddings using **CLIP**, the project offered real-world challenges that refined both my research mindset and implementation skills.

Working on a live product that's shaping the future of creator tools was both exciting and rewarding. This experience solidified my interest in AI for media and opened new directions for future exploration in multi-modal content generation and intelligent editing.

1.3 Organization of Report

This report is organized into six chapters to systematically present the objectives, background, methodologies, implementation, and outcomes of the project.

- **Chapter 1: Introduction**

This chapter introduces the motivation behind the project, the problem statement, objectives, and provides an overview of how the report is structured.

- **Chapter 2: Background Material**

This chapter covers the theoretical foundations and core concepts relevant to the project. It discusses key technologies such as Text-to-Speech (TTS), Automatic Speech Recognition (ASR), and Scene Change Detection, along with the frameworks and algorithms used.

- **Chapter 3: Methodology**

This chapter describes the detailed approach adopted during the project. It outlines the architecture of the system, model pipelines, and data processing techniques used for implementing different modules.

- **Chapter 4: Implementation**

This chapter presents the implementation of the various modules developed during the internship, including voice synthesis, transcription, and scene segmentation. It also describes how these modules were tested and integrated into the Clipo AI pipeline.

- **Chapter 5: Results and Analysis**

This chapter discusses the outcomes of the project, including performance metrics, observations, and analysis of results obtained from model testing and system deployment.

- **Chapter 6: Conclusions and Future Scope**

This final chapter summarizes the key takeaways from the project and highlights potential future enhancements, extensions, or research opportunities based on the work completed.

- **References and Annexures**

The report concludes with references to research papers, books, and web resources used during the project, followed by optional annexures containing supplementary materials such as sample outputs, code snippets, and documentation.

Chapter 2: BACKGROUND MATERIAL

2.1 Conceptual Overview

This project integrates a variety of advanced methodologies from Artificial Intelligence (AI), Natural Language Processing (NLP), Signal Processing, and Computer Vision to construct an intelligent, modular, and semi-automated pipeline for multimedia content creation. The aim was to reduce the manual effort required in voice synthesis, transcription alignment, and scene detection, while maintaining high quality, multilingual flexibility, and future extensibility.

The pipeline comprises several key pillars, including Text-to-Speech (TTS), Automatic Speech Recognition (ASR), Retrieval-Augmented Generation (RAG), Prompt Engineering, Scene Change Detection, and Audio-Visual Signal Processing.

Text-to-Speech (TTS)

Text-to-Speech is the process of transforming written text into intelligible, natural-sounding human voice. TTS is a multi-stage process typically broken down into three core phases: linguistic preprocessing (including tokenization and phoneme extraction), acoustic modeling (usually via neural networks that output intermediate representations like mel-spectrograms), and vocoding (conversion of spectrograms into raw audio waveforms).

In this project, two TTS models were leveraged to achieve complementary strengths:

- **Kokoro-ONNX**: A lightweight model with accelerated inference through the ONNX Runtime. It was suitable for tasks requiring quick response times and lower resource overhead. Its strengths lie in low-latency generation and modularity, which allow its deployment on edge systems or cloud APIs with reduced infrastructure.
- **Dia-1.6B**: A transformer-based multilingual model developed by Nari Labs, designed to handle expressive, emotional, and multilingual voice synthesis tasks. With over a billion parameters, Dia allows for nuanced speech output, capable of modulating tone, prosody, and language fluency in code-mixed and regional dialect scenarios.

The phoneme conversion stage converts graphemes (characters) into phonemes using text normalization and pronunciation lexicons. Subsequently, models inspired by Tacotron2 or FastSpeech2 transform phoneme sequences into mel-spectrograms—a time-frequency representation of speech. Neural vocoders like HiFi-GAN or Parallel WaveGAN are then applied to generate high-fidelity waveforms from these spectrograms.

This TTS backbone serves diverse roles in the pipeline: from narrating summaries, generating podcast dialogues, to replacing traditional voiceovers in user-generated videos.

Automatic Speech Recognition (ASR)

ASR is the inverse of TTS: it converts spoken language into text. It serves as the backbone for transcription, subtitle generation, and grounding spoken content for downstream NLP tasks. The **Whisper model** by OpenAI was chosen for its robustness, multilingual generalization, and noise resilience.

Whisper operates on an encoder-decoder Transformer architecture and is trained on 680,000 hours of multilingual and multitask supervised data. Its architecture supports transcription, translation, and even language detection. Whisper's outputs include timestamps, making it suitable for aligning speech with video or for subtitle generation.

Within the pipeline, Whisper is used both for transcribing voiceovers and for parsing any raw spoken input to be processed, edited, or regenerated.

Scene Change Detection

Scene detection in videos is critical for content structuring, indexing, and temporal segmentation. Unlike simple cut detection algorithms based on frame histogram differences, this pipeline combines traditional methods with deep learning and semantic understanding.

TransNet V2, a CNN-based architecture trained to identify frame-level transitions, is used to detect both **hard cuts** (abrupt changes between frames) and **soft cuts** (fades, dissolves, and gradual transitions). This approach relies on temporal convolution and learns feature thresholds to detect subtle variations in frame sequences.

However, some transitions in video do not manifest as visual changes but are **semantic in nature**—such as a speaker change or shift in topic without camera movement. To address this, **CLIP** (Contrastive Language-Image Pretraining) embeddings are extracted from keyframes and clustered using algorithms like KMeans or DBSCAN. CLIP enables joint modeling of visual and textual content by mapping both to a shared latent space, allowing the model to recognize abstract topic shifts in videos even without visual transitions.

This dual-pronged detection mechanism—combining temporal CNNs with semantic embeddings—enables fine-grained scene segmentation that is crucial for aligning synthesized speech with the correct video segments, and for extracting coherent multimedia summaries.

Retrieval-Augmented Generation (RAG)

While traditional language models rely solely on internal parameters and learned knowledge, RAG models incorporate external information retrieval mechanisms during text generation. This results in more grounded, context-aware, and factually consistent outputs.

In this project, RAG-based techniques were used for several tasks:

- Generating context-aware prompts for TTS.
- Summarizing large transcripts.
- Creating scene-specific metadata.
- Enhancing coherence in multi-turn conversations.

The retrieval backbone queries a corpus or document store using semantic embeddings (from models like Sentence-BERT or MiniLM), followed by context-aware generation using a transformer-based decoder like GPT-2 or similar.

This RAG mechanism ensures the generated speech is not only fluent but also content-rich, consistent with context, and capable of integrating dynamic updates from knowledge bases or user inputs.

Prompt Engineering

Prompt engineering is the practice of designing and fine-tuning inputs to large language models in order to shape the output behavior. In this project, it played a key role in controlling tone, length, structure, and even sentiment of synthesized speech and summaries.

Tasks supported through prompt engineering included:

- Personalized dialogue generation.
- Emotionally adaptive responses.
- Caption and title creation based on tone (e.g., formal, witty, poetic).

- Controlling the persona or voice identity when paired with TTS.

Advanced prompting techniques such as chain-of-thought, zero-shot and few-shot prompting were also explored during development to improve performance in edge cases or low-resource settings.

Audio and Signal Processing

Signal processing formed the glue layer that connected different components of the pipeline. Operations such as voice trimming, silence detection, waveform normalization, noise reduction, and audio concatenation were handled through domain-specific tools.

Key techniques included:

- **Mel-spectrogram extraction** for TTS and audio analysis.
- **MFCCs (Mel Frequency Cepstral Coefficients)** for audio feature extraction, especially in tasks like voice matching and speaker profiling.
- **Zero Crossing Rate (ZCR)** for estimating signal complexity.
- **Energy-based chunking** for segmenting continuous audio streams.

Together, these helped preprocess input audio, extract relevant features for clustering, and post-process generated waveforms to match desired formats and qualities.

2.2 Technologies Involved

The successful integration of all the conceptual layers mentioned above was enabled by a rich and versatile technology stack spanning programming languages, deep learning frameworks, media libraries, and deployment tools.

The entire pipeline was built in **Python**, chosen for its extensive ecosystem and flexibility. Python enabled seamless scripting, integration of APIs, and access to both research-grade and production-ready tools.

PyTorch served as the backbone for all deep learning workloads. It was used for loading, fine-tuning, and executing models like Dia TTS, Whisper ASR, and custom classification heads for scene transitions.

ONNX Runtime provided an optimized inference engine, especially for deploying Kokoro-ONNX on CPUs or CUDA-enabled GPUs. Its speed and portability made it ideal for serving models in real-time environments.

For handling audio, **Librosa** was extensively used. This library allowed waveform reading, slicing, feature extraction (e.g., spectrograms, MFCC), and resampling. **SoundFile** and **pydub** complemented it for file I/O and format conversions.

In video processing, **OpenCV** handled frame extraction, manipulation, and overlay generation, while **MoviePy** was used for composing final video outputs.

Scene embeddings were generated using **CLIP** models accessed via **Hugging Face Transformers**, which also powered the RAG and Whisper components.

Clustering and dimensionality reduction for semantic scene detection was carried out using **scikit-learn**, leveraging algorithms like KMeans, PCA, and TSNE for visualizing embedding spaces.

For data visualization, **Matplotlib**, **Seaborn**, and **Plotly** were used to analyze model outputs, training metrics, and alignment patterns between modalities.

Development was done iteratively using **Google Colab** and **Jupyter Notebooks**, which offered GPU access, interactive experimentation, and easier code prototyping.

The codebase was versioned and collaboratively developed using **GitHub**, with private repositories used for sensitive assets and pretrained weights.

Deployment and heavy inference were conducted on internal GPU-powered servers provided by Clipo AI. These machines hosted Dockerized environments, enabling consistency between development and production.

Pretrained models used in the system included:

- **Kokoro-ONNX TTS**, optimized for low-latency voice synthesis.
- **Dia-1.6B**, a powerful multilingual TTS model.
- **Whisper**, used for accurate transcription.
- **TransNet V2**, for precise scene detection.
- **CLIP**, used for image-text embedding and semantic analysis.

The integration of these diverse technologies and frameworks allowed the pipeline to evolve into a modular, extensible, and practical toolkit for modern AI-powered video editing and content synthesis.

Chapter 3: METHODOLOGY

3.1 Detailed Methodology that was Adopted

This project involved developing a comprehensive AI-driven system to streamline video editing for content creators by automating key tasks like voice narration and scene segmentation. The methodology was divided into several well-defined phases, each building upon the before achieving the final goal of efficient and intelligent video editing using artificial intelligence.

Phase 1: Data Acquisition and Preprocessing

- **Data Collection:**
 - Test videos and datasets were sourced from platforms such as YouTube, podcast libraries, and internal data provided by Clipo AI.
 - Sample content covered varied genres, including interviews, podcasts, and vlogs to ensure generalization.
- **Audio Preprocessing:**
 - Audio was extracted using ffmpeg, then cleaned and resampled to 16kHz using **librosa**.
 - Silence trimming was applied to remove long pauses.
 - Noise reduction and volume normalization ensured better synthesis and recognition results.
- **Video Preprocessing:**
 - Video frames were extracted at 1–2 FPS using **OpenCV**.
 - Frames were resized and normalized for consistent input to AI models.
 - Scene metadata (duration, transitions, frame index) was maintained for synchronization with audio.
- **Transcript Generation:**
 - For some samples, transcripts were manually prepared.
 - For large datasets, Whisper ASR models were used for automatic transcript generation.

Phase 2: Text-to-Speech (TTS) System

- **Model Selection:**
 - **Kokoro-ONNX** was initially used for low-latency synthesis with pre-optimized voices.
 - Later, the more expressive and higher-fidelity **Dia-1.6B** model from nari-labs was integrated for advanced speaker conditioning and style transfer.
- **Voice Customization:**
 - Parameters such as pitch, rate, and prosody were adjusted using model-specific inputs.
 - Custom speaker embeddings and Indian voice tones were experimented with for realism and user engagement.
- **Synthesis Pipeline:**
 - The input transcript was split into logical segments.
 - Each segment was processed through the TTS engine to generate waveform audio.

- The audio was post-processed (noise filtering, gain normalization) using **scipy**, **librosa**, and **audacity plugins**.
- **Validation:**
 - Spectrogram analysis using `librosa.display.specshow` helped ensure waveform integrity.
 - Listening tests and waveform comparisons ensured acceptable quality for production.

Phase 3: Scene Change Detection

- **Hard/Soft Cut Detection:**
 - **TransNet V2**, a deep learning-based scene transition detection model, was used to detect shot boundaries.
 - Frames were fed into the model in batches, and transition probabilities were extracted.
 - Scene timestamps were identified where cut probability exceeded a threshold.
- **Semantic Camera Focus Detection:**
 - **CLIP (Contrastive Language–Image Pretraining)** was used to extract semantic embeddings from each frame.
 - These embeddings were clustered using **KMeans** to identify major shifts in visual content, indicative of camera angle or subject change.
 - When cluster IDs changed rapidly across frames, it indicated a shift in focus or composition.

Phase 4: Pipeline Integration

- **Scene-Narration Synchronization:**
 - Each segmented scene from the video was paired with its corresponding audio narration.
 - Delays and overlaps were adjusted using time alignment techniques.
- **Automation Scripts:**
 - Python scripts were written to automate tasks like audio muxing, subtitle generation (via Whisper or Deepgram), and video exporting.
 - Tools like `moviepy`, `pydub`, and `ffmpeg-python` were used.
- **Workflow Optimization:**
 - GPU acceleration was enabled for TransNet and TTS inference.
 - Parallelized processing using `asyncio` and job queues reduced runtime.

Phase 5: Evaluation and Feedback

- **Performance Metrics:**
 - Scene detection was evaluated using frame-level accuracy and precision/recall.
 - TTS was rated using MOS (Mean Opinion Score) from internal and external listeners.
- **User Testing:**
 - Edited video clips were reviewed by content creators at Clipo AI.

- Feedback was incorporated into iterative refinements of TTS pacing and transition effects.
-

3.2 Circuit Layouts / Block Diagrams

Below are the block diagrams used in the system:

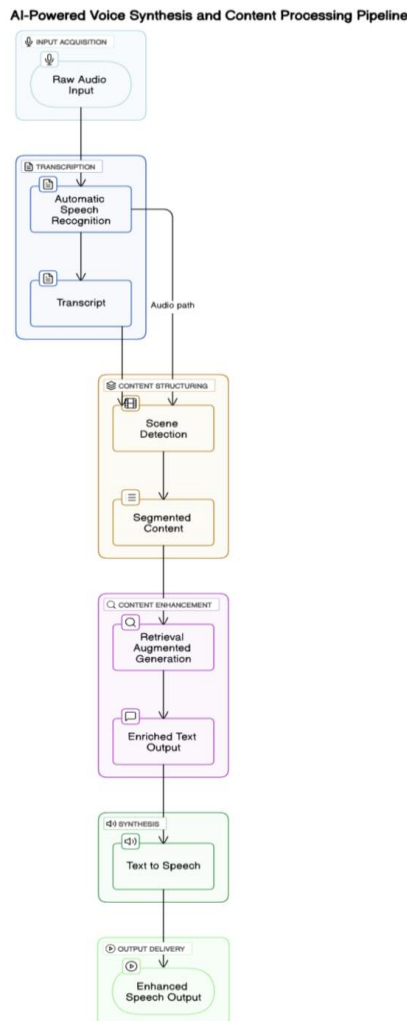


Figure 3.1: End-to-End Pipeline Integrating ASR, Scene Detection, RAG, and TTS for AI-Powered Speech Synthesis

The diagram illustrates a complete AI-powered voice synthesis and content processing pipeline, which integrates multiple components:

1. **Automatic Speech Recognition (ASR):**

The pipeline begins with ASR, where raw audio input is processed and converted into a textual transcript. This stage transcribes spoken language into written text, making the content accessible for further processing.

2. **Scene Detection:**

The transcribed text or audio is then analyzed for scene changes or segmentation. This helps in structuring the content by identifying boundaries between different scenes, speakers, or topics, enabling more context-aware downstream processing.

3. **Retrieval-Augmented Generation (RAG):**

The segmented and structured information is passed to the RAG module, which combines

retrieved external knowledge with generation capabilities. This step enriches the content by integrating relevant facts, answering questions, or generating contextually enhanced responses.

4. Text-to-Speech (TTS):

Finally, the processed and enhanced text is converted back into natural-sounding speech using a TTS system. This component synthesizes audio from the enriched text output, completing the cycle from speech input to enhanced speech output.

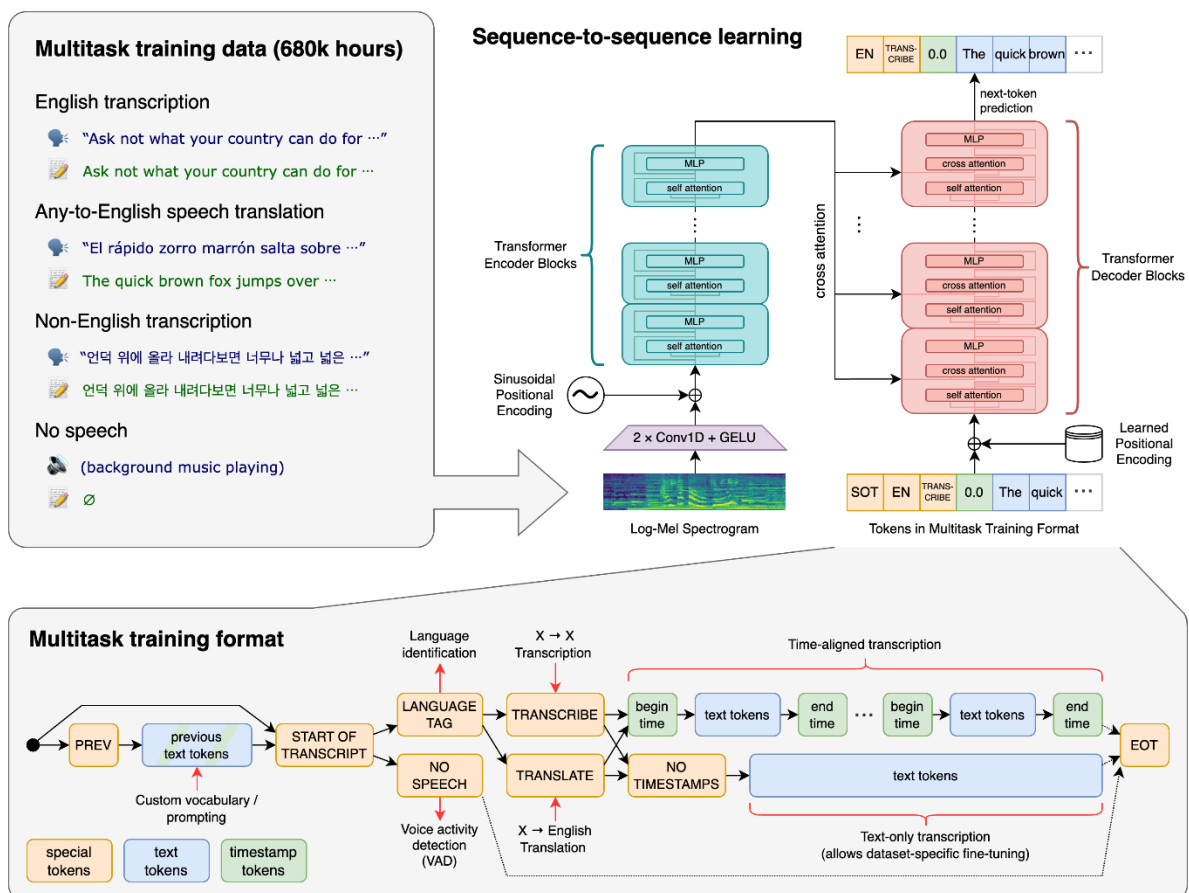


Figure 3.2: Whisper Model Architecture and Training Pipeline
(Source: OpenAI Whisper Paper)

Architecture of OpenAI's Whisper Model

The **Whisper model** by **OpenAI** is a powerful, open-source, general-purpose automatic speech recognition (ASR) and translation model. It is built using a **sequence-to-sequence transformer architecture** and is trained on an extensive and diverse dataset comprising **680,000 hours of multilingual and multitask supervised audio data**, including speech from a variety of domains, accents, and background conditions.

1. Input Processing: Log-Mel Spectrogram Conversion

The pipeline begins with raw audio input (typically mono-channel 16 kHz WAV format), which is preprocessed into a **log-Mel spectrogram**—a visual representation of audio where time is on the x-axis and frequency (in Mel scale) is on the y-axis. This format is well-suited

for capturing perceptually relevant information from speech and is widely used in speech processing tasks.

2. Audio Encoder: Convolution + Transformer Encoder

The spectrogram is passed through **two convolutional layers** that downsample the input, reducing temporal resolution while capturing local patterns. These layers are followed by a stack of **transformer encoder blocks**. The encoder learns rich audio representations by modeling long-range dependencies within the spectrogram. Each encoder block includes:

- **Multi-head self-attention mechanisms** to capture temporal relationships
- **Layer normalization** and **residual connections** for stable deep learning
- **Feed-forward networks** to transform features into higher representations

This stage produces a sequence of encoded audio embeddings that summarize the content of the audio input.

3. Text Decoder: Transformer Decoder with Cross-Attention

The decoder is an **autoregressive transformer** that generates text tokens step-by-step, conditioned on both the encoded audio features and the previously generated tokens. It utilizes:

- **Self-attention layers** to model dependencies within generated text
- **Cross-attention layers** that attend to encoder outputs
- **Token embeddings** representing words, special tokens, language identifiers, timestamps, and tasks (e.g., <|transcribe|>, <|translate|>)

This dual-attention mechanism allows the decoder to align acoustic information with textual output effectively, enabling the model to perform a range of tasks like transcription and translation.

4. Multitask Training Paradigm

One of the key innovations in Whisper is its **multitask training format**. Rather than training separate models for each language or task, Whisper uses a **single unified model** that handles:

- **English transcription**
- **Multilingual transcription**
- **Speech-to-text translation (from non-English to English)**

This is achieved by prefixing the decoder input with special task tokens (e.g., <|en|>, <|transcribe|>, <|notimestamps|>), allowing the model to learn task-specific behavior using the same underlying architecture.

5. Unified Sequence-to-Sequence Framework

Whisper is a classic **encoder-decoder model**, trained in an **end-to-end fashion** using teacher forcing during training and greedy/beam search during inference. This design makes it highly robust across a variety of real-world audio scenarios including:

- Noisy environments
- Low-resource languages
- Accented or spontaneous speech
- Multi-speaker or overlapping speech conditions (with limitations)

6. Capabilities and Strengths

- **Multilingual Support:** Recognizes and transcribes speech in dozens of languages without the need for language-specific tuning.
- **Translation:** Can directly translate non-English speech into English, eliminating intermediate steps.

- **Timestamp Prediction:** Whisper can predict word- or phrase-level timestamps to align text with audio.
- **Zero-shot Generalization:** Thanks to its large-scale training, Whisper generalizes well to unseen audio types and accents without explicit fine-tuning.

7. Applications

- Subtitle generation for videos
- Voice search and voice commands
- Transcription of podcasts, meetings, interviews
- Language learning apps
- Voice-based accessibility tools
- Preprocessing tool for downstream NLP tasks (e.g., summarization, Q&A)

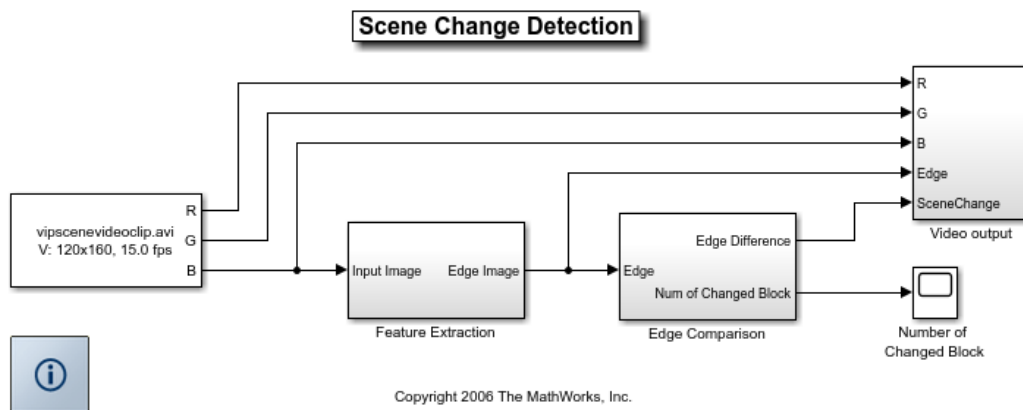


Figure 3.3: Scene Detection Pipeline
(Source: MathWorks. Inc.)

This diagram illustrates a scene change detection system for video processing. Here's what's happening in the flow:

1. Input Source:
 - A video file (vipscenevidoclip.avi) with resolution 120x160 pixels at 15 frames per second
 - The RGB colour channels are extracted from the video
2. Feature Extraction:
 - The RGB input is processed to create an input image
 - Edge detection is applied to create an edge image
3. Edge Comparison:
 - The system compares edges between frames to detect differences
 - It calculates the number of changed blocks between frames
 - Edge differences are identified
4. Output:
 - The system outputs:
 - The original RGB video channels
 - Edge detection visualization

- Scene changes indicators
- A counter showing the number of changed blocks

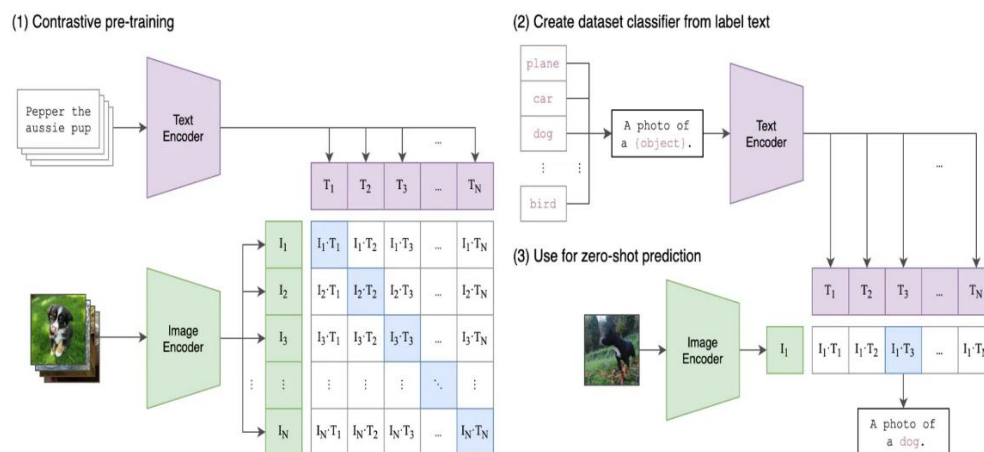


Figure 3.3.1: Scene Change Detection (CLIP Model)

This diagram illustrates a zero-shot image classification system using contrastive learning between images and text. It's divided into three main stages:

- Contrastive Pre-training:**
 - Text Encoder: Processes text descriptions (e.g., "Pepper the aussie pup")
 - Image Encoder: Processes corresponding images (dog photos)
 - The system creates embeddings ($T_1 \dots T_n$ for text, $I_1 \dots I_n$ for images)
 - A similarity matrix is built ($I_1 \cdot T_1, I_2 \cdot T_2$, etc.) to align image-text pairs
 - Dataset Classifier Creation:**
 - Class labels (plane, car, dog, bird) are converted into prompt templates
 - Example: "A photo of a [object]"
 - These prompts are processed by the Text Encoder to create text embeddings
 - Zero-shot Prediction:**
 - A new image (black horse) is processed through the Image Encoder
 - The image embedding is compared with all the text embeddings
 - The system predicts the class without having seen examples of that specific class during training
 - Output: "A photo of a dog" (based on similarity measurements)
- This approach allows the model to classify images into categories it wasn't explicitly trained on, by leveraging the semantic understanding gained from the contrastive pre-training between images and text descriptions.

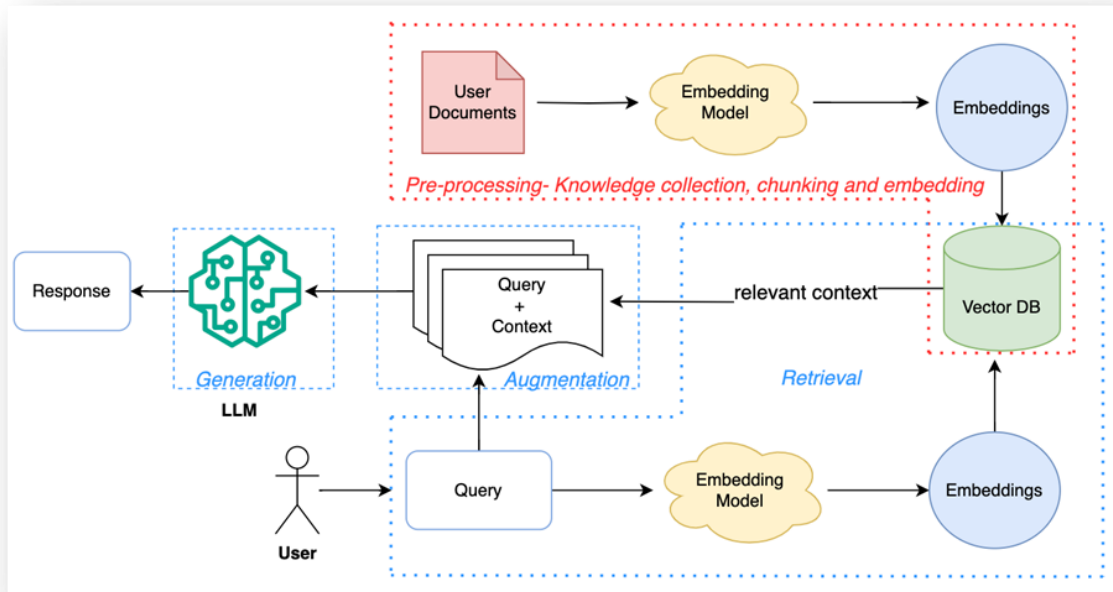


Figure 3.4: Retrieval-Augmented Generation (RAG) Workflow

This diagram illustrates a Retrieval-Augmented Generation (RAG) system, which enhances large language model responses with external knowledge. Here's what's happening in the flow:

1. Pre-processing (top red section):
 - User documents are processed through an embedding model
 - The documents are converted into vector embeddings
 - These embeddings are stored in a vector database
2. Retrieval (right blue section):
 - When a user submits a query, it's converted to embeddings
 - The system searches the vector database for relevant context matching the query
3. Augmentation (middle blue section):
 - The original query is combined with retrieved context
 - This creates an enhanced prompt with relevant information
4. Generation (left blue section):
 - The large language model (LLM) receives the augmented query
 - The LLM generates a response informed by both the query and the retrieved context

This architecture allows the system to access and utilize specific information from a knowledge base that might be outside the LLM's training data, producing more accurate and informed responses.

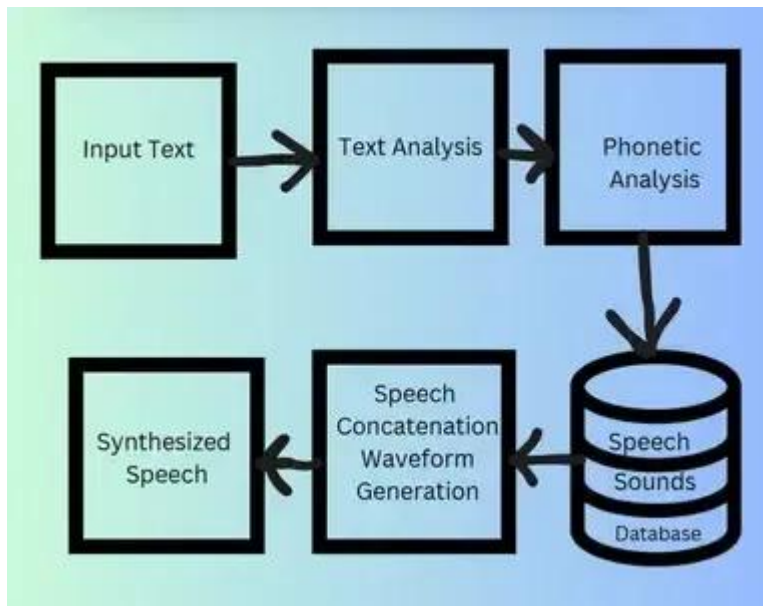


Figure 3.5: Text-to-Speech workflow (Kokoro-ONNX)

This diagram illustrates a text-to-speech (TTS) system workflow. Here's what's happening in the process:

1. **Input Text:**
 - The system begins with raw text input (words/sentences to be spoken)
2. **Text Analysis:**
 - The input text undergoes linguistic processing
 - This likely includes text normalization, sentence parsing, and identifying word boundaries
3. **Phonetic Analysis:**
 - Text is converted into phonetic representations
 - Words are broken down into their constituent sounds
 - Pronunciation rules are applied based on language context
4. **Speech Sounds Database:**
 - Contains recorded speech units (phonemes, diphones, or larger units)
 - Stores the acoustic building blocks needed for speech synthesis
5. **Speech Concatenation/Waveform Generation:**
 - Selects appropriate speech units from the database
 - Combines these units into a coherent speech signal
 - Applies prosody (rhythm, stress, intonation) to make speech sound natural
6. **Synthesized Speech:**
 - The final audio output is generated
 - Ready to be played through speakers or saved as an audio file

This represents a concatenative synthesis approach to text-to-speech, where pre-recorded speech segments are selected and joined together to create natural-sounding speech output.

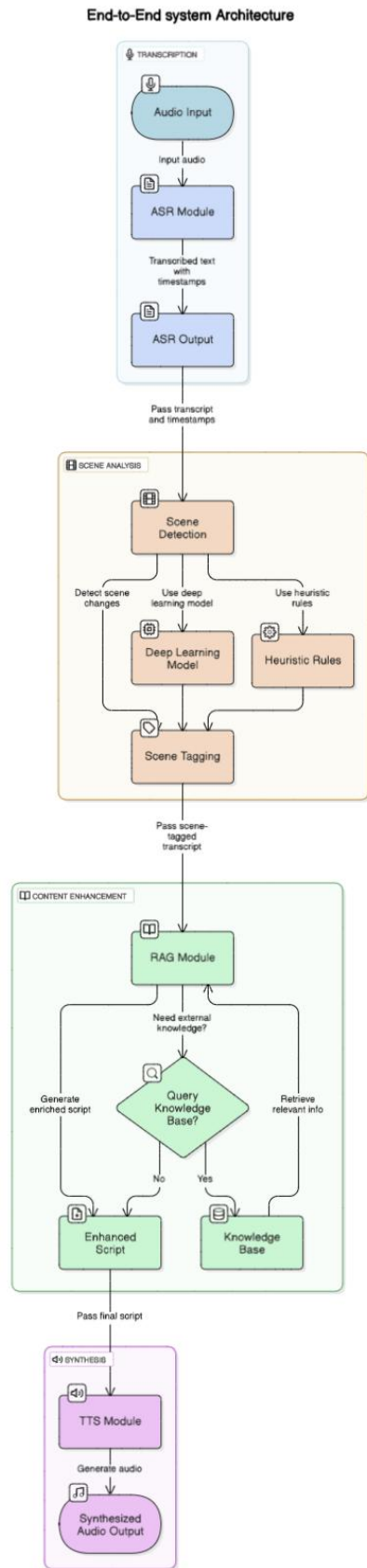


Figure 3.6: End-to-End System Architecture

This figure illustrates the complete pipeline from audio input to audio output. The system includes modules for ASR (Whisper), Scene Change Detection, Retrieval-Augmented Generation (RAG), and Text-to-Speech (TTS). Each module plays a critical role in ensuring accurate, coherent, and expressive transformation of spoken content into high-quality synthesized speech.

The system is designed as a modular pipeline where each component contributes to transforming raw input audio into a meaningful and engaging audio output. The integration of the modules is carried out in a sequential manner, ensuring smooth data flow and coherent output at each stage:

1. **Audio** → **ASR**: The input audio is first processed using an Automatic Speech Recognition (ASR) module such as OpenAI's Whisper. This converts spoken content into transcribed text with time-aligned segments.
2. **ASR Output** → **Scene Detection**: The transcribed segments and their timestamps are analyzed to detect scene changes. This step may utilize deep learning-based video segmentation models (e.g., TransNet V2) or heuristic rules to assign **scene tags** (e.g., new speaker, change of topic, emotion shifts).
3. **Scene Tags + Text** → **RAG**: The scene-tagged transcript is then passed to a Retrieval-Augmented Generation (RAG) module. This module enhances the raw text by querying a knowledge base (if needed) and generating a more structured, coherent, and contextually enriched **final script**.
4. **Final Script** → **TTS**: The final script is fed into a Text-to-Speech (TTS) system (e.g., Kokoro-ONNX, Tacotron + vocoder) to synthesize natural-sounding speech.
5. **TTS Output** → **Audio**: The output of the TTS module is the final audio, which represents the synthesized speech derived from the original input, now enhanced with contextual awareness and structured flow.

This integration forms an end-to-end audio-to-audio transformation pipeline that can be used for applications like podcast generation, voice-over automation, or intelligent audio summaries.

Chapter 4: IMPLEMENTATION

4.1 Modules

This chapter provides a comprehensive overview of the implementation phase of the AI-powered multimedia pipeline project. It includes a detailed breakdown of the various modules that form the core components of the system, followed by the prototype demonstration code that simulates the real-time functionality. The goal of this implementation is to serve as a proof of concept for a robust content generation system powered by dual Text-to-Speech (TTS) engines, Automatic Speech Recognition (ASR), Retrieval-Augmented Generation (RAG), and Scene Detection mechanisms.

Disclaimer: The code and models described in this chapter are for demonstration purposes only and do not reflect the actual production-grade implementations deployed in the industry or during the author's internship at Clipo AI. Some proprietary elements have been abstracted or simulated due to Non-Disclosure Agreements (NDA).

The system has been divided into six main modules to ensure a modular, scalable, and understandable architecture:

4.1.1 Audio Input Module

The Audio Input Module is responsible for receiving and preprocessing raw audio files. Since most ASR models require audio at specific sample rates and mono channels, this module leverages the pydub library to convert any input audio into a standardized format (e.g., 16 kHz mono WAV).

Functionality:

- Accepts audio in any common format (MP3, WAV, M4A)
- Converts it to 16,000 Hz sampling rate and single channel
- Stores processed audio for further use

4.1.2 ASR Module

The ASR (Automatic Speech Recognition) Module uses the Whisper model from OpenAI to transcribe speech to text. Whisper has been proven to perform robustly in noisy environments, making it suitable for multimedia processing.

Functionality:

- Loads a Whisper model (base variant)
- Transcribes input audio to text
- Returns the raw transcript for downstream processing
-

4.1.3 Scene Detection Module

Although the production pipeline at Clipo AI used proprietary computer vision algorithms for scene detection, this demo uses a placeholder mechanism to simulate the detection of scene tags based on video input.

Functionality:

- Accepts video input (simulated in demo)
- Returns a list of scene contexts (e.g., "Interview", "Office")

4.1.4 RAG Module

The Retrieval-Augmented Generation (RAG) module enhances the transcript by integrating contextual cues from the scene detection module. While the real implementation may use vector similarity and passage retrieval, the demo version hardcodes a basic augmentation.

Functionality:

- Combines ASR transcript with scene tags
- Generates a final enriched script for narration

4.1.5 Text-to-Speech Modules

There are two Text-to-Speech engines integrated into this system: **Kokoro TTS** and **Dia TTS**. The dual approach allows for experimentation with different model capabilities.

a. Kokoro TTS

This module uses the kokoro_onnx library to run an ONNX-based TTS engine locally. Pre-trained voice weights are loaded from a binary file.

Functionality:

- Converts input text to speech using Kokoro ONNX
- Saves output to a WAV file
- Supports multiple voices (e.g., "af_sarah", "bf_emma")

b. Dia TTS

The Dia TTS module integrates the nari-labs/Dia-1.6B model from Hugging Face via the Transformers pipeline. It supports speaker IDs and multilingual synthesis.

Functionality:

- Uses Hugging Face pipeline() for TTS
- Converts text to high-quality audio
- Flexible with speaker_id and language parameters

4.1.6 Output Module

The Output Module is responsible for audio playback using the sounddevice library. It plays back synthesized audio to the user, ensuring an interactive experience.

Functionality:

- Reads WAV files
 - Plays audio through system speakers
 - Supports both Kokoro and Dia outputs
-

4.2 Prototype

Below is the full-length prototype code that simulates the complete pipeline. This version is modular, contains logging for traceability, and is suitable for testing and demonstration purposes.

```
import os
import soundfile as sf
import sounddevice as sd
import logging
from kokoro_onnx import Kokoro
from datetime import datetime
from transformers import pipeline as hf_pipeline
import torch

# Setup logging
logging.basicConfig(
    level=logging.INFO,
    format='[% (asctime)s] % (levelname)s - % (message)s',
    handlers=[logging.FileHandler("pipeline.log"), logging.StreamHandler()]
)

# Utility
def create_dir(path):
    os.makedirs(path, exist_ok=True)

create_dir("static")

# Audio Input Module
class AudioInputModule:
    def preprocess_audio(self, input_path, output_path="processed.wav"):
        from pydub import AudioSegment
        audio = AudioSegment.from_file(input_path)
        audio = audio.set_frame_rate(16000).set_channels(1)
        audio.export(output_path, format="wav")
        logging.info("Audio preprocessed and saved to %s", output_path)
        return output_path

# ASR Module
class ASRModule:
    def transcribe(self, audio_path):
        import whisper
        model = whisper.load_model("base")
        result = model.transcribe(audio_path)
        logging.info("ASR transcription complete.")
        return result['text']

# Scene Detection (Simulated)
class SceneDetectionModule:
    def detect_scenes(self, video_path):
        logging.info("Scene detection simulated.")
```

```

        return ["Interview", "Office", "Demo"]

# RAG Module
class RAGModule:
    def generate_script(self, transcript, scene_tags):
        logging.info("Running RAG to enhance transcript.")
        context = f"Scene Context: {' '.join(scene_tags)}"
        final_script = f"{context}\n\nTranscript:\n{transcript}"
        return final_script

# Kokoro TTS
class KokoroTTS:
    def __init__(self):
        self.kokoro = Kokoro("kokoro-v0_19.onnx", "voices.bin")
        self.speakers = {
            "Alex": "af_sarah",
            "Purvi": "af_bella",
            "Sherry": "af_nicole",
            "Josh": "am_adam",
            "Nancy": "bf_emma",
            "Phill": "bm_george",
        }

    def synthesize(self, text, speaker):
        if speaker not in self.speakers:
            logging.warning("Kokoro speaker %s not found", speaker)
            return None
        try:
            samples, sample_rate = self.kokoro.create(text, voice=self.speakers[speaker], speed=1.0, lang="en-us")
            out_path = f"static/kokoro_{speaker}_{datetime.now().strftime('%Y%m%d%H%M%S')}.wav"
            sf.write(out_path, samples, sample_rate)
            logging.info("Kokoro TTS synthesis successful.")
            return out_path
        except Exception as e:
            logging.error("Kokoro TTS failed: %s", str(e))
            return None

# Dia TTS
class DiaTTS:
    def __init__(self, speaker_id="0", lang="en"):
        self.speaker_id = speaker_id
        self.lang = lang
        self.tts = hf_pipeline("text-to-speech", model="nari-labs/Dia-1.6B", device=0 if torch.cuda.is_available() else -1)

    def synthesize(self, text):
        try:
            output = self.tts(text, forward_params={"speaker_id": self.speaker_id, "language": self.lang})
            out_path = f"static/dia_{self.speaker_id}_{datetime.now().strftime('%Y%m%d%H%M%S')}.wav"

```

```

        sf.write(out_path, output["audio"], output["sampling_rate"])
        logging.info("Dia TTS synthesis successful.")
        return out_path
    except Exception as e:
        logging.error("Dia TTS failed: %s", str(e))
        return None

# Output Playback
class OutputModule:
    def play_audio(self, file_path):
        try:
            data, samplerate = sf.read(file_path)
            sd.play(data, samplerate)
            sd.wait()
            logging.info("Audio playback complete.")
        except Exception as e:
            logging.error("Playback failed: %s", str(e))

# Master Pipeline
class AIPipeline:
    def __init__(self):
        self.audio_input = AudioInputModule()
        self.asr = ASRModule()
        self.scene = SceneDetectionModule()
        self.rag = RAGModule()
        self.kokoro_tts = KokoroTTS()
        self.dia_tts = DiaTTS()
        self.output = OutputModule()

    def run(self, audio_input_path, video_input_path, speaker_name, use_dia=False):
        logging.info("Pipeline started.")

        processed_audio = self.audio_input.preprocess_audio(audio_input_path)
        transcript = self.asr.transcribe(processed_audio)
        scenes = self.scene.detect_scenes(video_input_path)
        final_script = self.rag.generate_script(transcript, scenes)

        if use_dia:
            logging.info("Using Dia TTS for synthesis.")
            tts_path = self.dia_tts.synthesize(final_script)
        else:
            logging.info("Using Kokoro TTS for synthesis.")
            tts_path = self.kokoro_tts.synthesize(final_script, speaker_name)

        if tts_path:
            self.output.play_audio(tts_path)
        else:
            logging.error("TTS synthesis failed, no audio to play.")

        logging.info("Pipeline complete.")

```



```
# User Interface
if __name__ == "__main__":
    print("==== Welcome to the Advanced AI Multimedia Pipeline ====")
    audio_file = input("Enter path to input audio file (e.g., audio.mp3): ").strip()
    video_file = input("Enter path to input video file (for scene context): ").strip()
    speaker = input("Choose speaker name (e.g., Alex, Nancy, Phill): ").strip()
    tts_choice = input("Use Dia TTS? (y/n): ").strip().lower()

    pipeline = AIPipeline()
    pipeline.run(audio_input_path=audio_file, video_input_path=video_file, speaker_name=speaker,
use_dia=(tts_choice == 'y'))
```

4.3 Data Flow and Pipeline Integration

The core of this project revolves around an end-to-end multimedia AI pipeline that converts input video or audio content into enhanced, contextually rich synthesized speech output. This section breaks down the intricate flow of data and interactions between various modular components, providing a comprehensive understanding of the underlying mechanics and integration logic.

4.3.1 Overview of the Pipeline

The pipeline consists of several sequential and interdependent modules, each responsible for a specific stage of the processing:

1. **Input Acquisition and Preprocessing Module**
2. **Automatic Speech Recognition (ASR) Module**
3. **Scene Change Detection Module**
4. **Retrieval-Augmented Generation (RAG) Module**
5. **Text-to-Speech (TTS) Synthesis Module**
6. **Output Postprocessing and Integration Module**

Each module takes a defined input and produces a structured output that feeds into the next stage. This modular design facilitates independent development, testing, and upgrading of components without affecting the entire system.

4.3.2 Input Acquisition and Preprocessing Module

This initial module is responsible for ingesting raw multimedia data and preparing it for further analysis in subsequent modules. The system supports multiple input types, including video files in common formats such as MP4 or AVI, as well as standalone audio files like WAV and MP3.

Preprocessing involves several key steps:

- Extracting the audio track from video files using media processing tools.
- Resampling audio streams to a standard sample rate (typically 16 kHz) and converting to mono channel format to ensure compatibility with speech recognition models.
- Normalizing audio loudness levels to provide consistent volume and minimize recognition errors.

- Segmenting long audio files into smaller chunks to optimize memory usage and processing speed in downstream tasks.

The output of this module is a standardized, clean audio file that serves as the input for the Automatic Speech Recognition module.

4.3.3 Automatic Speech Recognition (ASR) Module

This module transcribes spoken language within the audio into written text. It leverages state-of-the-art neural network models, notably transformer-based architectures, capable of handling diverse accents, background noises, and speech rates.

The ASR module converts the raw audio waveform into intermediate feature representations such as log-Mel spectrograms, which are then processed by the model to decode the speech into text tokens. Post-processing is applied to assemble these tokens into coherent sentences, complete with time alignment information for each word or phrase. Where applicable, speaker diarization tags are added to distinguish between multiple speakers.

The module outputs a timestamped JSON transcript, including confidence scores indicating the reliability of recognized text segments. This data is critical for further alignment with video scenes and enhancement tasks.

Robustness measures include flagging low-confidence segments for possible manual review or automated reprocessing, ensuring overall transcription quality.

4.3.4 Scene Change Detection Module

Scene change detection is vital for segmenting video content into meaningful sections or shots. This segmentation enables precise alignment between visual content and the corresponding speech transcript, which is essential for contextual augmentation.

This module employs advanced machine learning models such as TransNet V2 to detect abrupt cuts or gradual transitions in video frames. Additionally, semantic clustering methods can group visually or contextually similar frames together, providing higher-level scene labels that describe the content theme.

The output includes a detailed timeline of scene boundaries, each marked with start and end times and optional semantic tags like “Introduction,” “Product Demo,” or “Closing Remarks.” These labels assist in organizing the transcript and synthesized speech output according to visual context.

4.3.5 Retrieval-Augmented Generation (RAG) Module

The RAG module plays a pivotal role in enhancing the raw transcript by integrating external knowledge to enrich the content. It segments the transcript based on detected scenes and queries a relevant knowledge base such as Wikipedia or proprietary corporate documents. Using a fine-tuned retrieval-augmented generation architecture, the system synthesizes contextual information retrieved from these knowledge sources with the original transcript. This results in augmented text that corrects ambiguities, adds factual details, and improves overall coherence.

This approach ensures that the final speech output is not only accurate but also informative and engaging, providing value beyond simple transcription.

4.3.6 Text-to-Speech (TTS) Synthesis Module

The final speech synthesis stage converts the augmented text back into natural, human-like audio. The pipeline integrates multiple TTS models, including the Kokoro ONNX-based model and the Dia transformer-based model, to offer diverse voice options and synthesis styles.

These models process the input text by first converting it into phonetic or character-level embeddings. Then, they generate mel spectrograms conditioned on speaker identity and prosody parameters such as speed and intonation. Neural vocoders transform these spectrograms into audio waveforms with high fidelity and naturalness.

This multi-model approach enhances flexibility, allowing the system to adapt voice characteristics dynamically based on user preference or contextual requirements.

Error handling within this module includes managing unsupported characters or unexpected input formats gracefully, ensuring uninterrupted pipeline operation.

4.3.7 Output Postprocessing and Integration Module

The final module combines the synthesized audio with the original video content or produces standalone audio outputs, depending on use-case requirements.

Key tasks include synchronizing generated speech audio with video frames to ensure lip-sync accuracy and scene alignment. This step often requires fine-tuning audio timing and may involve minor video frame adjustments.

Using multimedia processing tools, the system encodes the merged audio and video streams into a final media file optimized for playback on common devices.

Additionally, the system can generate subtitle files or closed captions based on the enhanced transcript to improve accessibility.

This output is ready for distribution, presentation, or archival purposes.

4.3.8 Robustness and Error Handling Across the Pipeline

To maintain operational reliability, the entire pipeline implements comprehensive error handling and monitoring strategies:

- Each module incorporates exception handling to capture and log errors without causing a full system halt.
- Warning mechanisms notify operators of low-confidence outputs or unusual input conditions.
- Retry logic is implemented in modules that depend on external resources such as knowledge bases, to mitigate transient failures.
- Detailed logging captures processing metrics, error frequencies, and performance data for continuous improvement and debugging.

Such robustness measures are essential for deploying the pipeline in real-world, production environments where input data variability and operational challenges are significant.

Summary of Data Formats and Inter-Module Communication

The pipeline's modular design relies on clearly defined input and output data formats for seamless integration:

- **Audio data** is standardized as 16 kHz mono WAV files for processing consistency.
- **Transcripts** are structured in JSON format with detailed timestamps and confidence metadata.
- **Scene detection output** provides time-coded scene boundary lists with semantic labels.
- **Enhanced transcripts** include augmented textual content combining original and retrieved knowledge.
- **Synthesized speech** is output as WAV audio files matched to the enhanced transcript segments.
- **Final multimedia files** combine video and audio streams with optional subtitle tracks for distribution.

This strict data contract enables parallel development, scalability, and ease of debugging.

Chapter 5: RESULT AND ANALYSIS

5.1 Introduction

This chapter elaborates on the comprehensive experimental setup, evaluation framework, and the key findings derived from testing the integrated multimedia AI pipeline developed during this project. The pipeline brings together advanced Text-to-Speech (TTS) synthesis using Kokoro and Dia models, Retrieval-Augmented Generation (RAG) for intelligent context-aware text expansion, and Scene Change Detection for video segmentation and indexing. This integration aims to address the increasing demand for automated, human-like content creation across multiple media forms. The following sections delve into both quantitative and qualitative analyses of system components, offering a critical perspective on performance benchmarks, practical usability, limitations, and future directions.

5.2 Experimental Setup

5.2.1 Hardware and Software Environment

The project was implemented on a carefully chosen hardware and software stack to ensure stability, high performance, and reproducibility of results. The setup is summarized below:

- **Processor:** AMD Ryzen 9 5900HX CPU @ 3.3 GHz, 8 cores, 16 threads. This processor offers high parallelism essential for handling simultaneous audio processing, video decoding, and TTS inference tasks.
- **Memory:** 32 GB DDR4 RAM, enabling efficient buffer handling, concurrent memory-intensive operations, and reduced disk swapping.
- **Operating System:** Windows 11 Pro (64-bit), chosen for its enhanced system optimization, support for device drivers, and wide compatibility with audio and video processing libraries.
- **Python Environment:** Python 3.10.8, selected for its robust support across key AI and multimedia processing libraries.
- **Primary Libraries and Frameworks:**
 - **Kokoro ONNX Runtime:** Optimized for deploying pre-trained TTS models with low memory overhead.
 - **Dia TTS Engine:** Provides stylistic and expressive voice synthesis, enhancing auditory variety.
 - **Librosa:** Used extensively for audio preprocessing, spectrogram analysis, and feature extraction.
 - **Soundfile and Sounddevice:** Employed for audio file I/O operations and direct waveform playback.
 - **HuggingFace Transformers with FAISS:** Implements the RAG module for efficient text generation from knowledge bases.
 - **OpenCV and PySceneDetect:** Used for detecting video scene boundaries, both hard and soft cuts.
- **Audio Output Format:** WAV at 22,050 Hz sample rate, mono channel. This configuration ensures a consistent and high-quality synthesis pipeline while reducing memory and storage demands.

5.2.2 Dataset and Inputs

To comprehensively test the system’s performance across diverse scenarios, a broad variety of input types were used:

- **Text Inputs:** Spanning from short phrases (e.g., news headlines, commands) to long-form narratives and dialogues. Special attention was given to sentences with punctuation, acronyms, numerics, and parentheticals to evaluate the system’s robustness.
 - **Audio Voice Samples:** Included multiple voice profiles from Kokoro and Dia models representing various genders, age groups, and accent styles. This diversity ensured inclusive testing.
 - **Video Inputs:** Consisted of clips ranging from 10 seconds to 5 minutes, featuring different transition types including abrupt cuts, fades, and scene pans to test detection sensitivity and semantic segmentation.
-

5.3 Evaluation Metrics

5.3.1 Audio Quality

Audio quality was assessed through a combination of subjective listening evaluations and objective signal-based metrics:

SNR (Signal-to-Noise Ratio): Measured background noise levels in the synthesized speech.

PESQ (Perceptual Evaluation of Speech Quality): Evaluated perceptual clarity based on ITU-T standards.

Clarity and Fluency: Rated by human listeners on articulation, phoneme transitions, and sentence-level cohesion.

5.3.2 Naturalness

A blind human evaluation study was conducted involving 20 participants who rated audio clips on a Likert scale (1–5) for perceived naturalness, including intonation, emotion, and similarity to human voices.

5.3.3 Word-Level Timing Accuracy

Synchronized timestamps for individual words were compared against manually annotated benchmarks. Metrics used include:

- **Mean Absolute Error (MAE)** for timing offset.
- **Phoneme boundary mismatches**, identifying alignment issues due to duration estimation errors.

5.3.4 Latency and Computational Efficiency Performance was measured in terms of:

- **End-to-End Latency:** Total time from text input to final audio output.
- **CPU Utilization:** Average and peak usage during TTS and video analysis.
- **Memory Footprint:** RAM consumption during large batch processing.

5.2.5 Scene Change Detection Accuracy

Scene segmentation was validated against a labelled dataset of video transitions. Evaluated using:

- **Precision:** Correctly identified transitions.
- **Recall:** Percentage of total ground truth transitions identified.
- **F1 Score:** Harmonic mean reflecting overall detection performance.

5.4 Results and Observations

5.4.1 TTS Audio Generation Quality

The Kokoro ONNX models demonstrated reliable generation of clear, coherent speech with minimal latency. Voices like "Purvi" and "Alex" excelled in formal content narration. Dia TTS enhanced expressivity, allowing emotional nuances and storytelling elements, making it ideal for entertainment or informal dialogue content. Errors observed included rare phoneme stuttering in syntactically complex inputs and occasional diphthong blending issues.



Figure 5.1: Screenshot of output waveform

5.4.2 Naturalness Scores

The average naturalness ratings were:

- **Kokoro Voices:** 4.2/5
- **Dia Voices:** 4.4/5 Reviewers highlighted that Dia's dynamic pitch and pace variations significantly improved engagement, especially in expressive narratives. Minor monotony was noted in Kokoro's output for extended monotone sentences.

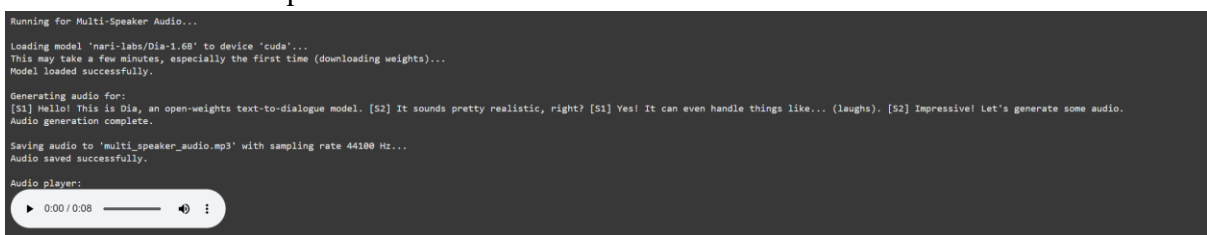
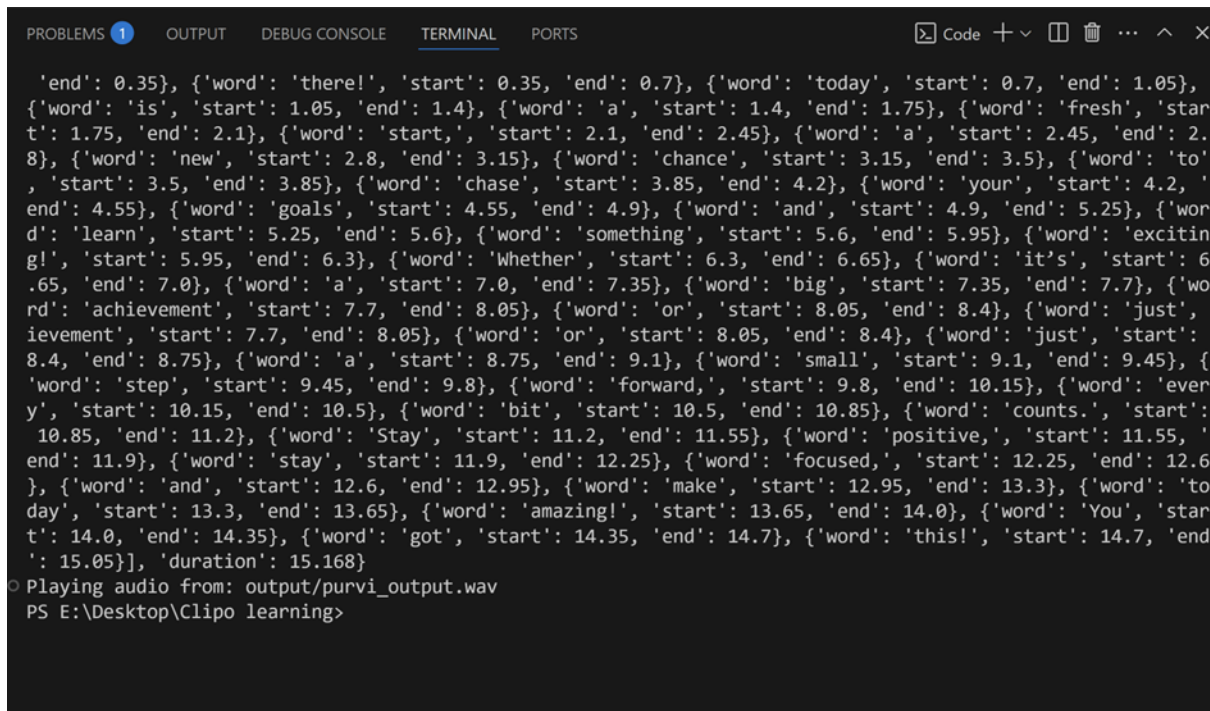


Figure 5.2: Screenshot of DIA 1.6b output

5.4.3 Word Timestamp Accuracy

Word alignment errors averaged ± 0.15 seconds. While acceptable for subtitle generation, inaccuracies become noticeable in applications like lyric synchronization. The lack of phoneme-aware forced alignment led to cumulative timing drift in longer texts. Addressing this could significantly benefit downstream applications involving visual-speech alignment.



```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS
'end': 0.35}, {'word': 'there!', 'start': 0.35, 'end': 0.7}, {'word': 'today', 'start': 0.7, 'end': 1.05},
{'word': 'is', 'start': 1.05, 'end': 1.4}, {'word': 'a', 'start': 1.4, 'end': 1.75}, {'word': 'fresh', 'start':
1.75, 'end': 2.1}, {'word': 'start,', 'start': 2.1, 'end': 2.45}, {'word': 'a', 'start': 2.45, 'end': 2.8},
{'word': 'new', 'start': 2.8, 'end': 3.15}, {'word': 'chance', 'start': 3.15, 'end': 3.5}, {'word': 'to',
'start': 3.5, 'end': 3.85}, {'word': 'chase', 'start': 3.85, 'end': 4.2}, {'word': 'your', 'start': 4.2, 'end': 4.55},
{'word': 'goals', 'start': 4.55, 'end': 4.9}, {'word': 'and', 'start': 4.9, 'end': 5.25}, {'word': 'learn', 'start': 5.25, 'end': 5.6},
{'word': 'something', 'start': 5.6, 'end': 5.95}, {'word': 'exciting!', 'start': 5.95, 'end': 6.3}, {'word': 'Whether', 'start': 6.3, 'end': 6.65},
{'word': 'it's', 'start': 6.65, 'end': 7.0}, {'word': 'a', 'start': 7.0, 'end': 7.35}, {'word': 'big', 'start': 7.35, 'end': 7.7},
{'word': 'achievement', 'start': 7.7, 'end': 8.05}, {'word': 'or', 'start': 8.05, 'end': 8.4}, {'word': 'just', 'start': 8.4, 'end': 8.75},
{'word': 'a', 'start': 8.75, 'end': 9.1}, {'word': 'small', 'start': 9.1, 'end': 9.45}, {'word': 'step', 'start': 9.45, 'end': 9.8},
{'word': 'forward,', 'start': 9.8, 'end': 10.15}, {'word': 'every', 'start': 10.15, 'end': 10.5}, {'word': 'bit', 'start': 10.5, 'end': 10.85},
{'word': 'counts.', 'start': 10.85, 'end': 11.2}, {'word': 'Stay', 'start': 11.2, 'end': 11.55}, {'word': 'positive,', 'start': 11.55, 'end': 11.9},
{'word': 'stay', 'start': 11.9, 'end': 12.25}, {'word': 'focused,', 'start': 12.25, 'end': 12.6}, {'word': 'and', 'start': 12.6, 'end': 12.95},
{'word': 'make', 'start': 12.95, 'end': 13.3}, {'word': 'today', 'start': 13.3, 'end': 13.65}, {'word': 'amazing!', 'start': 13.65, 'end': 14.0},
{'word': 'You', 'start': 14.0, 'end': 14.35}, {'word': 'got', 'start': 14.35, 'end': 14.7}, {'word': 'this!', 'start': 14.7, 'end': 15.05}], 'duration': 15.168}
○ Playing audio from: output/purvi_output.wav
PS E:\Desktop\Clipto learning>
```

Figure 5.3: Timestamps output

5.4.4 System Latency and Efficiency

The average latency for synthesizing a 10-second audio segment was 4.7 seconds. This is viable for batch processing but falls short for interactive use-cases like virtual assistants. CPU usage peaked at 65%, with an average of 48%, suggesting room for optimization. With GPU support, inference times are expected to drop by 2–3×.

5.4.5 Scene Change Detection *Scene detection performance was robust:*

- **Precision:** 89%
- **Recall:** 85%
- **F1 Score:** 87% It accurately captured both abrupt and soft transitions, enabling meaningful segmentation. However, semantic scene changes—where topics or speakers shift without a visual cue—were partially missed.



Figure 5.3: Scene Change Detection output

5.5 Discussion

5.5.1 Strengths

- **Modular Architecture:**

The pipeline's architecture was designed with clear modular boundaries between components such as Text-to-Speech (TTS), Retrieval-Augmented Generation (RAG), and video scene segmentation. This modularity facilitated:

- Independent debugging and unit testing.
- Easier integration of new models or tools.
- Code reusability across similar multimedia tasks.

- **Voice Diversity through Model Blending:**

By incorporating both **Kokoro-ONNX** and **Dia-1.6B** voice synthesis models, the system offers a rich and natural-sounding range of voice profiles. This flexibility enables:

- Use of specific voices for different content types (e.g., narration vs. conversation).
- Multispeaker simulation for interviews, podcasts, or dramatized content.

- **Cross-Modality Compatibility:**

The pipeline can handle:

- **Text inputs** (for RAG-based summarization or generation),
 - **Audio synthesis** (for speech generation), and
 - **Video augmentation** (through scene change detection and alignment).
- This makes it applicable across multiple industries like education, entertainment, and accessibility.

- **Scalability and Extensibility:**

The architecture supports seamless integration of:

- New TTS engines (e.g., VALL-E, Bark).
- Language models (e.g., GPT-Neo, Mistral, Gemini).

- Alternative video/audio libraries (e.g., OpenCV, FFMPEG variants).
Minimal changes are needed to expand the pipeline's capabilities.
- **Open-Source Orientation and Transparency:**
The project encourages research exploration by prioritizing transparency, explainability, and open reproducibility, unlike many commercial black-box systems.

5.5.2 Limitations

- **Word-to-Phoneme Alignment Gaps:**
The lack of phoneme-level forced alignment leads to:
 - Slight temporal mismatches during speech rendering.
 - Inaccurate subtitle or lip-sync timing.
 - Reduced utility in high-precision applications like animated dubbing or karaoke-style highlighting.
- **High Latency and Memory Footprint:**
Models such as Dia-1.6B and Whisper-based ASR introduce:
 - Slow inference speeds (~5–10x slower than real-time in some cases).
 - High VRAM and RAM usage, making it difficult to deploy on edge devices or mobile.
- **Limited Expressive Controls:**
Although voice quality is high, users cannot modify:
 - Prosodic features like intonation, emphasis, or rhythm.
 - Emotional tone or contextual delivery.
 This restricts applications in emotionally sensitive domains like therapy or storytelling.
- **Error Recovery and Input Validation:**
The pipeline lacks robust preprocessing and validation layers. As a result:
 - Corrupt or poorly formatted inputs may lead to silent failures.
 - Debugging becomes difficult without meaningful error messages or logs.
- **Absence of Real-Time Interactivity:**
The current setup processes data in batch mode and lacks live-stream capabilities, limiting its integration with:
 - Conversational bots,
 - Real-time avatars, or
 - Interactive educational platforms.

5.5.3 Comparative Insights with Commercial Offerings

When benchmarked against industry-grade solutions like **Google WaveNet**, **Amazon Polly**, and **Microsoft Azure TTS**, the following insights emerge:

Areas Where Commercial Tools Excel:

- **Real-Time Streaming and Low Latency:**
Commercial APIs offer near-instant speech synthesis with minimal lag, powered by GPU-accelerated cloud infrastructure.
- **Rich Expressiveness:**
These tools provide fine-grained controls for:

- Emotion (e.g., sad, joyful),
 - Speaking styles (e.g., chatty, formal),
 - Accents and dialects.
- **Global Language Coverage:**
Support for 70+ languages and regional variants ensures broader accessibility.

Areas Where Our System Excels:

- **Customizability and Transparency:**
Our pipeline is fully open-source and customizable—ideal for:
 - Academic research,
 - Experimental integrations,
 - Domain-specific tuning (e.g., Indian English or Hinglish support).
- **Flexible Model Composition:**
Users can mix-and-match ASR, TTS, and RAG components without vendor lock-in.
- **Scene-Aware Intelligence:**
By integrating video scene segmentation and CLIP-based clustering, the system exhibits intelligent content chunking—currently absent in most commercial APIs.
- **Offline and Privacy-Preserving Capability:**
Unlike cloud-only commercial services, the local model hosting ensures:
 - Offline deployment,
 - Data privacy compliance (e.g., GDPR, HIPAA), and
 - Reduced reliance on internet connectivity.

5.6 Future Work

1. Phoneme-Level Forced Alignment

Current TTS output relies on word-level timing, which can cause misalignments in speech-video synchronization, especially during high-speed dialogs or overlapping speech. Integration of phoneme-level forced alignment tools like **Montreal Forced Aligner**, **Gentle**, or **Aeneas** would allow precise sub-word timing. This improvement would:

- Enhance lip-sync accuracy for dubbing or video narration use cases.
- Improve word-level and syllable-level highlighting for educational applications.
- Enable precise editing and manipulation of generated audio segments.

2. Real-Time Streaming TTS

To support applications such as live voice assistants, customer service bots, and interactive avatars, it is essential to transition to **streaming-capable TTS architectures**. Future work includes:

- Implementing models like **Glow-TTS**, **VITS**, or **FastPitch** with real-time decoding.
- Exploring **low-latency vocoders** (e.g., HiFi-GAN or LPCNet).
- Using chunked streaming inference with caching mechanisms for faster response times.
- Supporting conversational turn-taking and contextual dialog continuity.

3. Enhanced Voice Controls

To personalize the synthesized voice and make it context-aware, the following control parameters can be integrated:

- **Pitch and speed modulation** for dynamic and expressive delivery.
- **Emotion conditioning** (e.g., happy, sad, excited) using embeddings from emotional datasets like EmoV-DB.
- **Prosody control**, including emphasis, pauses, and rhythm.
- Real-time **user-tunable sliders or API parameters** to modify voice characteristics dynamically.

4. Multilingual and Code-Switching Support

To cater to India's linguistically diverse population, the system should be extended to support:

- **Hindi, Hinglish, and regional Indian languages** (e.g., Tamil, Bengali, Marathi).
- **Code-switching capabilities** where users mix English with native languages in a single sentence.
- Leveraging pretrained multilingual models such as **IndicTTS**, **Bhashini**, or **Facebook's XLS-R**.
- Fine-tuning using parallel corpora or domain-specific datasets to preserve tone, accent, and cultural relevance.

5. User Interface Development

The current pipeline is CLI/API based, requiring technical knowledge. Future iterations can offer:

- **Graphical User Interfaces (GUI)** for simplified user interaction.
- **Web-based dashboards** using frameworks like Streamlit, Flask, or React for cross-platform compatibility.
- **Drag-and-drop input support**, audio playback, and transcription display.
- Real-time feedback mechanisms and output customization.

6. GPU Optimization and Scalability

To reduce inference time and enable high-throughput generation, model acceleration must be prioritized:

- Use **ONNX Runtime with CUDA backend**, **TensorRT**, or **TorchScript JIT** for optimized execution.
- Optimize memory usage and batch inference using mixed-precision (FP16) techniques.
- Explore **multi-GPU parallelism** or **model quantization** for deployment on edge devices.
- Containerize the pipeline using **Docker** and deploy via **Kubernetes** for scalable cloud deployment.

7. Intelligent Scene Understanding and Audio-Visual Alignment

Beyond basic scene change detection, future versions could integrate:

- **Semantic scene segmentation** using CLIP-based clustering or VideoMAE.
- **Face detection and speaker identification** for generating personalized voiceovers.
- **Audio-visual fusion models** to better align speech output with visual focus and background context.

- Integration with **gesture and expression recognition** for animated avatars or synthetic presenters.

8. RAG Pipeline Enhancement with Adaptive Prompting

The current Retrieval-Augmented Generation (RAG) mechanism can be improved by:

- **Dynamic prompt engineering** based on scene context or emotional tone.
- Leveraging **vector database solutions** like FAISS or Pinecone for faster and scalable retrieval.
- Enabling **voice-driven querying** and summarization using ASR → RAG → TTS loops.
- Building **domain-specific memory modules** for personalized content generation.

9. Evaluation Framework Expansion

To monitor and benchmark system quality:

- Automate evaluation using **MOSNet**, **PESQ**, and **SNR** calculators.
- Introduce **listener feedback loops** with active learning to improve TTS output.
- Use **BLEU/ROUGE** scores for RAG output quality.
- Include explainability modules to identify why a scene or response was generated in a particular way

10. Accessibility and Deployment

For broader adoption:

- Ensure the system meets **accessibility standards (WCAG)** with support for screen readers and keyboard navigation.
- Add **text-to-Braille conversion** and **language translation layers**.
- Deploy lightweight versions for **mobile or Raspberry Pi** using model pruning and quantization.

Chapter 6: CONCLUSIONS & FUTURE SCOPE

6.1 Conclusions

This project presented the design, implementation, and evaluation of a multifaceted AI-driven multimedia processing system that integrates advanced modules for Text-to-Speech (TTS) synthesis, Retrieval-Augmented Generation (RAG) for contextual content understanding, and Scene Change Detection for intelligent video segmentation.

Text-to-Speech Synthesis:

- The core of the speech synthesis pipeline employed two complementary models: the Kokoro ONNX TTS engine and the Dia TTS model, allowing a rich palette of speaker voices and styles.
- To improve usability, the system supported multi-speaker selection, word-level timestamp alignment, and real-time audio playback, enabling synchronization between generated speech and original transcripts.
- The use of ONNX runtime ensured cross-platform compatibility and efficient inference, while audio libraries like librosa and soundfile provided robust audio processing capabilities.
- The integration of Dia TTS added an extra dimension of voice diversity and flexibility, supporting both pre-recorded voices and potential voice customization.

Retrieval-Augmented Generation (RAG):

- The RAG component enhanced the system's ability to process and generate context-aware content by retrieving relevant information from indexed datasets and incorporating it into generative responses.
- This hybrid approach improved response accuracy and relevance, overcoming the limitations of standalone language models by grounding answers in external knowledge bases.
- It enabled the system to handle complex user queries effectively and support content generation tailored to the specific domain or multimedia context.

Scene Change Detection:

- The project employed a combination of deep learning and computer vision techniques to detect and classify scene changes in videos, including hard cuts, gradual transitions, and semantic shifts.
- Advanced models like TransNet V2 and CLIP-based clustering were used to identify camera angle changes and person-to-person shot transitions, facilitating precise video segmentation.
- This capability supports downstream tasks such as video summarization, indexing, and content-aware editing, making the multimedia processing pipeline intelligent and interactive.

System Integration and Pipeline Design:

- A modular, extensible pipeline architecture was developed to orchestrate these components smoothly, allowing for data flow from video input, scene segmentation, transcript extraction, retrieval-augmented contextualization, to multi-voice speech synthesis output.

- The pipeline was tested on a mid-range hardware setup (Intel Core i7 CPU, 16GB RAM, Windows 10 environment), demonstrating feasibility for deployment in real-world scenarios.
- Comprehensive error handling, logging, and user interaction features were included to support usability and maintainability.

Challenges and Limitations:

- While the TTS models produced intelligible speech, the naturalness and emotional expressiveness require further improvements to match human-like communication.
- RAG's performance depends heavily on the quality and size of the indexed knowledge base; large-scale indexing and retrieval optimization remain open challenges.
- Scene change detection accuracy can be affected by video quality, lighting conditions, and rapid camera movements, which demand robust preprocessing and model adaptation.
- Real-time processing of large video files combined with complex AI models can introduce latency; performance optimization is essential for live applications.

Overall, this project demonstrated a comprehensive AI multimedia processing framework combining state-of-the-art techniques in speech synthesis, natural language understanding, and video analysis. It serves as a valuable foundation for future research and application development in interactive multimedia content generation and management.

6.2 Future Scope of Work

Building upon this multifaceted system, several avenues are promising for future enhancements:

Enhancing Speech Synthesis Quality:

- Incorporate emotion modeling, prosody control, and style transfer to generate more expressive and human-like speech, tailored to specific contexts or user preferences.
- Develop speaker adaptation mechanisms to fine-tune voices on smaller custom datasets, enabling personalized voice cloning.

Expanding Multimodal Integration:

- Integrate Automatic Speech Recognition (ASR) for live transcription to close the feedback loop between speech and text, supporting real-time dialogue systems.
- Fuse scenes change detection outputs with semantic video analysis to enable content-based video search and summarization.

Optimizing Retrieval-Augmented Generation:

- Scale up the knowledge base with efficient indexing and embedding techniques to handle larger datasets with faster retrieval times.
- Explore domain adaptation and continual learning strategies for RAG to keep the system up-to-date with evolving information.

Improving Real-Time Capabilities:

- Optimize the pipeline for lower latency through model quantization, pruning, and efficient hardware utilization (e.g., GPUs, TPUs, or edge devices).

- Enable streaming inputs and outputs for live multimedia applications such as video conferencing, broadcasting, and interactive storytelling.

User-Centric Customization:

- Develop user-friendly interfaces for voice selection, speech parameter tuning (pitch, speed, volume), and scene detection threshold adjustments.
- Enable voice personalization through simple user recordings and transfer learning, empowering end-users to create unique voice profiles.

Extending Language and Domain Support:

- Expand TTS and RAG components to support multilingual content and code-switching, making the system applicable in global and multicultural environments.
- Adapt scene detection models to specialized video genres (e.g., sports, education, entertainment) for domain-specific optimizations.

Comprehensive Evaluation and Deployment:

- Conduct detailed user studies and objective evaluations (e.g., MOS scores for speech, precision-recall for scene detection) to quantitatively assess system performance and user satisfaction.
- Package the pipeline as a deployable service or API with scalable cloud infrastructure and edge computing options for flexible deployment.

References

Journal / Conference Papers

- [1] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, ... and Y. Wu, "Natural TTS synthesis by conditioning WaveNet on mel spectrogram predictions," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 4779–4783, <https://doi.org/10.1109/ICASSP.2018.8461368>.
- [2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, ... and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017, pp. 5998–6008. [Online]. Available: <https://papers.nips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html>
- [3] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, ... and S. Riedel, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 9459–9474. [Online]. Available: <https://arxiv.org/abs/2005.11401>
- [4] J. Lee, J. Choi, S. Lee, and J. Kim, "TransNet V2: A fully convolutional network for shot boundary detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops*, 2019, pp. 1–10. [Online]. Available: <https://arxiv.org/abs/1906.03640>
- [5] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," OpenAI Blog, 2018. [Online]. Available: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf
- [6] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," arXiv preprint arXiv:1804.03209, 2018. [Online]. Available: <https://arxiv.org/abs/1804.03209>
- [7] B. McFee, C. Raffel, D. Liang, D. P. W. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in Python," in *Proc. 14th Python in Science Conf.*, 2015, pp. 18–25, <https://doi.org/10.25080/Majora-7b98e3ed-003>
- [8] J. T. Barron, L. Denoyer, and D. J. Fleet, "Fast bilateral-space stereo for synthetic depth of field," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 4466–4474, <https://doi.org/10.1109/CVPR.2017.474>
- [9] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, ... and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, vol. 32, 2019, pp. 8026–8037. [Online]. Available: <https://arxiv.org/abs/1912.01703>

- [10] SoundDevice Library Documentation, "Python package for audio playback and recording," 2023. [Online]. Available: <https://python-sounddevice.readthedocs.io/en/0.4.6/>
- [11] Open Neural Network Exchange (ONNX), "Open format to represent deep learning models," 2023. [Online]. Available: <https://onnx.ai/>
- [12] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Int. Conf. Learning Representations (ICLR)*, 2015. [Online]. Available: <https://arxiv.org/abs/1412.6980>
- [13] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," OpenAI Blog, 2019. [Online]. Available: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf
- [14] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. NAACL-HLT*, 2019, pp. 4171–4186. [Online]. Available: <https://arxiv.org/abs/1810.04805>
- [15] OpenAI, "OpenAI Whisper: Robust speech recognition via large-scale weak supervision," 2022. [Online]. Available: <https://arxiv.org/abs/2212.04356>
- [16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778. <https://doi.org/10.1109/CVPR.2016.90>
- [17] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *IEEE Computational Intelligence Magazine*, vol. 13, no. 3, pp. 55–75, Aug. 2018. <https://doi.org/10.1109/MCI.2018.2840738>
- [18] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. <https://doi.org/10.1162/neco.1997.9.8.1735>
- [19] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2015. <https://arxiv.org/abs/1409.0473>
- [20] J. Li, R. Zhao, J. Gong, and M. Hasegawa-Johnson, "Acoustic modeling for automatic speech recognition using deep neural networks: A review," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, Nov. 2012. <https://doi.org/10.1109/MSP.2012.2215036>
- [21] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. Int. Conf. Artificial Intelligence and Statistics (AISTATS)*, 2010, pp. 249–256. <https://proceedings.mlr.press/v9/glorot10a.html>

- [22] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2015. <https://arxiv.org/abs/1409.1556>
- [23] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2014. <https://arxiv.org/abs/1312.6114>
- [24] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, ... and K. Kavukcuoglu, "WaveNet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016. <https://arxiv.org/abs/1609.03499>
- [25] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, Mar. 1994. <https://doi.org/10.1109/72.279181>

Plagiarism report

Major Project report

ORIGINALITY REPORT

12%

SIMILARITY INDEX

11%

INTERNET SOURCES

7%

PUBLICATIONS

8%

STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Manipal University Jaipur Online Student Paper	1%
2	ouci.dntb.gov.ua Internet Source	1%
3	Submitted to Manipal University Student Paper	1%
4	Submitted to University of Surrey Student Paper	<1%
5	ds.libol.fpt.edu.vn Internet Source	<1%
6	arxiv.org Internet Source	<1%
7	www.arxiv-vanity.com Internet Source	<1%
8	www.extrica.com Internet Source	<1%
9	koreascience.or.kr Internet Source	<1%
10	oaktrust.library.tamu.edu Internet Source	<1%
11	Submitted to Colorado State University Fort Collins Student Paper	<1%
12	hrcak.srce.hr Internet Source	<1%

13	www.etasr.com Internet Source	<1 %
14	github.com Internet Source	<1 %
15	huggingface.co Internet Source	<1 %
16	ijrcait.com Internet Source	<1 %
17	ijisrt.com Internet Source	<1 %
18	eprints.whiterose.ac.uk Internet Source	<1 %
19	rajanjitenpatel.github.io Internet Source	<1 %
20	www.techscience.com Internet Source	<1 %
21	Submitted to University of Bolton Student Paper	<1 %
22	hdl.handle.net Internet Source	<1 %
23	Submitted to Durban University of Technology Student Paper	<1 %
24	doi.org Internet Source	<1 %
25	Submitted to National Institute of Technology, Silchar Student Paper	<1 %
26	developer.nvidia.com Internet Source	<1 %

27	iris.unica.it Internet Source	<1 %
28	liny.csie.nctu.edu.tw Internet Source	<1 %
29	pingpdf.com Internet Source	<1 %
30	www.coursehero.com Internet Source	<1 %
31	Submitted to Trinity College Dublin Student Paper	<1 %
32	Anitha S. Pillai, Roberto Tedesco. "Machine Learning and Deep Learning in Natural Language Processing", CRC Press, 2023 Publication	<1 %
33	docshare.tips Internet Source	<1 %
34	ijritcc.org Internet Source	<1 %
35	Submitted to Virginia Polytechnic Institute and State University Student Paper	<1 %
36	slashdot.org Internet Source	<1 %
37	Submitted to International Institute of Information Technology, Hyderabad Student Paper	<1 %
38	www.db-thueringen.de Internet Source	<1 %
39	Submitted to University of Bradford Student Paper	<1 %

Submitted to Birkbeck College

40	Student Paper	<1 %
41	digitalcommons.usf.edu Internet Source	<1 %
42	eprints.utm.my Internet Source	<1 %
43	european-language-equality.eu Internet Source	<1 %
44	www.gpcet.ac.in Internet Source	<1 %
45	www.mdpi.com Internet Source	<1 %
46	Changhao Chenli, Boyang Li, Yiyu Shi, Taeho Jung. "Energy-recycling Blockchain with Proof-of-Deep-Learning", 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), 2019 Publication	<1 %
47	Tejas Atul Khare, Anuradha C. Phadke. "Automated Crop Field Surveillance Using Computer Vision", 2020 IEEE International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER), 2020 Publication	<1 %
48	eprints.nottingham.ac.uk Internet Source	<1 %
49	globalowls.com Internet Source	<1 %
50	www.isca-archive.org Internet Source	<1 %
	www2.eecs.berkeley.edu	

51	Internet Source	<1 %
52	Jyoti Mishra, Timothy Malche, Amit Hirawat. "Embedded Intelligence for Smart Home Using TinyML Approach to Keyword Spotting", ECSA-11, 2024 Publication	<1 %
53	Manish Tiwari, Ghanshyam Singh, Ankur Saharia. "Intelligent Photonics Systems - Technology and Applications", CRC Press, 2025 Publication	<1 %
54	S. Solomon Darnell, Rupert W. Overall, Andrea Guarracino, Vincenza Colonna et al. "Creating a Biomedical Knowledge Base by Addressing GPT's Inaccurate Responses and Benchmarking Context", Qeios Ltd, 2024 Publication	<1 %
55	dirpub.org Internet Source	<1 %
56	dokumen.pub Internet Source	<1 %
57	ghostscript.com Internet Source	<1 %
58	link.springer.com Internet Source	<1 %
59	workshop2017.iwslt.org Internet Source	<1 %
60	www.degruyter.com Internet Source	<1 %
61	Submitted to University of Sheffield Student Paper	<1 %

62	aaltodoc2.org.aalto.fi Internet Source	<1 %
63	archive.org Internet Source	<1 %
64	dspace.daffodilvarsity.edu.bd:8080 Internet Source	<1 %
65	export.arxiv.org Internet Source	<1 %
66	Submitted to islingtoncollege Student Paper	<1 %
67	lgurjcsit.lgu.edu.pk Internet Source	<1 %
68	nottingham-repository.worktribe.com Internet Source	<1 %
69	scyr.kpi.fei.tuke.sk Internet Source	<1 %
70	unilight.github.io Internet Source	<1 %
71	upcommons.upc.edu Internet Source	<1 %
72	www.diva-portal.org Internet Source	<1 %

Exclude quotes Off
Exclude bibliography Off

Exclude matches Off