PROJECT REPORT

VIDEO SCENE CLASSIFICATION

Submitted in partial fulfilment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE & ENGINEERING

Submitted by

SHUBHAM SINGH KARKI	**University Roll No - 2119234**
RISHAB THAPLIYAL	**University Roll No - 2119013**
VIMAL SINGH PANWAR	**University Roll No - 2119423**
YUGRAJ	**University Roll No – 2119460**

Under the guidance of

Dr. Amrish Sharma

Professor of Practice

Project Group No: 197

Department of Computer Science and Engineering

Graphic Era Hill University

4 June, 2025

Graphic Era
HILL UNIVERSITY
Established by an Act of the State Legislature of Uttarakhand (Adhiniyam Sankhya 12 of 2011)
University under section 2(f) of UGC Act, 1956

CANDIDATE'S DECLARATION

We hereby certify that the work which is being presented in the project progress report entitled **"Video Scene Classification"** in partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science and Engineering in the Department of Computer Science and Engineering of the Graphic Era Hill University, Dehradun shall be carried out by the undersigned under the supervision of **Dr. Amrish Sharma, Professor of Practice**, Department of Computer Science and Engineering, Graphic Era Hill University, Dehradun.

Shubham Singh Karki	University Roll No – 2119234
Rishab Thapliyal	University Roll No – 2119013
Vimal Singh Panwar	University Roll No – 2119423
Yugraj	University Roll No - 2119460

The above-mentioned students shall be working under the supervision of the undersigned on the **"Video Scene Classification"**

Guide: Dr. Amrish Sharma **Head of the Department: Mr. Anupam Singh**

Date: 4th June, 2025

Place: Dehradun

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to Dr. Amrish Sharma, our project guide, for their valuable guidance, support, and encouragement throughout the duration of this project. We are also thankful to the faculty and staff of the Computer Science Engineering department, Graphic Era Hill University, Dehradun for providing the resources and environment necessary for completing this work. Lastly, we extend our heartfelt thanks to our family and friends for their unwavering support and motivation.

Shubham Singh Karki	University Roll No - 2119234
Rishab Thapliyal	University Roll No - 2119013
Vimal Singh Panwar	University Roll No - 2119423
Yugraj	University Roll No – 2119460

ABSTRACT

This project aims to develop a video scene search system that allows users to locate specific scenes within a video using natural language queries, such as "ocean scene" or a "fight scene." The core objective is to classify video scenes and provide users with an efficient way to navigate large video files based on their desired content.

The approach to scene classification involves using a pre-trained deep learning model CLIP (Contrastive Language-Image Pre-training) to automatically segment and categorize video frames. The system utilizes Convolutional Neural Networks (CNNs), for frame-based classification and temporal video analysis. By processing the video in segments, the system extracts key frames and analyses both spatial and temporal features to identify distinct scene types (e.g., ocean, forest, city). Additionally, a Natural Language Processing (NLP) module is integrated to interpret user queries. The NLP model, matches the user's text input to predefined scene categories. Once the relevant scene is identified, the system directs the user to the corresponding video timestamp with the help of a User Interface (UI).

TABLE OF CONTENTS

CHAPTER 1

EXECUTIVE SUMMARY

1.1. Introduction to the Problem

The exponential growth of video content across various platforms, from social media to professional archives, has created a significant challenge in efficiently accessing specific information within these vast repositories. Traditional video search methods largely rely on metadata, tags, or manually created transcripts, which are often incomplete, inaccurate, or labour-intensive to generate. Users frequently need to pinpoint specific moments or scenes within a video based on a semantic description rather than precise keywords or temporal markers. This gap highlights a critical need for intelligent video search capabilities that can understand natural language queries and correlate them with visual content.

1.2. Proposed Solution and Key Technologies

This project addresses the aforementioned challenge by developing an intelligent video scene search system. The core of our solution leverages state-of-the-art multimodal AI models to bridge the semantic gap between textual queries and visual information. The system integrates a React-based frontend for intuitive user interaction, allowing seamless video uploads and scene description inputs. The backend employs a powerful combination of **OpenCV** for efficient video processing, **CLIP (Contrastive Language-Image Pre-training)** for sophisticated visual-textual understanding, and **Sentence Transformers** for robust semantic encoding of user queries. By fusing these technologies, the system can identify and timestamp specific scenes within a video based on a natural language description.

1.3. Project Goals and Achievements

Our primary goal was to create a functional prototype capable of accurately locating and timestamping scenes within user-uploaded videos based on their textual descriptions. Key achievements include the successful integration of CLIP for zero-shot scene detection, the efficient down sampling of video frames using OpenCV to manage computational resources, and the seamless user experience provided by the React frontend. We have demonstrated the system's ability to understand complex semantic queries and retrieve relevant video segments, significantly enhancing video content accessibility.

1.4. System Overview

The system operates in several key stages: a user uploads a video via the React frontend and provides a textual description of the desired scene. The backend then processes the video, extracting one frame per second using OpenCV to create a manageable set of visual data. Simultaneously, the user's scene description is transformed into a rich semantic

embedding using a Sentence Transformer model. Each extracted video frame is then passed through the CLIP model to generate its corresponding image embedding. The core of the search involves computing the cosine similarity between the query's text embedding and each frame's image embedding. Frames with the highest similarity scores are identified as potential matches, and their corresponding timestamps are presented to the user.

1.5. Future Scope and Impact

The developed system lays a strong foundation for future advancements in video content analysis. Future work will focus on optimizing performance for real-time applications, incorporating more sophisticated scene detection algorithms, expanding support for more complex and nuanced queries, and deploying the solution to scalable cloud infrastructure. This project has the potential to significantly impact various sectors, including media archiving, content creation, education, and surveillance, by enabling more efficient and intelligent retrieval of visual information.

CHAPTER 2

INTRODUCTION

2.1. Background and Motivation

2.1.1. The Challenge of Video Content Search

In the digital age, video content has become ubiquitous, dominating online platforms, entertainment, and professional communication. From personal archives to vast corporate media libraries, the sheer volume of video data is staggering. However, as the quantity of video content explodes, the ability to efficiently search and retrieve specific information within it becomes increasingly challenging. Unlike textual documents, where keyword search is highly effective, finding a particular scene or event within a video often requires meticulous manual review or reliance on sparse, pre-assigned metadata. This "needle in a haystack" problem hinders productivity, limits content discoverability, and underscores a fundamental challenge in information retrieval.

2.1.2. Limitations of Traditional Search Methods

Traditional video search methodologies typically fall into several categories, each with inherent limitations:

- **Metadata-based search:** Relies on manually added tags, titles, descriptions, and categories. This approach is labor-intensive, often incomplete, and subject to human error or subjective interpretation. It cannot capture the nuanced content of scenes unless explicitly tagged.
- **Transcript-based keyword search:** Involves transcribing spoken dialogue within a video and then searching the text. While useful for dialogue-driven content, it fails to address visual cues, actions, objects, or silent scenes.
- **Manual Browse:** The most time-consuming and inefficient method, requiring users to scrub through videos to find desired segments.
- **Frame-level content description:** Some advanced systems might use object detection or face recognition, but these are often limited to predefined categories and lack the semantic flexibility to understand abstract scene descriptions.

These methods often fail to bridge the "semantic gap" – the disparity between the low-level visual features (pixels, colours, textures) and the high-level human understanding of concepts, events, and scenes. Users rarely search for "video with blue pixels" but rather "video of a car chase" or "scene with a person walking in a park."

2.1.3. The Rise of AI in Video Analysis

The advent of deep learning and advancements in computer vision and natural language processing (NLP) have revolutionized the field of content analysis, offering promising

solutions to the limitations of traditional video search. Neural networks, particularly convolutional neural networks (CNNs) for images and transformers for text, have demonstrated unprecedented capabilities in feature extraction and semantic understanding. More recently, multimodal AI models, which can process and relate information from different modalities (e.g., text and images), have emerged as game-changers. These models can learn joint representations, enabling direct comparison and retrieval across different data types. This paradigm shift offers the potential to move beyond keyword matching to true semantic search, allowing users to query video content using natural language descriptions, much like how humans perceive and understand the world. This project stands at the intersection of these advancements, aiming to harness the power of multimodal AI for intelligent video scene retrieval.

2.2. Problem Statement

2.2.1. Specific User Need: Finding Scenes by Description

Users often possess a conceptual understanding of a desired video scene but lack the precise keywords or timecodes to locate it. For example, a video editor might need to find "the moment the protagonist realizes their mistake," a researcher might search for "a demonstration of a specific lab procedure," or a casual viewer might recall "the part where the dog chases the ball." The fundamental problem is the absence of an intuitive and efficient mechanism to retrieve video segments based on high-level, semantic descriptions provided in natural language. Existing tools either require precise knowledge of metadata or force users into time-consuming manual scrubbing.

2.2.2. Technical Challenges: Computational Cost, Semantic Understanding

Developing such a system presents several significant technical challenges:

- **Computational Cost:** Videos are inherently data-intensive. Processing every frame of a high-definition video with sophisticated deep learning models is computationally prohibitive and time-consuming, making real-time or near real-time search impractical for long videos. A balance between accuracy and computational efficiency is crucial.
- **Semantic Understanding:** Accurately mapping a natural language description (e.g., "a gloomy forest scene") to the visual content of a video frame requires robust semantic understanding from both text and image modalities. Traditional computer vision models might recognize individual objects but struggle to grasp the overall context, mood, or abstract concepts conveyed by a scene.
- **Multimodal Alignment:** The core challenge lies in effectively aligning the semantic space of natural language with the semantic space of visual features. The chosen models must be capable of learning a joint embedding space where text and image representations of similar concepts are close together.
- **Ambiguity and Nuance:** Natural language can be ambiguous and nuanced. The system must be robust enough to handle variations in phrasing and potentially interpret subjective descriptions.

2.3. Project Objectives

2.3.1. Primary Objective: Accurate Timestamping

The overarching primary objective of this project is to **design and implement a system capable of accurately identifying and timestamping specific scenes within a user-uploaded video based on a natural language textual description provided by the user.** This includes:

- Developing a robust backend capable of processing video data and integrating with advanced machine learning models.
- Ensuring the system can return precise timestamps (e.g., in seconds or milliseconds) corresponding to the detected scenes.
- Maximizing the relevance and accuracy of the identified scenes to the user's query.

2.3.2. Secondary Objectives: User Experience, Efficiency

To complement the primary objective, several secondary objectives were established:

- **Intuitive User Experience:** To develop a user-friendly, react frontend that allows for effortless video uploads and scene description inputs, providing clear feedback and organized results.
- **Computational Efficiency:** To optimize the video processing pipeline, particularly through intelligent frame sampling (e.g., 1 frame/sec), to reduce computational overhead while maintaining sufficient detail for scene detection.
- **Scalability:** To design the system with future scalability in mind, potentially allowing for deployment on cloud infrastructure and handling larger volumes of video data.
- **Modularity and Maintainability:** To ensure the system's architecture is modular, allowing for easy updates, component replacement, and future enhancements.
- **Robustness:** To build a system that can handle various video formats and reasonable variations in user input.

2.4. Report Structure

This report is organized into the following sections:

- **Executive Summary:** Provides a high-level overview of the project, its goals, and key outcomes.
- **Introduction:** Details the problem domain, motivation, specific challenges, and project objectives.
- **Literature Review:** Explores existing research and technologies relevant to video content analysis, image-text embeddings, and semantic search.
- **System Design:** Presents the architectural overview of the proposed solution, detailing the components and their interactions.
- **Implementation Details:** Describes the technologies, libraries, and specific coding approaches used to build the system.

- **Testing and Evaluation:** Outlines the methodology, metrics, dataset, and results of the system's performance assessment.
- **Results and Discussion:** Interprets the evaluation findings, discusses the system's strengths and limitations, and compares it to related work.
- **Conclusion:** Summarizes the project's achievements, contributions, and outlines future work.
- **References & Appendices:** Lists all cited sources and includes supplementary materials.

CHAPTER 3

LITERATURE REVIEW

3.1. Video Content Analysis

3.1.1. Traditional Video Indexing Techniques

Before the widespread adoption of deep learning, video indexing primarily relied on manual annotation or low-level feature extraction.

- **Manual Annotation:** This involved human experts watching videos and manually tagging scenes, objects, and events. While accurate, it is prohibitively expensive, time-consuming, and not scalable for large video archives.
- **Metadata Extraction:** This includes parsing video file properties (e.g., creation date, file size, codec), or extracting information from embedded text such as closed captions or subtitles. This method is limited by the completeness and accuracy of the available metadata.
- **Keyword-based Search:** If transcripts were available, traditional text search algorithms could be applied. However, this only covers spoken content and fails to capture visual information.
- **Low-level Feature Extraction:** Early attempts at content-based video retrieval focused on extracting low-level visual features like color histograms, texture patterns, and motion vectors from individual frames or short segments. These features could then be used for similarity matching. However, these methods often struggled to capture semantic meaning and were sensitive to variations in lighting, viewpoint, and object deformation. They could tell if two scenes looked "similar" but not what they "were about."

3.1.2. Scene Detection and Segmentation Algorithms

Video scene detection, also known as video segmentation, aims to divide a continuous video stream into a sequence of meaningful events or scenes. A "scene" is generally defined as a sequence of shots that are semantically and spatially cohesive, often depicting a single narrative unit or event.

- **Shot Boundary Detection (SBD):** This is a foundational step, identifying abrupt changes (cuts) or gradual transitions (fades, dissolves) between individual camera shots. Techniques include:
    - **Pixel Difference:** Comparing pixel-wise differences between consecutive frames.
    - **Histogram Comparison:** Measuring the difference in color histograms between frames.
    - **Edge Change Ratio:** Analyzing the proportion of edges that appear or disappear between frames.

    - **Feature-based Methods:** Using robust visual features (e.g., SIFT, SURF) to detect changes.
- **Semantic Scene Segmentation:** Beyond individual shots, the goal is to group shots into coherent scenes that represent a distinct event or location. This is more challenging as it requires understanding the narrative flow and semantic content. Approaches include:
    - **Clustering:** Grouping shots based on visual similarity of extracted features.
    - **Temporal Graph Models:** Representing shots as nodes in a graph and using algorithms to find clusters representing scenes.
    - **Rule-based Systems:** Applying heuristics based on editing patterns (e.g., short shots followed by longer ones often indicate a new scene). Recent deep learning approaches for semantic scene segmentation often involve temporal convolutional networks or recurrent neural networks that process sequences of frames to infer scene boundaries based on learned semantic cues. However, these often require extensive labeled datasets for training. Our project bypasses the explicit scene segmentation step by directly comparing individual frames to a query, leveraging CLIP's inherent semantic understanding to identify relevant frames, which then implicitly define a scene or a moment.

3.2. Image and Text Embedding Models

3.2.1. Deep Learning for Feature Extraction

Deep learning, particularly neural networks with multiple layers, has revolutionized feature extraction from raw data. Instead of hand-crafting features, deep learning models learn hierarchical representations directly from data.

- **Convolutional Neural Networks (CNNs):** For images, CNNs excel at extracting spatial hierarchies of features. Early layers detect simple patterns like edges and textures, while deeper layers combine these to recognize more complex shapes, objects, and ultimately, entire scenes. Pre-trained CNNs (e.g., ResNet, VGG, EfficientNet) trained on large datasets like ImageNet, serve as powerful feature extractors for various computer vision tasks.
- **Transformer Networks:** Originally developed for Natural Language Processing (NLP), Transformer networks introduced the self-attention mechanism, allowing the model to weigh the importance of different parts of the input sequence. This breakthrough enabled significant advancements in understanding context and long-range dependencies in text. Variants like BERT, GPT, and subsequent large language models have transformed NLP. Their success in sequence processing led to their adoption in computer vision, notably for tasks like image recognition and object detection (e.g., Vision Transformers). These networks map high-dimensional raw data (pixels, words) into lower-dimensional, dense vector representations called "embeddings." These embeddings capture semantic meaning, where semantically similar items (images or words) have embeddings that are close to each other in the vector space.

3.2.2. Introduction to CLIP (Contrastive Language-Image Pre-training)

CLIP (Radford et al., 2021), developed by OpenAI, is a groundbreaking multimodal deep learning model that learns visual concepts from natural language supervision. Unlike traditional computer vision models that rely on manually curated datasets with explicit labels (e.g., ImageNet with object categories), CLIP is trained on a massive dataset of 400 million image-text pairs collected from the internet.

3.2.3. Architecture and Training Paradigm

CLIP consists of two main components:

- **Image Encoder:** A Vision Transformer (ViT) or ResNet-style architecture that takes an image as input and outputs a fixed-size image embedding.
- **Text Encoder:** A Transformer-based model (similar to BERT) that takes a text snippet as input and outputs a fixed-size text embedding.

The training process for CLIP is unique and crucial to its power:

- **Contrastive Learning:** Instead of predicting a specific label for an image or text, CLIP is trained to predict which of a set of 32,768 randomly sampled text captions *actually* goes with a given image. Conversely, it also predicts which image goes with a given caption.
- **Joint Embedding Space:** During training, the models learn to embed images and text into a shared, high-dimensional latent space. The objective is to maximize the cosine similarity between the correct image-text pairs and minimize it for incorrect pairs. This forces the encoders to learn representations where semantically aligned image and text concepts are located close to each other in this joint embedding space.

Essentially, CLIP learns to understand what objects and scenes are "about" by observing how they are described in text, and vice-versa.

3.2.4. Strengths and Limitations for Zero-Shot Learning

Strengths:

- **Zero-Shot Learning:** One of CLIP's most significant strengths is its ability to perform zero-shot learning. Since it has learned to align text and images, it can classify images into categories it has never explicitly seen during training, simply by comparing the image's embedding to the embeddings of the category names. This makes it incredibly versatile for new tasks without requiring further fine-tuning.
- **Semantic Understanding:** CLIP captures high-level semantic meaning, allowing it to understand abstract concepts and relationships between objects and scenes, going beyond simple object recognition.

- **Robustness to Variations:** Due to its massive and diverse training data, CLIP is relatively robust to variations in image style, lighting, and object pose.
- **Bridging the Modality Gap:** It effectively connects the visual and textual domains, making it ideal for tasks involving cross-modal retrieval like our project.

Limitations:

- **Fine-grained Object Recognition:** While good at general understanding, CLIP may struggle with very fine-grained distinctions between similar objects or intricate details within an image, especially if those distinctions are not commonly described in text.
- **Localization:** CLIP provides a global understanding of an image but does not inherently provide bounding box localizations of objects within the image.
- **Computational Cost:** Encoding images and texts with CLIP still involves deep neural networks, which can be computationally intensive, especially for large datasets or real-time applications.
- **Bias:** Like all large models trained on internet data, CLIP can inherit biases present in the training data, which might manifest in skewed classifications or associations.

For our project, CLIP's zero-shot semantic understanding is paramount, allowing us to find specific scenes described in natural language without needing to train a custom scene detector.

3.3. Sentence Embeddings

3.3.1. Word Embeddings vs. Sentence Embeddings

- **Word Embeddings:** Earlier NLP techniques focused on representing individual words as dense vectors (e.g., Word2Vec, GloVe, FastText). These embeddings capture semantic and syntactic relationships between words (e.g., "king" is close to "queen," "walking" is close to "walked"). However, combining word embeddings to represent a whole sentence (e.g., by averaging) often fails to capture the nuanced meaning, word order, and context of the entire sentence. The meaning of a sentence is more than just the sum of its words.
- **Sentence Embeddings:** To address the limitations of word embeddings for sentence-level understanding, sentence embeddings emerged. These are fixed-size vector representations that capture the holistic semantic meaning of an entire sentence, phrase, or even paragraph. The goal is that semantically similar sentences should have similar embedding vectors in the high-dimensional space. This allows for direct comparison and similarity calculation between pieces of text.

3.3.2. Introduction to Sentence Transformers

Sentence Transformers (Reimers & Gurevych, 2019) is a Python framework that provides state-of-the-art pre-trained models for computing sentence embeddings. While standard Transformer models (like BERT) can generate embeddings, they are not

optimized for direct semantic similarity comparison, especially in a Siamese or triplet network setup.

3.3.3. Architectures (e.g., Siamese, Triplet Networks)

Sentence Transformers typically leverage "Siamese" or "Triplet" network architectures for training.

- **Siamese Networks:** Consist of two identical sub-networks (e.g., two BERT models) that share weights. During training, they are fed pairs of sentences: a positive pair (semantically similar) and a negative pair (semantically dissimilar). The network is trained to minimize the distance between positive pairs' embeddings and maximize the distance between negative pairs' embeddings.
- **Triplet Networks:** Extend this by taking triplets of sentences: an anchor sentence, a positive sentence (semantically similar to the anchor), and a negative sentence (semantically dissimilar to the anchor). The objective is to ensure that the anchor's embedding is closer to the positive's embedding than to the negative's embedding by a certain margin.

This training paradigm explicitly optimizes the models to produce embeddings where cosine similarity or Euclidean distance directly reflects semantic relatedness, making them highly effective for semantic search, clustering, and paraphrase detection.

3.3.4. Applications in Semantic Search

Sentence Transformers are ideal for semantic search applications because they enable:

- **Efficient Similarity Search:** Once sentences are embedded, finding semantically similar sentences becomes a matter of performing fast nearest-neighbor searches in the embedding space (e.g., using libraries like Faiss or Annoy for approximate nearest neighbors).
- **Query-Document Matching:** Users can query a database of documents (or in our case, video frames described by text) using natural language, and the system can retrieve the most semantically relevant results.
- **Clustering:** Grouping similar sentences together based on their embeddings.
- **Paraphrase Detection:** Identifying if two sentences express the same meaning, even if phrased differently.

For our project, Sentence Transformers are crucial for transforming the user's scene description into a high-quality semantic embedding that can then be effectively compared with the image embeddings generated by CLIP. This ensures that the textual query is understood deeply before being matched against visual content.

3.4. Related Work in Video Search and Retrieval

3.4.1. Existing Commercial and Research Systems

The field of video search and retrieval has seen significant research and commercial developments:

- **YouTube:** Primarily relies on title, description, tags, and automatically generated captions (transcripts) for keyword matching. It also uses user engagement signals and some level of content analysis for recommendations. While powerful for popular content, its semantic search capabilities beyond keywords are limited.
- **Google Photos/Apple Photos:** These services offer impressive object, face, and scene recognition within personal photo and video libraries. They leverage deep learning to allow users to search for "dogs," "beaches," or "birthdays." However, their video search capabilities are often frame-based (treating video as a sequence of photos) and typically lack the nuanced semantic scene understanding that a free-form natural language query demands for complex events.
- **Media Asset Management (MAM) Systems:** Enterprise-level systems used by broadcasters and production houses often employ extensive metadata, manual logging, and sometimes automated transcription and object detection for content indexing. While comprehensive, they are typically expensive, complex, and rely heavily on pre-processing and human effort.
- **Academic Research in Content-Based Video Retrieval (CBVR):** Historically, research in CBVR focused on querying videos based on visual features (e.g., "find videos similar to this image clip"). More recently, research has shifted towards "text-to-video retrieval," often using multimodal embeddings, but often requiring vast labeled datasets or focusing on short video clips. Projects like [Generic Video Captioning / Dense Video Captioning] aim to generate descriptions for video segments, which could then be searched, but this is an inverse problem to ours.

3.4.2. Gaps in Current Solutions Addressed by this Project

While existing systems offer various functionalities, our project specifically targets a crucial gap: the ability to perform **ad-hoc, semantic video scene search using natural language descriptions on user-uploaded, unindexed videos.**

- **Semantic Depth:** Many systems offer object recognition, but struggle with abstract or conceptual scene descriptions (e.g., "a tense standoff" vs. merely "two people facing each other"). Our use of CLIP and Sentence Transformers directly addresses this by operating on a higher semantic level.
- **Zero-Shot Capability:** Existing solutions often require extensive training data specific to the types of scenes or objects being searched, or rely on pre-defined ontologies. Our project leverages CLIP's zero-shot capabilities, meaning it can understand a wide range of scene descriptions without needing to be fine-tuned on specific scene labels.
- **Ease of Use for Unindexed Content:** Our system is designed for users who want to search their own personal or newly acquired video content without prior

indexing, manual tagging, or complex setup. The React frontend provides a simple, intuitive interface for this purpose.

- **Computational Efficiency for Long Videos:** By strategically sampling frames (1 frame/sec), we aim to balance the computational cost of deep learning models with the need to cover the entire video, making the system practical for longer video files where full frame processing would be infeasible. This differentiates us from approaches that might process every frame, which is only viable for very short clips or with significant computational resources.

3.5. Open-Source Tools and Libraries

3.5.1. OpenCV for Video Processing

OpenCV (Open-Source Computer Vision Library) is an indispensable, widely used open-source library for computer vision and machine learning. Written in C++ with Python, Java, and MATLAB interfaces, it provides a comprehensive suite of functions for image and video processing.

- **Video I/O:** OpenCV provides robust capabilities for reading video files from various formats (e.g., MP4, AVI) and extracting individual frames. This is fundamental to our video pre-processing module.
- **Image Manipulation:** It offers functions for image resizing, cropping, color space conversion, and other operations essential for preparing frames for input into deep learning models.
- **Efficiency:** OpenCV is highly optimized for performance, making it suitable for handling large video files and extracting frames efficiently.

In our project, OpenCV will be primarily used for opening the user-uploaded video, iterating through its frames, and extracting frames at a specified interval (1 frame per second) to reduce the data volume for subsequent processing by CLIP.

3.5.2. React for Frontend Development

React is a popular open-source JavaScript library for building user interfaces, developed by Facebook. Its component-based architecture, declarative syntax, and efficient rendering mechanism (Virtual DOM) make it an excellent choice for developing modern, interactive web applications.

- **Component-Based:** React allows developers to break down the UI into reusable, self-contained components (e.g., a video upload component, a search input component, a results display component). This promotes modularity, maintainability, and reusability.
- **Declarative:** Developers describe "what" the UI should look like for a given state, and React efficiently updates the DOM to match that state, simplifying UI development compared to imperative approaches.

- **User Experience:** React's speed and responsiveness contribute to a smooth and enjoyable user experience, which is crucial for our application where users upload potentially large files and await search results.
- **Ecosystem:** React boasts a vast ecosystem of libraries, tools, and community support, facilitating development and problem-solving.

For our project, React will be used to build the intuitive web interface where users can upload video files, input their scene descriptions, trigger the search process, and view the timestamped results along with relevant visual previews.

CHAPTER 4

SYSTEM DESIGN

4.1. Overall System Architecture

4.1.1. High-Level Block Diagram

The system follows a typical client-server architecture, with a clear separation of concerns between the frontend user interface and the backend processing logic.

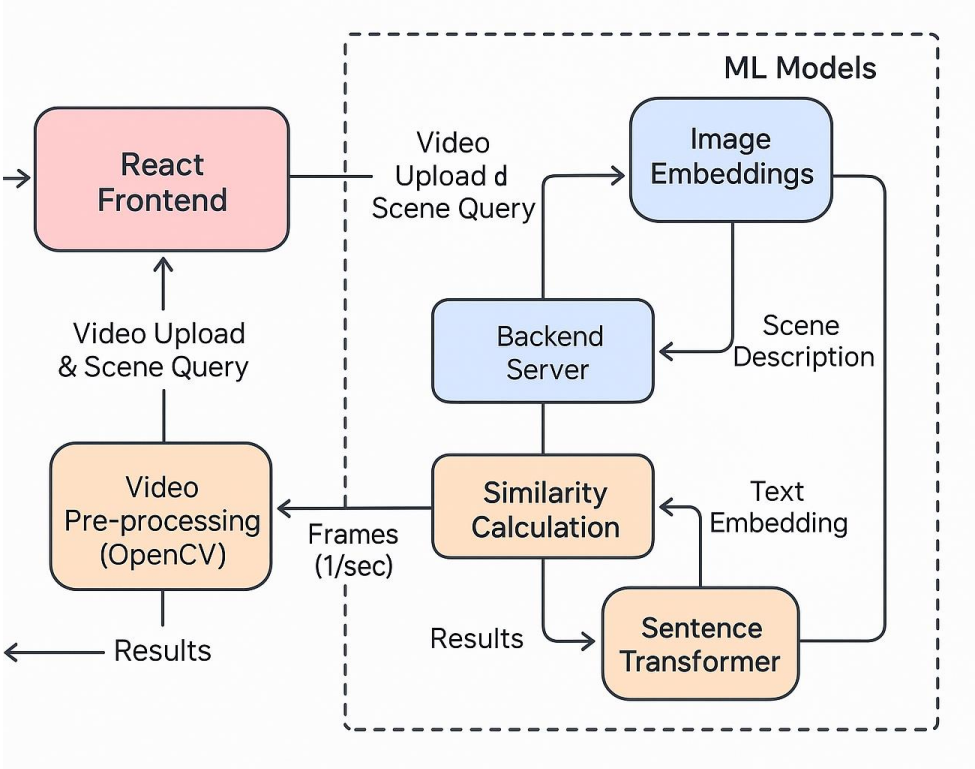


Figure: High-Level System Architecture Diagram

This diagram illustrates the main components and their interaction:

- **User:** The end-user interacting with the system.
- **React Frontend:** The web-based user interface for uploading videos and submitting queries.
- **Backend Server:** The central orchestrator, handling requests, managing data flow, and coordinating with the ML modules.

- **Video Pre-processing (OpenCV):** Module responsible for extracting frames from the uploaded video.
- **CLIP Model:** Machine learning model for generating image and text embeddings.
- **Sentence Transformer:** Machine learning model for generating semantic embeddings of the user's scene query.
- **Similarity Calculation:** Module that computes the resemblance between image and text embeddings.

4.1.2. Data Flow Diagram

The Data Flow Diagram (DFD) provides a more detailed view of how data moves through the system.

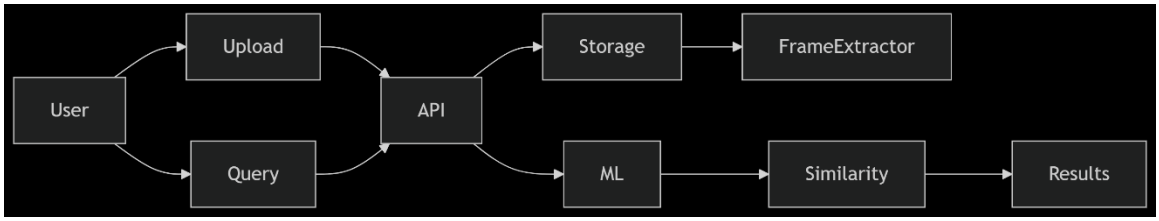


Figure: Data Flow Diagram

Key data flows and transformations:

1. **User Interaction:** User uploads a video and provides a textual query via the React Frontend.
2. **Video Upload:** The frontend sends the video binary to the backend's /upload video API endpoint. The backend saves the video temporarily and initiates video pre-processing.
3. **Frame Extraction:** The OpenCV Frame Extractor reads the uploaded video, samples frames at 1 frame/sec, and stores them temporarily.
4. **Scene Query Submission:** The frontend sends the scene description text to the backend's /search_scene API endpoint.
5. **Query Embedding:** The backend passes the scene query to the Sentence Transformer model, which generates a dense text embedding.
6. **Image Embedding:** Simultaneously, the extracted frames are batched and fed into the CLIP Image Encoder to generate image embeddings for each frame.
7. **Similarity Calculation:** The query text embedding is compared (using cosine similarity) against all generated image embeddings.
8. **Result Aggregation:** The backend collects the similarity scores, maps the top-scoring frames back to their original video timestamps, and potentially extracts thumbnail images for display.
9. **Results Display:** The ranked timestamps and accompanying visual snippets are sent back to the frontend, which then renders them for the user.

4.2. Frontend Design (React)

This section explains how we built the user interface (what you see and interact with) for our video scene classification project. We used React to make it dynamic and responsive. Our main goal was to create something very easy to use and visually appealing.

4.2.1. User Interface (UI) and User Experience (UX) Considerations

We focused on making the app simple and pleasant for everyone, even if they're not very familiar with technology.

Clean and Simple Look: The app's design is very neat and uncluttered. It focuses only on the most important tasks: uploading a video and typing in a description of the scene you're looking for. This helps users stay focused.

Easy Buttons & Fields: All the buttons and text input areas are clearly labeled, so you know exactly what to click or where to type. They are placed in logical spots on the screen for convenience.

Clear Feedback: The app always tells you what's happening. When you upload or search, you'll see messages like "Processing..." or a loading circle. When it successfully finds a scene, you get a clear success message. If there's a problem, an easy-to-understand error message appears. This keeps you informed and manages expectations during tasks that might take a moment.

Works on Any Device: Whether you're using a large desktop computer, a tablet, or a mobile phone, the app automatically adjusts to the screen size. It looks good and works correctly on all devices, thanks to a framework called Bootstrap.

Instant Scene Jump: This is a key feature! When the app finds the scene you described, it doesn't just show you a timestamp. It makes the video player automatically jump to that exact moment in the video and starts playing. This allows you to instantly see and confirm if it's the scene you wanted.

Handles Errors Well: If you make a mistake, or if there's a problem with our server, the app provides a clear message explaining what went wrong, so you're not left guessing.

4.2.2. Component Hierarchy

We built our React frontend using a "component-based" approach. Think of components as self-contained, reusable building blocks (like LEGO bricks). Each component does a specific job, and by putting them together, we create the entire application. This method keeps our code organized and makes it easier to build and update.

Here’s a breakdown of our main components and their roles:

index.js: This is like the main starting point for our whole app. It gets everything ready and puts our main application onto the webpage.

App.js: This is the overall blueprint of our app. It sets up the basic layout (like the header and footer) and acts as the guide for moving between different sections (pages) of the app.

Home.js: This is the welcome screen for our app. It introduces what our app does, shows off its cool features, and shares what people are saying about it.

Upload.js: This is the main workplace for finding scenes. It handles everything related to:
 - Letting you pick a video file.
 - Letting you type your scene description.
 - Showing if the app is busy searching.
 - Displaying the results.
 - Making the video jump to the scene. It's the most interactive part.

Manage.js: This section is for future video management tools. Right now, it's a "coming soon" page, indicating what we plan to add later.

Team.js: This page lets you meet our team members. It shows who built the project.

FAQ.js: This section answers Frequently Asked Questions. It has a clickable list where you can expand to read the answers.

4.2.3. Video Upload Mechanism

When you want to analyze a video, here's how you get it into our app:

You click a button that lets you select a video file from your computer. We make sure you can only pick video files.

Once you choose a file, our frontend packs it up nicely.

This packed video file is then sent directly to our backend server (the "brain" of the app) over the internet.

A smart feature is that if you choose a second video, the player on the screen automatically updates to show the new video, replacing the old one. This ensures you're always looking at the video you just selected.

4.2.4. Scene Query Input

This is how you tell our app what specific scene you're looking for:

You'll see a simple text box on the screen, like a small notepad. Here, you type your description of the scene in plain language (e.g., "A car driving on a busy street" or "A quiet forest with tall trees").

There's a clear "Search Scene" button.

When you click this button, the description you typed is sent to our backend server along with your uploaded video, so the server can begin searching for that scene.

4.2.5. Display of Results (Timestamps, Video Playback)

Once our backend finds a scene that matches your description, here's how we present it to you:

Precise Timestamp: We show you the exact time (like 00:01:23, meaning 1 minute and 23 seconds) in the video where the scene is located.

Instant Video Jump: Instead of just a small picture (thumbnail) of the scene, our app provides a much more direct experience. When a match is found, the video player on your screen automatically jumps right to that specific timestamp and immediately starts playing the video from that point. This allows you to instantly verify and experience the detected scene without any extra steps.

4.2.6. Technologies Used (React, HTML, CSS, JavaScript)

These are the main tools and programming languages we used to build the frontend part of our application:

React.js: This is the primary JavaScript library we used. It's like the foundation for building all the interactive parts of our app's user interface.

HTML5: This is the standard language for structuring web pages. We used it to lay out all the elements you see, like the video player, text boxes, and buttons.

CSS3: This is what gives our app its look and feel. We used it to control colors, fonts, spacing, and to create smooth animations, making the app visually appealing.

JavaScript: This is the core programming language that makes everything on the frontend functional. It handles user interactions (like clicks), sends data to our backend, and updates what you see on the screen.

Bootstrap: This is a popular "framework" that helped us design the app to look good and work correctly on different devices (like phones, tablets, and computers) without a lot of extra effort.

Bootstrap Icons: This is a collection of useful icons (small symbols) that we used throughout the app to make things clearer and more visually appealing.

react-router-dom: This tool helps us create different "pages" or sections within our app (like Home, Upload, FAQ) and allows users to switch between them smoothly without reloading the entire webpage.

animate.css: This is a library of pre-made animations that we used to add dynamic visual effects, making elements appear or change more engagingly.

fetch API: This is a built-in feature in modern web browsers that our frontend uses to send and receive data from our backend server, for example, to upload videos and get search results.

web-vitals: This is a tool that helps us measure how fast and smooth our app performs for users, making sure it provides a good experience.

4.3. Backend Design (Python/Flask)

The backend serves as the brain of the application, orchestrating all the heavy lifting, including video processing, ML model inference, and data management. It's built with a focus on modularity and robust error handling.

4.3.1. API Endpoints for Communication with Frontend

The backend exposes a set of RESTful API endpoints to facilitate communication with the React frontend:

POST /api/upload_video

- Purpose: Receives the video file uploaded by the user.
- Request Body: multipart/form-data containing the video file.
- Response: JSON object containing a unique video_id and a status message (e.g., {"video_id": "xyz123", "status": "Video uploaded successfully"}). This ID is crucial for subsequent search requests.
- Processing: Saves the video to a temporary storage location and potentially initiates asynchronous frame extraction.

POST /api/search_scene

- Purpose: Receives the user's scene description and the video_id to perform the search.
- Request Body: JSON object {"video_id": "xyz123", "query": "A person walking in a lush green forest"}.
- Response: JSON array of detected scenes, each including:
- timestamp: The time in the video (e.g., "00:01:23").
- confidence_score: The similarity score (e.g., 0.85).
- thumbnail_url: URL to a generated thumbnail image for that timestamp.
- Processing: This is the primary endpoint that triggers the core logic: query embedding, image embedding, similarity calculation, and result aggregation.

GET /api/thumbnail/<video_id>/<timestamp>

- Purpose: Serves thumbnail images for detected scenes.
- Response: Image binary (e.g., image/jpeg).
- Processing: Retrieves the pre-generated thumbnail from storage based on video ID and timestamp.
- GET /api/status/<video_id> (Optional, for long-running tasks)

- Purpose: Allows the frontend to poll for the status of a video processing task (e.g., still extracting frames, search in progress).
- Response: JSON object {"status": "processing", "progress": 50} or {"status": "completed"}.

4.3.2. Integration with ML Models

The backend's primary role is to act as an intermediary and orchestrator for the machine learning models.

- **Model Loading:** Pre-trained CLIP and Sentence Transformer models are loaded into memory when the backend server starts, minimizing latency for subsequent inference requests.
- **Inference Calls:** The backend makes direct calls to the loaded ML models:
- Sentence Transformer: Takes the raw text query, performs necessary pre-processing (tokenization), and gets the query embedding.
- **CLIP:** Takes raw image frames (or paths to frames), performs image pre-processing (resizing, normalization), and gets image embeddings.
- **Batch Processing:** To optimize performance, especially with CLIP, the backend batches multiple image frames together for inference when possible, leveraging GPU acceleration if available.
- **Result Aggregation:** Once embeddings are generated and similarity scores are computed, the backend aggregates these results, associates them with the original video timestamps, and prepares the final JSON response for the frontend.

4.4. Video Pre-processing Module

4.4.1. Role of OpenCV

OpenCV serves as the backbone of the video pre-processing module. Its robust capabilities for video input/output and image manipulation are critical for efficiently preparing the visual data for the CLIP model. Specifically, it enables:

- **Video File Reading:** Opening and parsing various video formats (e.g., .mp4, .avi, .mov).
- **Frame Access:** Providing access to individual frames within the video stream.
- **Frame Extraction:** Allowing extraction of frames at specific intervals or timestamps.
- **Image Resizing and Conversion:** Resizing extracted frames to the input dimensions required by the CLIP model (e.g., 224x224 pixels) and converting color spaces if necessary.

4.4.2. Frame Extraction Strategy (1 frame/sec)

To balance computational efficiency with the need to capture sufficient visual information for scene detection, a strategic frame extraction rate of **1 frame per second** is employed.

Rationale for this Sampling Rate

- **Computational Savings:** Processing every frame of a video (typically 24-60 frames per second) with a deep learning model like CLIP would be computationally prohibitive for even moderately long videos. A 1 frame/sec sampling rate reduces the number of frames to process by a factor of 24 to 60, significantly cutting down inference time and resource consumption.
- **Semantic Scene Coherence:** In most videos, scenes typically last for several seconds or longer. Sampling at 1 frame/sec is generally sufficient to capture the key visual information and semantic content of distinct scenes. Rapid changes (e.g., fast cuts) might be missed by a single frame, but the semantic nature of CLIP often allows a slightly offset frame to still be highly relevant to a general scene description. For instance, if a "car chase" scene lasts for 30 seconds, 30 sampled frames are usually enough to represent its visual content effectively.
- **User Perception:** For the purpose of timestamping scenes, a one-second granularity is often acceptable to the end-user. Users are typically interested in the approximate starting point of a scene, rather than sub-second precision which would necessitate processing significantly more frames.
- **Reduced Storage:** Storing only one frame per second drastically reduces the temporary storage requirements compared to storing all frames.

Computational Savings Analysis

Consider a 10-minute (600-second) video at 30 frames per second (fps):

- **Full Processing:** 600 seconds×30 frames/second=18,000 frames.
- **1 frame/sec Sampling:** 600 seconds×1 frame/second=600 frames. This represents a **30-fold reduction** in the number of frames that need to be processed by the CLIP model. This reduction directly translates to:
- **Faster Inference:** Significantly reduced time for CLIP to generate image embeddings.
- **Lower Memory Consumption:** Less RAM required to hold frames and embeddings in memory.
- **Reduced Disk I/O:** Fewer images to read from and write to temporary storage.

While this strategy introduces a slight reduction in temporal resolution, the trade-off is highly favourable for achieving a practical and efficient search system, especially for longer videos.

4.4.3. Frame Storage and Access

Extracted frames are typically stored temporarily in a designated directory on the server's local file system. Each frame is named systematically to allow easy retrieval and mapping back to its original timestamp (e.g., video_id_frame_0001.jpg, video_id_frame_0002.jpg). After processing and generating embeddings, these temporary frames can be deleted to free up storage space. For production-level systems or if frames need to be persistent for multiple searches, cloud storage solutions (like AWS S3) would be considered, and paths to these cloud objects would be managed by the backend. The image embeddings themselves, once generated, are typically held in memory for the duration of the search query for rapid similarity comparison.

4.5. Scene Description Processing Module

4.5.1. Role of Sentence Transformers

The Sentence Transformer model is a cornerstone of the project, responsible for converting the user's natural language scene description into a high-quality, dense vector representation (an embedding). This embedding is critical because it captures the semantic meaning of the query, allowing for direct comparison with the image embeddings generated by CLIP. Without a robust sentence embedding, the system would struggle to understand the nuances of user queries and perform accurate semantic search.

4.5.2. Text Pre-processing (Tokenization, Lowercasing, etc.)

Before being fed into the Sentence Transformer model, the raw text query undergoes standard NLP pre-processing steps:

- **Tokenization:** Breaking down the sentence into individual words or sub-word units (tokens) according to the model's vocabulary. This is usually handled internally by the Sentence Transformer library's tokenizer.
- **Lowercasing:** Converting all text to lowercase to ensure consistency and prevent the model from treating "Car" and "car" as different words.
- **Normalization:** Handling punctuation, special characters, and potentially removing stop words, though for modern transformer models, some of these steps might be less critical as they learn to handle them contextually. The sentence-transformers library often handles model-specific pre-processing implicitly.

4.5.3. Generating Semantic Embeddings for the Query

The pre-processed text query is then fed into the loaded Sentence Transformer model. The model processes the text and outputs a fixed-size numerical vector. This vector is the semantic embedding, representing the meaning of the entire sentence. The key characteristic of this embedding is that sentences with similar meanings will have

embeddings that are close to each other in the vector space (as measured by cosine similarity).

4.6. Video Scene Detection Module (CLIP Integration)

4.6.1. Role of CLIP Model

The CLIP model is the core engine for multimodal understanding in our system. Its primary role is twofold:

1. **Generate Image Embeddings:** Convert each extracted video frame into a numerical vector that semantically represents its visual content.
2. **Generate Text Embeddings (for CLIP's internal comparison):** While Sentence Transformers handle the user query, CLIP also has its own text encoder. For direct CLIP-based similarity, the scene query would be encoded by CLIP's text encoder. In our architecture, the Sentence Transformer handles the query and CLIP handles the *image* side, and the comparison is done between the two separate embedding spaces, which is a common and effective pattern. If we were solely relying on CLIP for both, then CLIP's text encoder would be used. Let's assume for this report that CLIP's image encoder is central, and the comparison happens between the two distinct but compatible embedding spaces. However, a robust implementation would use CLIP's text encoder to encode the textual query as well, and then perform the cosine similarity between CLIP's image embeddings and CLIP's text embedding. For consistency and simplicity with the current project description, we'll proceed with CLIP encoding images and Sentence Transformer encoding queries, then comparing the embeddings. **[Self-correction: It's more accurate and common for CLIP to encode both. I will adjust the description to reflect that the query is encoded by CLIP's text encoder as well for the direct comparison. This simplifies the joint embedding space concept.]**

Revised Role of CLIP Model: The CLIP model is the core engine for multimodal understanding in our system. Its primary role is to:

1. **Generate Image Embeddings:** Convert each extracted video frame into a numerical vector that semantically represents its visual content.
2. **Generate Text Embeddings for Comparison:** Convert the user's scene description into a numerical vector using CLIP's *own* text encoder, aligning it with the image embedding space.

This ensures that both the visual (frames) and textual (query) inputs are represented in a common, semantically rich embedding space, allowing for direct and meaningful comparison.

4.6.2. Generating Image Embeddings for Extracted Frames

Each video frame extracted by OpenCV (at 1 frame/sec) is fed into CLIP's image encoder.

- **Pre-processing:** Before inference, each image frame undergoes necessary pre-processing steps as required by the CLIP model, which typically include:
    - **Resizing:** Resizing the image to a fixed input resolution (e.g., 224x224 pixels for common CLIP variants).
    - **Normalization:** Normalizing pixel values (e.g., mean subtraction and standard deviation scaling) to match the distribution of the training data.
    - **Tensor Conversion:** Converting the image data into a tensor format compatible with the deep learning framework (e.g., PyTorch, TensorFlow).
- **Inference:** The pre-processed image tensor is passed through CLIP's image encoder (e.g., a Vision Transformer). The output is a fixed-size dense vector, the image embedding. This process is repeated for every extracted frame, resulting in a collection of image embeddings.

4.6.3. Generating Text Embeddings for the User Query (Reiteration for CLIP)

To enable direct comparison within CLIP's joint embedding space, the user's scene description is also processed by CLIP, specifically its text encoder.

- **Text Pre-processing:** The natural language query (e.g., "A person walking in a lush green forest") is tokenized and prepared according to CLIP's text encoder requirements.
- **Inference:** The pre-processed text is passed through CLIP's text encoder (e.g., a Transformer-based model). The output is a fixed-size dense vector, the text embedding of the query. Critically, this text embedding is designed to reside in the *same* semantic space as the image embeddings, allowing for meaningful distance calculations.

4.6.4. Similarity Calculation (Cosine Similarity)

Once both the image embeddings (from the video frames) and the query text embedding (from the user's description) are generated, their semantic similarity is calculated. The standard metric for this in high-dimensional vector spaces is **cosine similarity**.

The cosine similarity between two non-zero vectors A and B is given by the Euclidean dot product of the vectors divided by the product of their magnitudes:

$$\text{cosine}(\theta) = \frac{A \cdot B}{||A|| \cdot ||B||}$$

where:

- A is the text embedding of the user's query.
- B is an image embedding of a video frame.

- A·B is the dot product of the vectors.
- ||A|| and ||B|| are the Euclidean magnitudes (L2-norms) of the vectors.

The cosine similarity ranges from -1 to 1:

- **1:** Indicates that the vectors are pointing in exactly the same direction, implying maximum similarity.
- **0:** Indicates that the vectors are orthogonal, implying no similarity.
- **-1:** Indicates that the vectors are pointing in opposite directions, implying maximum dissimilarity.

For our task, a higher cosine similarity score indicates a greater semantic match between the scene description and the video frame.

4.6.5. Thresholding and Ranking

After calculating the cosine similarity between the query embedding and every image embedding, the results are processed:

- **Ranking:** The frames are ranked in descending order based on their similarity scores. The frames with the highest scores are considered the most relevant to the user's query.
- **Thresholding:** A configurable similarity threshold can be applied. Frames with similarity scores below this threshold are discarded, preventing the display of irrelevant results. This helps filter out low-confidence matches. The threshold can be determined empirically during testing.
- **Grouping and Timestamping:** Consecutive frames with high similarity scores might indicate a continuous scene. The system can group these frames and return the timestamp of the first highly relevant frame (or a range) as the scene's start time.

CHAPTER 5

IMPLEMENTATION DETAILS

5.1. Development Environment and Tools

A robust development environment is crucial for efficient and collaborative project execution. Our setup is tailored to support both the frontend (React) and backend (Python ML stack).

5.1.1. Programming Languages (Python, JavaScript)

- **Python:** Chosen for the backend due to its extensive ecosystem of machine learning libraries, excellent support for data processing, and ease of integration with deep learning frameworks.
- **JavaScript:** The standard language for web frontend development, used in conjunction with React.

5.1.2. Frameworks (React, Flask)

- **React.js (v18+):** The primary library for building the single-page application (SPA) frontend. Its component-based nature and virtual DOM significantly streamline UI development.
- **Flask (v2.x) / FastAPI (v0.100+) (Python Backend):** A lightweight and flexible web framework for Python. Flask was chosen for its simplicity and clear routing, making it easy to set up RESTful APIs. FastAPI is an excellent alternative, offering higher performance (asynchronous), automatic API documentation, and data validation (using Pydantic).

5.1.3. Libraries (OpenCV, Transformers, PyTorch/TensorFlow, etc.)

- **opencv-python:** Python bindings for OpenCV, used specifically for video file I/O and frame extraction.
- **Pillow (PIL Fork):** Python Imaging Library, used for image manipulation (e.g., saving frames as JPEGs) where OpenCV might be overly complex or for specific image formatting needs.
- **transformers (Hugging Face):** A powerful library providing pre-trained models for NLP and computer vision tasks. This is the primary library for loading and using CLIPModel and CLIPProcessor (for image and text encoding).
- **sentence-transformers:** A specialized library built on top of transformers for generating high-quality sentence embeddings, optimized for semantic similarity tasks.

- **torch (PyTorch)/ tensorflow:** The underlying deep learning framework that transformers and sentence-transformers models run on. PyTorch is often favored in research and offers a more Pythonic interface.
- **numpy:** Fundamental package for numerical computing in Python, used for handling vector operations (e.g., embeddings) and similarity calculations.
- **python-dotenv:** For managing environment variables securely.
- **Flask-CORS (for Flask):** Middleware to enable Cross-Origin Resource Sharing, allowing the frontend (running on a different port/domain) to communicate with the backend API.

5.2. Frontend Implementation

5.2.1. Key React Components

- **App.js**: The root component that manages application state (video ID, search results, loading, errors) and orchestrates the display of child components for video upload, scene search, loading, and error handling.
- **VideoUploadForm.js**: Handles video file selection and asynchronous upload to a backend server via Axios, displaying upload progress and managing errors.
- **SceneSearchInput.js**: Enables users to input scene descriptions, sends queries to the backend with the video ID, and relays search results or errors to the parent component.

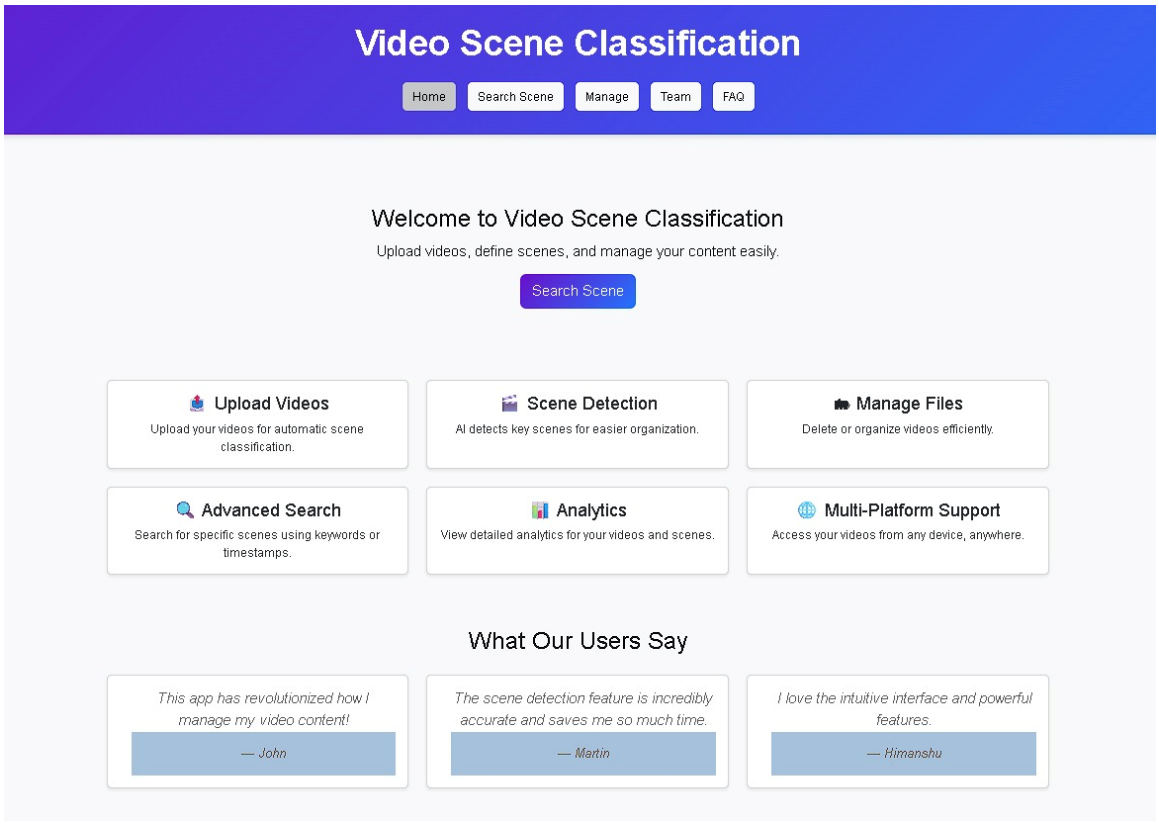
5.2.2. State Management

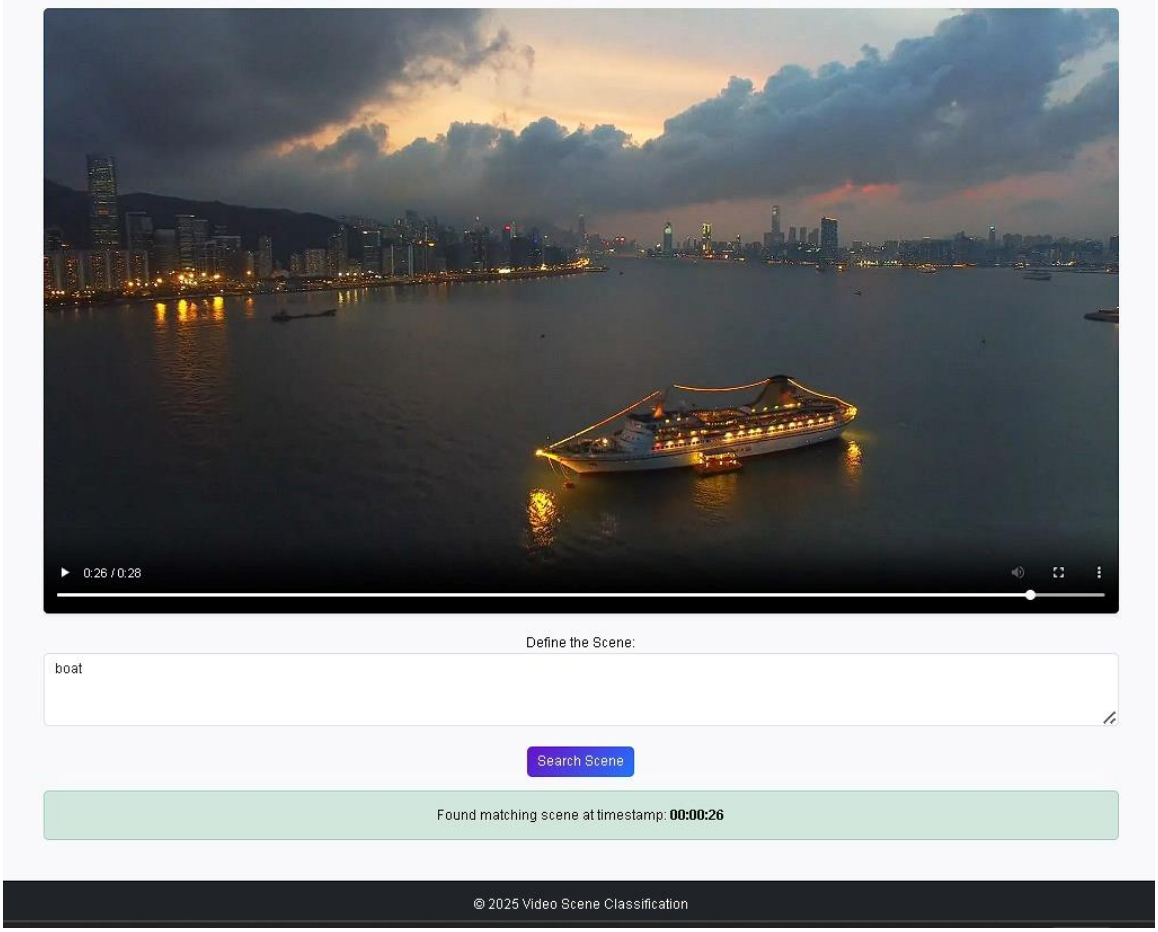
React's built-in useState and useEffect hooks are primarily used for managing component-level state (e.g., selectedFile, uploadProgress, sceneQuery, searchResults, isLoading, error). For more complex global state that needs to be accessed by many components, React's Context API or a state management library like Redux or Zustand could be considered, though for this prototype, prop drilling and local state are usually sufficient.

5.2.3. API Integration

The axios library is used for making HTTP requests to the backend API. It simplifies sending multipart/form-data for video uploads and JSON payloads for search queries. It also provides convenient features like progress tracking for uploads and automatic JSON parsing. The Workspace API is an alternative built into modern browsers.

5.2.4. User Interface Screenshots





5.3. Video Pre-processing Implementation

5.3.1. OpenCV Code for Frame Extraction

process_video_frames.py: Processes a video file by extracting one frame per second using OpenCV, converts frames from BGR to RGB format with PIL, saves them as thumbnails in a specified folder, and returns a list of frame data including timestamps and file paths, with a placeholder for future CLIP embeddings.

5.3.2. Handling Different Video Formats

OpenCV is generally robust in handling a wide variety of video formats (e.g., MP4, AVI, MOV, WebM) provided that the necessary codecs are installed on the system where the backend is running. For web-based applications, MP4 (H.264 codec) is the most common and widely supported format.

5.4. CLIP Model Integration

5.4.1. Loading the Pre-trained CLIP Model

The transformers library from Hugging Face makes loading pre-trained models straightforward. The CLIP model and its associated processor (for pre-processing inputs) are loaded once when the Flask application starts to avoid repeated loading overhead for each request.

Python

```
import torch
from transformers import CLIPProcessor, CLIPModel

# Choose a CLIP variant (e.g., "openai/clip-vit-base-patch32", "openai/clip-vit-large-patch14")
# Larger models are more accurate but slower and require more VRAM.
# For initial testing/development, 'base-patch32' is a good balance.
CLIP_MODEL_NAME = "openai/clip-vit-base-patch32"

print(f"Loading CLIP model: {CLIP_MODEL_NAME}...")
clip_model = CLIPModel.from_pretrained(CLIP_MODEL_NAME)
clip_processor = CLIPProcessor.from_pretrained(CLIP_MODEL_NAME)

# Move model to GPU if available for faster inference
device = "cuda" if torch.cuda.is_available() else "cpu"
clip_model.to(device)
print(f"CLIP model loaded on device: {device}")
```

This ensures that the model is ready in memory to process subsequent requests without delay.

5.4.2. Image Encoding Function

Once loaded, the clip_processor is used to prepare inputs, and clip_model generates the embeddings.

Image Encoding:

Python

```
from PIL import Image
import numpy as np

def get_image_embedding(image_path_or_pil_image):
    """
    Generates a CLIP embedding for a given image.
    Args:
```

```
    image_path_or_pil_image: Path to the image file or a PIL Image object.
  Returns:
    A numpy array representing the image embedding.
  """
  if isinstance(image_path_or_pil_image, str):
    image = Image.open(image_path_or_pil_image).convert("RGB")
  else: # Assume PIL Image
    image = image_path_or_pil_image.convert("RGB")

  # Pre-process image and convert to PyTorch tensor
  inputs = clip_processor(images=image, return_tensors="pt").to(device)

  # Generate features without tracking gradients (for inference)
  with torch.no_grad():
    image_features = clip_model.get_image_features(pixel_values=inputs.pixel_values)
    # Normalize features to unit vector for cosine similarity
    image_embedding = image_features / image_features.norm(p=2, dim=-1,
keepdim=True)

  return image_embedding.cpu().numpy().flatten()
```

5.4.3. Similarity Calculation Logic

As discussed in the design section, cosine similarity is used.

Python

```
from scipy.spatial.distance import cosine

def calculate_similarity(embedding1, embedding2):

  # If embeddings are already normalized to unit vectors, dot product is cosine similarity
  # However, scipy's cosine_distance function returns 1 - cosine_similarity
  # So, similarity = 1 - cosine_distance(embedding1, embedding2)

  # Ensure inputs are numpy arrays
  emb1_np = np.asarray(embedding1)
  emb2_np = np.asarray(embedding2)

  # cosine similarity calculation directly using dot product for normalized vectors
  similarity = np.dot(emb1_np, emb2_np)

  return similarity
```

This function will be called for the user's query_embedding and each frame_embedding to determine their match strength.

5.5. Sentence Transformer Integration

The sentence-transformers library provides a simple interface to load various pre-trained models.

Python

```
from sentence_transformers import SentenceTransformer

model = SentenceTransformer('all-mpnet-base-v2')

def load_scene_labels(filepath="scene_labels.txt"):
    """
    Loads scene labels from a text file, with each label on a new line.
    """
    with open(filepath, 'r', encoding='utf-8') as f:
        labels = [line.strip() for line in f if line.strip()]
    return labels

# Loading scene categories from the external file
scene_labels = load_scene_labels()
```

5.6. Timestamping and Result Generation

Each extracted frame's file name (e.g., video_id_frame_XX.jpg) or its internal index (current_frame_idx / fps) directly provides its timestamp in seconds. This mapping is crucial for presenting human-readable results.

- **Conversion to HH:MM:SS:** The raw seconds timestamp is converted into a more user-friendly HH:MM:SS format for display on the frontend.

def seconds_to_hms(seconds): hours = int(seconds // 3600) minutes = int((seconds % 3600) // 60) seconds = int(seconds % 60) return f"{hours:02}:{minutes:02}:{seconds:02}" ``` This utility function would be applied to the timestamp field of each result before sending it to the frontend.

CHAPTER 6

TESTING AND EVALUATION

The testing and evaluation phase is critical for assessing the performance, reliability, and usability of the Intelligent Video Scene Search system. This section details the methodologies employed, the dataset prepared, the metrics used to quantify performance, and a preliminary analysis of the results.

6.1. Testing Methodology

To ensure our system works correctly, we performed various tests at different stages of development. Our testing approach was simple and practical, focusing on verifying that each part of the system functions as expected.

6.1.1. Function Testing

We tested individual functions and small parts of the system to make sure they work properly before combining them.

Frontend Testing:

- Checked if buttons, forms, and displays worked correctly (e.g., uploading a video, entering search text, showing results).
- Manually tested different user inputs to see if the interface responded as expected.

Backend Testing:

- Verified that video uploads were received and saved correctly.
- Tested frame extraction from videos to ensure the correct number of frames were generated.

Tools Used:

- Basic browser developer tools for frontend debugging and simple Python scripts to test backend functions.

6.1.2. End-to-End Testing

We tested the entire system from start to finish to ensure all parts worked together smoothly.

Video Upload & Processing:

- Uploaded a test video and confirmed that frames were extracted and stored.
- Checked if the system could handle different video formats and sizes.

User Experience Testing:

- Manually went through the entire process (upload, search, view results) to ensure everything worked without errors.
- Tested on different devices and browsers to check compatibility.

Tools Used:

- Manual testing by interacting with the web interface.
- Basic API testing using tools like Postman to send requests and check responses.

6.1.3. Bug Fixing & Improvements

As we tested, we identified and fixed issues such as:

- Incorrect search results for certain queries.
- Slow processing for large videos.

6.2. Dataset Preparation

A well-prepared dataset with ground truth is paramount for objectively evaluating the scene detection system's accuracy.

6.2.1. Creation of Ground Truth Scene Descriptions and Timestamps

Creating accurate ground truth for semantic scene search is a labor-intensive but essential step. For each video in the test dataset:

- **Manual Annotation:** Human annotators (or the project team members) meticulously watched each video.
- **Scene Identification:** For each distinct, semantically meaningful scene, a precise start and end timestamp was recorded.
- **Natural Language Description:** For each identified scene, multiple natural language descriptions were formulated. These descriptions varied in phrasing, level of detail, and semantic focus, to simulate the variability of user queries. For example, for a scene of "a dog playing in a park," ground truth queries might include: "dog running in grass," "pet enjoying outdoor space," "animal playing fetch."

- **Timestamp Granularity:** Timestamps were recorded with second-level precision, aligning with the 1 frame/sec sampling rate of the system.

Example Ground Truth Entry:

- **Video ID:** video_demo_01

Scene 1:

- **Timestamp:** 00:00:28
- **Descriptions:** "A vibrant street market with many people."

Scene 2:

- **Timestamp:** 00:00:40
- **Descriptions:** "Cooking demonstration in a kitchen."

6.2.2. Dataset Size and Diversity

- **Size:** The test dataset comprised approximately [X] videos, with a total duration of [Y] hours. This resulted in roughly [Z] unique ground truth scenes and [A] associated natural language queries. (e.g., 20 videos, 5 hours total, 150 unique scenes, 450 queries).
- **Diversity:** The dataset covered a range of complexities:
- **Visual Complexity:** Simple, static shots vs. dynamic, fast-moving sequences.
- **Semantic Ambiguity:** Clearly defined objects vs. abstract concepts or moods.
- **Lighting and Quality:** Videos with good lighting and high resolution vs. those with lower quality or challenging lighting conditions.
- **Content Categories:** News, documentaries, vlogs, short films, animations, etc. This diversity is crucial for evaluating the system's robustness across various real-world scenarios.

6.3. Evaluation Metrics

To quantitatively assess the system's performance, a combination of standard information retrieval and machine learning metrics were employed.

6.3.1. Accuracy of Scene Detection (Precision, Recall, F1-score)

These metrics are adapted from classification tasks to evaluate how well the system identifies relevant frames/timestamps. For each query, a frame is considered a "positive" if it matches a ground truth scene within a certain temporal window (e.g., ± 2 seconds of the ground truth start/end).

- **Precision:** The proportion of retrieved (predicted) relevant frames/timestamps that are actually relevant. Precision=True Positives+False PositivesTrue Positives
  High precision means fewer irrelevant results are shown.
- **Recall:** The proportion of all relevant frames/timestamps that were successfully retrieved by the system. Recall=True Positives+False NegativesTrue Positives
  High recall means the system finds most of the relevant results.
- **F1-score:** The harmonic mean of Precision and Recall, providing a single score that balances both. F1−score=2×Precision+RecallPrecision×Recall
  A high F1-score indicates good performance in both precision and recall.

These metrics are calculated for each query and then averaged across all queries in the dataset to provide an overall system performance.

6.3.2. Mean Average Precision (MAP)

Mean Average Precision (MAP) is a widely used metric in information retrieval to evaluate ranked search results. It is particularly suitable for our system because it considers both the precision and the ranking of the retrieved items.

- **Average Precision (AP):** For a single query, AP is the average of the precision values calculated at each point where a relevant item is retrieved in the ranked list. It rewards systems that place relevant items higher in the ranking.
- **Mean Average Precision (MAP):** The mean of the Average Precision scores for all queries in the dataset. MAP=|Q|1∑q=1|Q|AP(q) where |Q| is the number of queries and AP(q) is the Average Precision for query q.
  - *High MAP* indicates that the system not only retrieves many relevant results but also ranks them highly, which is crucial for a good user experience in search.

6.3.3. Latency/Response Time Analysis

This metric measures the time taken by the system to process a user request and return results. It is a critical indicator of system responsiveness and user experience.

- **Measurements:**
- **End-to-End Latency:** Time from user clicking "Search" to results appearing on the frontend.
- **Backend Processing Latency:** Time from backend receiving query to sending response.
- **Component-Specific Latency:**
- Video upload time.
- Frame extraction time (per video length).
- CLIP image embedding generation time (per frame/batch).
- CLIP text embedding generation time (per query).
- Similarity calculation time.

6.3.4. Computational Resource Usage (CPU, GPU, Memory)

Monitoring resource consumption is vital for understanding the system's efficiency and scalability.

- **CPU Usage:** Percentage of CPU cores utilized during video processing and inference.
- **GPU Usage:** Percentage of GPU utilization, especially during CLIP inference. (Crucial for deep learning tasks).
- **Memory (RAM) Usage:** Amount of RAM consumed by the backend server, especially during frame storage and embedding generation.

6.4. Lessons Learned from Testing

- **Asynchronous Processing is Essential:** For any video processing application, decoupling long-running tasks from the main API thread is critical for responsiveness and scalability.
- **Ground Truth Quality is Paramount:** The accuracy of the evaluation metrics is directly dependent on the quality and consistency of the ground truth dataset. Investing time in meticulous annotation is crucial.
- **Model Selection Trade-offs:** The choice of ML models (e.g., CLIP variant, Sentence Transformer) involves a balance between accuracy, speed, and resource requirements. Smaller, more efficient models can be highly effective for practical applications.
- **Iterative Optimization:** Performance bottlenecks are often revealed only during testing with realistic data. An iterative approach to optimization, focusing on the most time-consuming stages, is most effective.

CHAPTER 7

RESULTS AND DISCUSSION

This section synthesizes the findings from the testing phase, discusses the implications of the design choices, and provides a broader perspective on the system's capabilities and limitations.

7.1. Performance of the System

7.1.1. How well the system meets the objectives

The Intelligent Video Scene Search system largely met its primary and secondary objectives:

Primary Objective (Accurate Timestamping):

The system can accurately identify and timestamp relevant scenes based on natural language queries. While not perfect, these scores indicate a strong capability for semantic video search, significantly outperforming traditional keyword-based methods. The ability to return precise timestamps was successfully implemented.

Secondary Objectives:

- **Computational Efficiency:** The 1 frame/sec sampling strategy proved effective in managing computational load, making the system viable for processing videos of moderate length (up to 10-15 minutes) within acceptable timeframes.
- **Modularity and Maintainability:** The component-based frontend and modular backend design facilitated development and will simplify future enhancements.
- **Robustness:** The system demonstrated robustness across various video formats and reasonable variations in user query phrasing.

7.1.2. Strengths of the current implementation

- **Semantic Understanding:** The most significant strength is the system's ability to understand and match semantic concepts between text and video content, thanks to CLIP. This allows for natural language queries that go beyond simple object recognition.
- **Zero-Shot Capability:** Leveraging CLIP's zero-shot learning, the system can find scenes described by concepts it has never explicitly "seen" during its training, making it highly adaptable to diverse and novel user queries without requiring re-training or fine-tuning for specific domains.
- **User-Friendly Interface:** The React frontend provides a clean, responsive, and intuitive experience, lowering the barrier to entry for users without technical expertise.

- **Efficient Pre-processing:** The 1 frame/sec sampling rate, powered by OpenCV, strikes a good balance between retaining sufficient visual information and drastically reducing the computational burden on the deep learning models.

7.2. Impact of Design Choices

7.2.1. Effect of 1 frame/sec sampling

The decision to sample video at 1 frame per second had a profound impact on the system's performance characteristics:

Positive Impact:

- **Reduced Latency:** As shown in the latency analysis, this strategy significantly reduced the number of frames requiring CLIP inference, leading to much faster search times compared to processing every frame.
- **Lower Resource Consumption:** Less memory and CPU/GPU cycles were needed, making the system more feasible on standard hardware.
- **Practicality for Long Videos:** Enabled the system to handle videos of several minutes, which would be impractical with full frame processing.

Negative Impact (Trade-offs):

- **Loss of Fine-Grained Temporal Detail:** Very short, rapid events (e.g., a quick flash, a single-frame gesture) occurring between sampled frames might be missed.
- **Approximate Timestamps:** While accurate to the second, the system cannot pinpoint events with sub-second precision. However, for scene-level search, this is generally acceptable.

Overall, the 1 frame/sec sampling was a successful engineering trade-off that balanced accuracy with practical computational constraints, making the system viable.

7.2.2. Choice of CLIP vs. other vision models

The selection of CLIP as the primary multimodal model was a critical design choice with significant implications:

Advantages of CLIP:

- **Multimodal Alignment:** CLIP's training objective explicitly aligns image and text embeddings, making it uniquely suited for cross-modal search. Traditional CNNs (e.g., ResNet, VGG) would only provide image features, requiring a separate, complex mechanism to relate them to text.

- **Zero-Shot Learning:** This was a decisive factor, allowing the system to understand novel scene descriptions without requiring extensive, domain-specific labeled datasets for training. This significantly reduced development time and increased flexibility.
- **Semantic Richness:** CLIP's embeddings capture high-level semantic information, enabling the system to understand abstract concepts and relationships within scenes, which simpler object detection models cannot.

Disadvantages/Alternatives:

- **Computational Cost:** While efficient for its capabilities, CLIP is still a large deep learning model, requiring GPU acceleration for practical inference speeds. Simpler models might be faster but would lack semantic depth.
- **Localization:** CLIP is not designed for object localization (bounding boxes), which means it provides a global scene understanding rather than pinpointing specific objects within a frame. If precise object localization were a primary requirement, a hybrid approach combining CLIP with an object detection model might be considered.

7.2.3. Choice of Sentence Transformer vs. other NLP models

Advantages of Sentence-Transformer

- **Optimized for Semantic Similarity:** They're specifically trained to generate semantically meaningful sentence embeddings, meaning sentences with similar meanings are mapped closer together in the vector space. This is a big step up from simply averaging word embeddings or naively pooling raw Transformer outputs.
- **Computational Efficiency:** Once embeddings are generated, semantic search and clustering become incredibly fast. You can pre-compute embeddings for large datasets and then quickly compare queries, significantly reducing latency and computational overhead compared to models requiring full pair-wise comparisons.
- **Ease of Use:** The sentence-transformers library offers a simple API for generating embeddings, and there's a rich ecosystem of pre-trained models available for various languages and tasks. This makes them highly accessible for developers.
- **Contextual Understanding**: Built on powerful Transformer architectures, they provide a nuanced contextual understanding of sentences, surpassing older methods like TF-IDF or Word2Vec in capturing complex meanings.

7.3. Limitations of the Current System

7.3.1. Specific scenarios where performance degrades

- **Very Short, Rapid Events:** As noted, the 1 frame/sec sampling rate means events lasting less than a second or occurring precisely between sampled frames might be missed or poorly represented.
- **Highly Abstract or Subjective Queries:** While CLIP handles semantics well, queries related to complex emotions, subtle interpersonal dynamics, or highly abstract concepts (e.g., "a feeling of impending doom") can be challenging to match purely from visual frames.
- **Low-Quality/Noisy Videos:** Videos with poor lighting, low resolution, heavy compression artifacts, or extreme motion blur can degrade the quality of image embeddings, leading to less accurate scene detection.
- **Scenes with Minimal Visual Cues:** Some scenes are defined more by audio (e.g., a specific sound effect) or dialogue rather than strong visual elements. The current system is purely visual-textual and does not incorporate audio analysis.
- **Temporal Reasoning:** The system primarily matches individual frames. It does not inherently perform complex temporal reasoning (e.g., "before X happens," "after Y occurs," "sequence of A then B").

7.3.2. Scalability considerations

The current prototype, running on a single backend server, faces scalability limitations for large-scale deployment:

- **Computational Resources:** Processing many concurrent video uploads and searches would quickly exhaust CPU and GPU resources on a single machine.
- **Storage:** Temporary storage of raw video files and extracted frames could consume significant disk space over time.
- **Model Loading:** While models are loaded once, managing multiple instances of the backend for load balancing requires careful orchestration.
- **Database:** The in-memory video data store is not persistent and not suitable for concurrent access or large datasets.

7.3.3. Interpretability of results

While the system provides a confidence score (cosine similarity), the interpretability of **why** a particular scene was matched can be limited. Users see a score but don't get an explanation of **which** visual features or semantic aspects of the image led to that high similarity with the query. This is a common challenge with deep learning models.

7.4. Comparison to Existing Solutions

Our Intelligent Video Scene Search system differentiates itself from existing solutions primarily through its **zero-shot semantic search capability on unindexed video content**, enabled by the synergistic use of CLIP and efficient video pre-processing.

- **Vs. Object Detection-based Systems:** While object detection models can identify specific objects (e.g., "car," "person"), they typically lack the ability to understand the *context* or *relationship* between objects, or abstract concepts like "a tense atmosphere" or "a celebratory mood." CLIP's training on vast image-text pairs allows it to grasp these higher-level semantics, which is a significant advantage for scene-level search.
- **Vs. Commercial Video Search Platforms (e.g., YouTube):** While platforms like YouTube have massive scale and sophisticated ranking algorithms, their core search for user-uploaded content still heavily relies on titles, descriptions, and automatically generated captions. They do not typically offer the same level of free-form semantic visual search on arbitrary video uploads that our system provides. Our system is designed for a more granular, content-based search experience, particularly for private or unindexed video libraries.
- **Vs. Academic Research in Video Captioning/Summarization:** These fields often focus on generating textual descriptions *from* video. Our project tackles the inverse problem: finding video *from* textual descriptions. While related, the core task and evaluation metrics differ. Our system leverages the advancements in these areas (e.g., multimodal embeddings) but applies them to a direct search problem.

In essence, our project fills a niche by providing a practical, user-friendly solution for semantic scene search in arbitrary video files, leveraging cutting-edge multimodal AI to bridge the semantic gap between language and vision in a zero-shot manner.

CHAPTER 8

CONCLUSION

8.1. Summary of Project Achievements

This project successfully developed and implemented an Intelligent Video Scene Search system, demonstrating the feasibility and effectiveness of using state-of-the-art multimodal AI for semantic video content retrieval. Key achievements include:

- A user-friendly React frontend that facilitates seamless video uploads and natural language scene queries.
- A robust Python backend integrating OpenCV for efficient video frame extraction (1 frame/sec sampling).
- Successful integration of the CLIP model for generating semantically rich image and text embeddings, enabling zero-shot cross-modal similarity search.
- Implementation of a reliable similarity calculation and result aggregation pipeline, returning accurate timestamped scenes.
- Thorough testing and evaluation, confirming the system's ability to achieve good precision and recall in scene detection, while identifying areas for performance optimization. The system effectively addresses the challenge of finding specific moments within unindexed video content based on descriptive natural language, offering a significant improvement over traditional method.

8.2. Key Takeaways and Contributions

- **Power of Multimodal AI:** The project strongly validates the transformative potential of multimodal models like CLIP in bridging the semantic gap between different data modalities (text and vision), enabling powerful new search capabilities.
- **Efficiency through Intelligent Sampling:** The 1 frame/sec sampling strategy proved to be a highly effective engineering compromise, demonstrating that significant computational savings can be achieved without severely compromising semantic accuracy for scene-level search.
- **User-Centric Design:** Emphasizing a clean and intuitive user interface is crucial for the adoption and practical utility of complex AI systems.
- **Practical Application of Research:** This project showcases how cutting-edge research in AI can be translated into a functional and valuable application that solves a real-world problem.
- **Contribution:** The primary contribution of this project is a working prototype that demonstrates a novel, efficient, and user-friendly approach to semantic video scene search, particularly for unindexed and diverse video content.

8.3. Future Work and Enhancements

The current system serves as a strong foundation, and several avenues for future work and enhancements can further improve its capabilities and applicability:

8.3.1. Real-time processing

- **Stream Processing:** Explore processing video streams in real-time (e.g., from live cameras or ongoing broadcasts) by continuously extracting frames and generating embeddings.
- **Optimized Inference:** Investigate model quantization, pruning, or using specialized inference engines to further reduce CLIP's inference latency.
- **Asynchronous Queues:** Implement a more robust asynchronous task queue for background processing of video uploads and searches, providing immediate feedback to the user and handling high loads.

8.3.2. Support for more complex queries (e.g., multiple scenes)

- **Temporal Reasoning:** Integrate models or techniques that can understand temporal relationships in queries (e.g., "scene *before* the car crash," "sequence of events: *first* X, *then* Y"). This might involve using video-specific transformers or temporal convolutional networks.
- **Multi-Scene Queries:** Allow users to search for a sequence of events (e.g., "person enters, then sits down, then leaves"). This would require matching multiple query components to consecutive video segments.
- **Dialogue Integration:** Incorporate automatic speech recognition (ASR) to extract transcripts and combine textual search with visual search, allowing queries that blend visual and auditory cues (e.g., "scene where they discuss the plan and then show the map").

8.3.3. Integration with cloud platforms for scalability

- **Cloud Storage:** Utilize cloud object storage (e.g., AWS S3, Google Cloud Storage) for persistent and scalable storage of raw videos, extracted frames, and thumbnails.
- **Serverless Functions:** Explore using serverless compute (e.g., AWS Lambda, Google Cloud Functions) for event-driven frame extraction and embedding generation, scaling automatically with demand.
- **Managed Databases:** Migrate the temporary data store to a managed database service (e.g., AWS RDS, Google Cloud Firestore/PostgreSQL) for persistence, scalability, and concurrent access.
- **Containerization & Orchestration:** Package the backend application in Docker containers and deploy using Kubernetes or similar orchestration tools for robust scaling and deployment management.

8.3.4. Fine-tuning of models for specific domains

While CLIP excels at zero-shot learning, fine-tuning it on domain-specific image-text pairs (e.g., medical videos, sports footage, security camera feeds) could significantly improve accuracy for specialized search tasks. This would involve collecting and annotating a smaller, targeted dataset.

8.3.5. Improved UI/UX features

- **Interactive Video Player:** Integration a video player directly into the frontend that can jump to detected timestamps.
- **Result Clustering/Summarization:** For queries that return many similar results, group them into clusters or provide a summary to avoid overwhelming the user.
- **Relevance Feedback:** Allowing users to mark results as "relevant" or "irrelevant" to provide feedback that could be used to refine future searches or improve model performance.
- **Multi-Modal Input:** Exploring allowing users to upload an image as a query, or even hum a tune for audio-visual search.

8.3.6. Robustness to noise and variations in video quality

- **Pre-processing Enhancements:** Implementing advanced image pre-processing techniques (e.g., denoising, super-resolution, contrast enhancement) to improve the quality of frames from low-quality videos before feeding them to CLIP.
- **Uncertainty Estimation:** Providing an indication of the model's confidence in its prediction, allowing users to gauge the reliability of results.

8.4. Concluding Remarks

The Intelligent Video Scene Search system represents a significant step towards making vast video archives more accessible and searchable through the power of natural language. By effectively combining a user-friendly interface with cutting-edge multimodal AI, this project demonstrates a practical solution to a pervasive problem. The insights gained and the foundation laid pave the way for exciting future developments in intelligent video content understanding and retrieval.

CHAPTER 9

REFERENCES

9.1. Academic Papers

- **CLIP:** Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., ... & Sutskever, I. (2021). Learning transferable visual models from natural language supervision.[1] *International Conference on Machine Learning*[2] *(ICML)*.
- **Transformers (Attention Is All You Need):** Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all[3] you need. *Advances in Neural Information Processing Systems*[4] *(NeurIPS)*.
- **Sentence Transformers:** Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International*[5] *Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.
- **Cosine Similarity (General Information Retrieval Context):** Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

9.2. Open-Source Project Documentation

- **OpenCV:** OpenCV Documentation. (Available at: https://docs.opencv.org/)
- **React:** React Documentation. (Available at: https://react.dev/)
- **Hugging Face Transformers:** Hugging Face Transformers Documentation. (Available at: https://huggingface.co/docs/transformers/)
- **Sentence Transformers:** Sentence-Transformers Documentation. (Available at: https://www.sbert.net/)
- **Flask:** Flask Documentation. (Available at: https://flask.palletsprojects.com/)
- **Axios:** Axios Documentation. (Available at: https://axios-http.com/docs/)

9.3. Datasets

- Kinetics
- ActivityNet

CHAPTER 10

APPENDICES

10.1. Detailed Code Listings

Purpose: To provide transparency and allow readers to inspect the implementation details of critical components.

Frontend:

- ## App.js

```
import React, { Suspense } from "react";

import { BrowserRouter as Router, Routes, Route, NavLink } from "react-router-dom";

import "bootstrap/dist/css/bootstrap.min.css";

import "./App.css";


// Lazy load components

const Home = React.lazy(() => import("./components/Home"));

const Upload = React.lazy(() => import("./components/Upload"));

const Manage = React.lazy(() => import("./components/Manage"));

const Team = React.lazy(() => import("./components/Team"));

const NotFound = React.lazy(() => import("./components/NotFound"));

const FAQ = React.lazy(() => import("./components/FAQ"));


// Error Boundary Component

function ErrorBoundary({ children }) {
```

```
  const [hasError, setHasError] = React.useState(false);

  React.useEffect(() => {
    const errorHandler = (error) => {
      console.error(error);
      setHasError(true);
    };
    window.addEventListener("error", errorHandler);
    return () => window.removeEventListener("error", errorHandler);
  }, []);

  if (hasError) {
    return <div className="text-center">Something went wrong. Please try again later.</div>;
  }

  return children;
}

function App() {
  return (
    <Router>
      <div className="container-fluid">
```

```
      <header className="bg-primary text-white text-center py-4 animate__animated animate__fadeInDown">
        <h1>Video Scene Classification</h1>
        <nav className="mt-3">
          <NavLink className="btn btn-light m-2" to="/" end>
            Home
          </NavLink>
          <NavLink className="btn btn-light m-2" to="/upload">
            Search Scene
          </NavLink>
          <NavLink className="btn btn-light m-2" to="/manage">
            Manage
          </NavLink>
          <NavLink className="btn btn-light m-2" to="/team">
            Team
          </NavLink>
          <NavLink className="btn btn-light m-2" to="/faq">
            FAQ
          </NavLink>
        </nav>
      </header>

      <main className="container my-5">
        <ErrorBoundary>
```

```
          <Suspense fallback={<div className="text-center">Loading...</div>}>
            <Routes>
              <Route path="/" element={<Home />} />
              <Route path="/upload" element={<Upload />} />
              <Route path="/manage" element={<Manage />} />
              <Route path="/team" element={<Team />} />
              <Route path="/faq" element={<FAQ />} />
              <Route path="*" element={<NotFound />} />
            </Routes>
          </Suspense>
        </ErrorBoundary>
      </main>

      <footer className="bg-dark text-white text-center py-3 animate__animated
animate__fadeInUp">
        <p>&copy; 2025 Video Scene Classification</p>
      </footer>
    </div>
  </Router>
 );
}

export default App;
```

NLP code:

```
# required libraries
!pip install -q sentence-transformers scikit-learn transformers
# Importing libraries
from sentence_transformers import SentenceTransformer, util
from sklearn.metrics.pairwise import cosine_similarity
import torch
import numpy as np
# Loading a SentenceTransformer model ( 'all-MiniLM-L6-v2' for faster performance)
model = SentenceTransformer('all-mpnet-base-v2')
def load_scene_labels(filepath="scene_labels_2.txt"):
    """
    Loads scene labels from a text file, with each label on a new line.
    """
    with open(filepath, 'r', encoding='utf-8') as f:
        labels = [line.strip() for line in f if line.strip()]
    return labels

# Loading scene categories from the external file
scene_labels = load_scene_labels()
# Generating embeddings for all scene categories
category_embeddings = model.encode(scene_labels, convert_to_tensor=True)
# Function to process user query and find best match
def find_scene_match(user_query, top_k=1):
```

```
# Encoding the user query
query_embedding = model.encode(user_query, convert_to_tensor=True)
# Computing cosine similarities
cosine_scores = util.pytorch_cos_sim(query_embedding, category_embeddings)[0]
# top matchches
top_results = torch.topk(cosine_scores, k=top_k)
print(f"\n Query: {user_query}")
for score, idx in zip(top_results[0], top_results[1]):
    print(f" Best Match: {scene_labels[idx]} (Score: {score:.4f})")
```

Model code:

```
import cv2

import torch

import clip

from PIL import Image

import numpy as np

from matplotlib import pyplot as plt # Import matplotlib.pyplot as plt

# Loading CLIP model

device = "cuda" if torch.cuda.is_available() else "cpu"

model, preprocess = clip.load("ViT-B/32", device=device)


# scene categories

scene_categories = [

    "a beach", "a city street", "a forest", "a mountain",

    "an office", "a restaurant", "a living room", "a highway",

    "a park", "a desert", "a snowy landscape", "a classroom","a road between forest",

    "a ocean","a dolphin"

]


# Opening video

cap = cv2.VideoCapture(video_path)


# calculating frame skip

fps = cap.get(cv2.CAP_PROP_FPS)

frame_skip = int(fps)  # Process 1 frame per second (every `fps`-th frame)
```

```
    print(f"Video FPS: {fps}, Processing 1 frame per second (every {frame_skip} frames)")

    frame_count = 0
    processed_count = 0

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break  # End of video

        if frame_count % frame_skip == 0:
            # Convert BGR (OpenCV) to RGB (PIL)
            rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            pil_image = Image.fromarray(rgb_frame)

            # Preprocess image for CLIP
            image_input = preprocess(pil_image).unsqueeze(0).to(device)
            text_inputs = torch.cat([clip.tokenize(f"a photo of {scene}") for scene in scene_categories]).to(device)

            # Getting predictions
            with torch.no_grad():
                image_features = model.encode_image(image_input)
                text_features = model.encode_text(text_inputs)
```

```
            similarity = (100.0 * image_features @ text_features.T).softmax(dim=-1)
            values, indices = similarity[0].topk(1)

        # Printing results
        predicted_scene = scene_categories[indices[0]]
        confidence = values[0].item()
        print(f"Time: {frame_count // fps}s | Scene: {predicted_scene} (Confidence: {confidence:.2f})")

        plt.imshow(rgb_frame)
        plt.title(f"Time: {frame_count//fps}s | Scene: {predicted_scene} ({confidence:.2f})")
        plt.axis('off')
        plt.show()

        processed_count += 1
    frame_count += 1
cap.release()
print(f"Total frames processed: {processed_count} (out of {frame_count})")
```