

CS 516: Information Retrieval and Text Mining

Information Technology University (ITU)
Fall 2025

Course Instructor: Dr. Ahmad Mustafa

Submitted By:

Syed Zain ul Abideen MSCS24047

Video Retrieval System Using Speech Transcription and IR Techniques

Abstract

This thesis presents a Video Retrieval System that allows users to retrieve relevant video segments using textual queries. Videos are converted to audio, transcribed using Whisper ASR, and indexed with TF-IDF. Users can search for keywords, and the system returns timestamped segments for playback, improving accessibility and search efficiency.

1. Motivation & Introduction

Problem Statement:

Educational and instructional videos are increasing rapidly, making manual search for specific content time-consuming. Traditional video search relies on titles or metadata, which is insufficient.

Relevance to IR/TM:

Information Retrieval and Text Mining techniques can automatically process video transcripts to extract meaningful, searchable information, enabling efficient retrieval of video segments relevant to user queries.

2. Related Work & Limitations

Prior Work:

YouTube auto-caption search

Google Speech API, IBM Watson Speech-to-Text

Subtitle-based video search systems in research papers

Limitations:

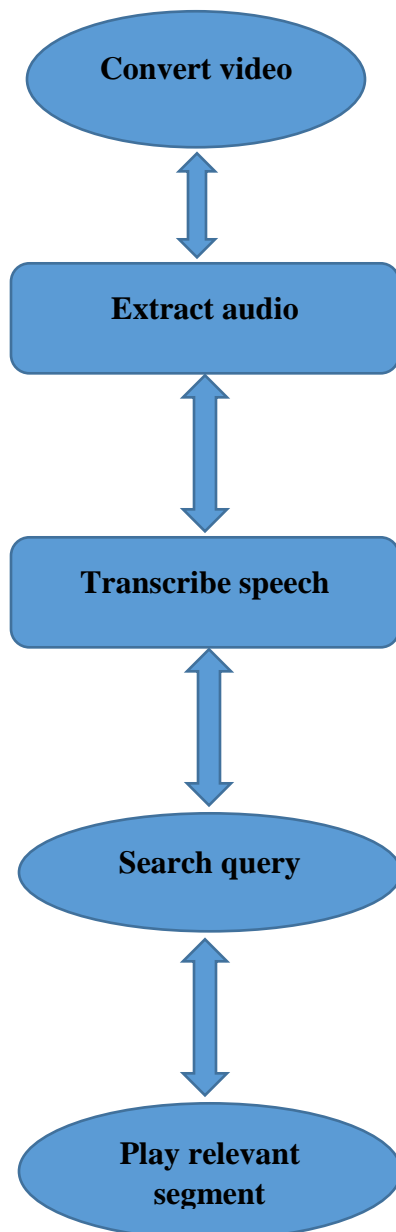
- Many tools are cloud-dependent and cannot work offline
- Lack of time stamp-level retrieval
- Limited control over indexing, search algorithms, and relevance scoring

Motivation:

A system that is offline, locally customizable, and provides segment-level retrieval improves usability for research and education.

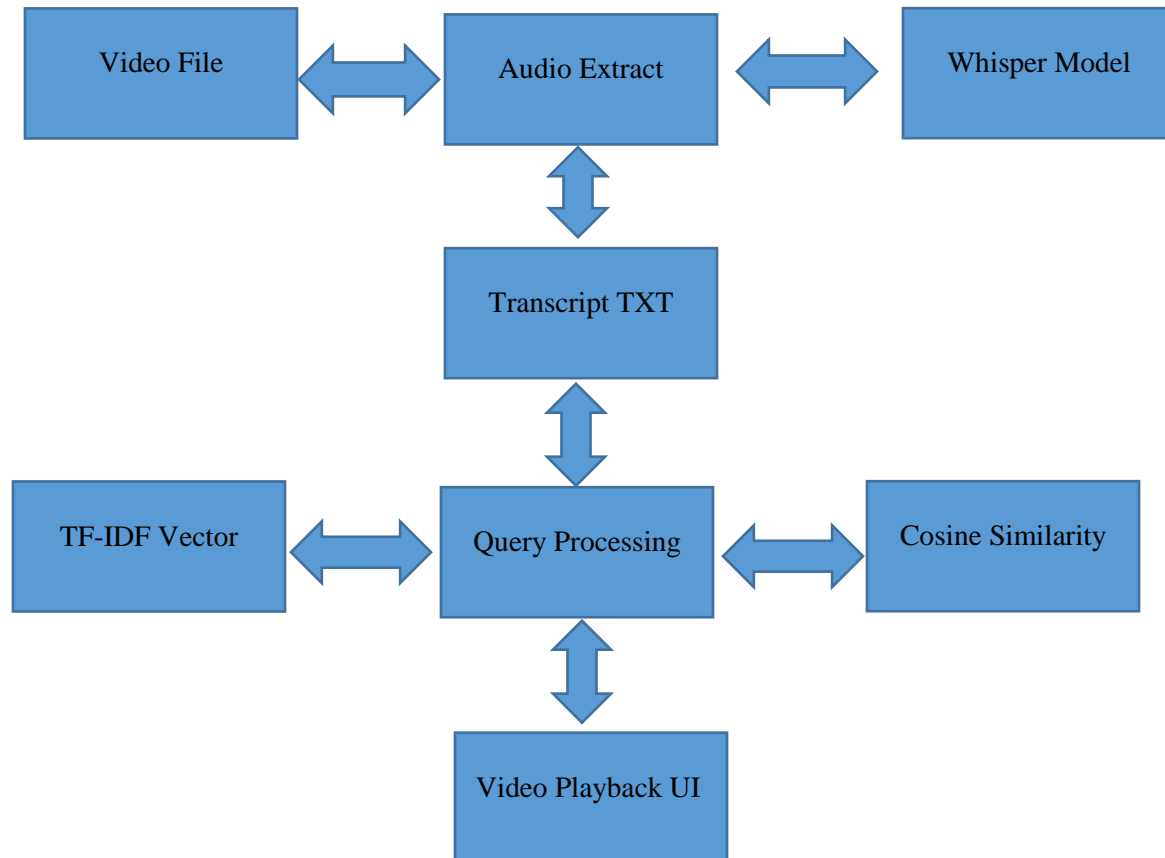
3. Proposed System / Solution

High-Level Concept:



System Diagram:

The following diagram represents the workflow of the video retrieval system.



Block diagram of Information retrieval system

Program Code

jupyter Notebook

<http://localhost:8888/notebooks/Untitled10.ipynb>

```
%pip install python-vlc
```

```
%pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121
```

```
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121
```

```
%pip install opencv-python
```

```
%pip install opencv-python-headless
```

```
# =====
```

```
# VIDEO RETRIEVAL SYSTEM (FULL)
```

```
# =====
```

```
import os
```

```
from pydub import AudioSegment
```

```
import speech_recognition as sr
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn.metrics.pairwise import cosine_similarity
```

```
import pickle
```

```
import nltk
```

```
from nltk.corpus import stopwords
```

```
import whisper
```

```
import cv2
```

```
from pydub import AudioSegment
```

```
try:
```

```
    stopwords.words('english')
```

```
except LookupError:
```

```
    nltk.download('stopwords')
```

```
# Set ffmpeg paths manually
```

```
AudioSegment.ffmpeg = r"D:\FFMPEG\1\bin\ffmpeg.exe"
```

```
AudioSegment.converter = r"D:\FFMPEG\1\bin\ffmpeg.exe"
```

```
AudioSegment.ffprobe = r"D:\FFMPEG\1\bin\ffprobe.exe"
```

```
VIDEO_PATH = r"D:\FFMPEG"
```

```
TRANSCRIPT_PATH = r"D:\FFMPEG"
```

```
os.makedirs(VIDEO_PATH, exist_ok=True)
```

```

os.makedirs(TRANSCRIPT_PATH, exist_ok=True)

def video_to_audio(video_file):
    audio_file = os.path.join(TRANSCRIPT_PATH, os.path.basename(video_file).replace(".mp4",
".wav"))

    AudioSegment.from_file(video_file).export(audio_file, format="wav")

    return audio_file

def audio_to_text(audio_file):
    r = sr.Recognizer()

    with sr.AudioFile(audio_file) as source:
        audio_data = r.record(source)

    try:
        text = r.recognize_google(audio_data)
    except sr.UnknownValueError:
        text = ""
    except sr.RequestError as e:
        print(f"Speech recognition error: {e}")
        text = ""

    return text

# -----
# STEP 1: TRANSCRIBE VIDEOS WITH TIMESTAMPS
# -----

def generate_transcripts():
    if not os.path.exists(TRANSCRIPT_PATH):
        os.makedirs(TRANSCRIPT_PATH)

    model = whisper.load_model("base") # can use "small" / "medium" for better accuracy

    for file in os.listdir(VIDEO_PATH):
        if file.endswith(".mp4"):

```

```

video_file = os.path.join(VIDEO_PATH, file)

print(f"Transcribing: {file} ...")

result = model.transcribe(video_file)

transcript_file = os.path.join(TRANSCRIPT_PATH, file.replace(".mp4", ".txt"))

with open(transcript_file, "w", encoding="utf-8") as f:

    for seg in result["segments"]:

        start = seg["start"]

        end = seg["end"]

        text = seg["text"].strip()

        f.write(f"{start:.2f}-{end:.2f} {text}\n")

print(f"Saved transcript: {transcript_file}")

# -----

# STEP 3: SEARCH QUERY

# -----

def search_video(query, vectorizer, vectors, files, timestamps, documents, top_n=5):

    q_vec = vectorizer.transform([query])

    scores = cosine_similarity(q_vec, vectors)[0]

    results = sorted(

        zip(files, timestamps, scores, documents),

        key=lambda x: x[2],

        reverse=True

   )[:top_n]

    print("\nSearch Results:\n")

    for idx, (f, ts, score, text) in enumerate(results, 1):

        print(f"{idx}. {f} | Timestamp: {ts} | Score: {score:.3f}")

        print(f"Text: {text}\n")

```

```

return results

# -----
# STEP 2: BUILD LINE-LEVEL INDEX
# -----

def build_index():

    documents = []

    timestamps = []

    files = []

    for file in os.listdir(TRANSCRIPT_PATH):

        if file.endswith(".txt"):

            path = os.path.join(TRANSCRIPT_PATH, file)

            with open(path, "r", encoding="utf-8") as f:

                for line in f:

                    if line.strip() == "":

                        continue

                    try:

                        ts, text = line.split(" ", 1)

                        documents.append(text.strip())

                        timestamps.append(ts.strip())

                        files.append(file.replace(".txt", ".mp4"))

                    except ValueError:

                        continue

    vectorizer = TfidfVectorizer(stop_words="english")

    vectors = vectorizer.fit_transform(documents)

    print("Index built successfully!")

    return vectorizer, vectors, files, timestamps, documents

```

```

# -----
# STEP 4: PLAY VIDEO SEGMENT
# -----

def play_video_segment(video_file, start_time, end_time):
    cap = cv2.VideoCapture(video_file)

    if not cap.isOpened():
        print("Error opening video file:", video_file)
        return

    fps = cap.get(cv2.CAP_PROP_FPS)
    start_frame = int(start_time * fps)
    end_frame = int(end_time * fps)
    cap.set(cv2.CAP_PROP_POS_FRAMES, start_frame)

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret or cap.get(cv2.CAP_PROP_POS_FRAMES) > end_frame:
            break

        cv2.imshow('Video Segment', frame)
        if cv2.waitKey(int(1000/fps)) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()
    player = vlc.MediaPlayer(video_file)
    player.play()
    time.sleep(0.5) # wait for player to start

```

```

# Jump to start time (milliseconds)
player.set_time(int(start_time * 1000))

# Play until end_time
duration = end_time - start_time
time.sleep(duration)

player.stop()

# -----
# STEP 5: MAIN FUNCTION TO RUN SYSTEM
# -----
def main():
    print("Generating transcripts...")
    generate_transcripts()

    print("\nBuilding index...")
    vectorizer, vectors, files, timestamps, documents = build_index()

    while True:
        query = input("\nEnter your search query (or 'exit' to quit): ")
        if query.lower() == "exit":
            break

    results = search_video(query, vectorizer, vectors, files, timestamps, documents)

    # Play the top result automatically
    if results:
        top_file, top_ts, top_score, top_text = results[0]
        start, end = map(float, top_ts.split("-"))

```

```
print(f"\nPlaying top video segment: {top_file} ({start}-{end} sec)")
play_video_segment(os.path.join(VIDEO_PATH, top_file), start, end)

# -----
# RUN
# -----
if __name__ == "__main__":
    main()
play_video_segment("D:\FFMPEG\a.mp4", 10, 15)
```

Components and Modules:

Module	Function	Tool/Technology
Video Input	Load videos from dataset	MP4 Files
Audio Extraction	Convert video to audio	FFmpeg, Pydub
Speech Transcription	Convert audio to text	Whisper ASR
Indexing	Generate TF-IDF vectors	Scikit-Learn
Query Search	Match query and rank results	Cosine Similarity
Video Playback	Play relevant video segment	OpenCV, VLC Python

4. Detailed Explanation of System Components

Video Input: Load videos in standard formats (MP4) for processing.

Audio Extraction: Convert video streams to WAV audio using FFmpeg and Pydub.

Speech Transcription: Use Whisper AI model to generate timestamped transcripts.

Indexing: Create TF-IDF vector space from transcript lines for semantic retrieval.

Query Search: Transform user query to TF-IDF vector and compute cosine similarity to rank segments.

Playback: Retrieve top segment and play using Open CV or VLC bindings.

5. Demonstration

Demo Description: User enters query System returns top 3 segments with time stamps and score
Playback automatically starts at the relevant time stamp

Demo Video Link: (Upload to cloud platform like Google Drive or YouTube and include URL here)

6. Takeaways, Limitations & Future Work

Takeaways:

Efficient transcript-based video retrieval

Segment-level timestamped playback

Limitations:

Reduced accuracy for noisy audio or multiple speakers

Base Whisper model may miss technical terms

Future Work:

Semantic search using embedding

GUI/Web interface for easier interaction

Multilingual support

Integrate face/object recognition to improve content filtering

7. References & Attributions

Whisper ASR – OpenAI

SpeechRecognition Python Library

Scikit-Learn TF-IDF Vectorizer

OpenCV for video handling

VLC Python Bindings

Git Hub link

<https://github.com/mscs24047-SyedzainulAbideen/MSCS24047-PROJECT-IR>