documentaion pour indexation

📄 Video Indexing Pipeline Documentation

🎯 Overview

This video indexing system transforms raw MP4 video files into structured, searchable, and semantically rich metadata using a combination of speech recognition, natural language processing (NLP), computer vision, and machine learning. It provides both textual summaries and visual annotations, enabling advanced search and real-time detection within video content.

🔁 Complete System Flow

Step 1: **User Uploads a Video**

- Users upload an `.mp4` video file through the Gradio interface.
- The file is passed to the `process_video()` function, which orchestrates the entire pipeline.

Step 2: **Audio Transcription (Speech-to-Text)**

Function: `transcribe_audio(video_path)`
Tool: OpenAI Whisper (`base` model)

Breakdown:

- The audio stream is extracted from the video.
- Whisper transcribes the audio into text.
- Tries to use detailed word-level timestamps directly from Whisper.
- If Whisper does not return `words`, it estimates timestamps by evenly distributing words across the segment's duration.

Output:

Each word segment includes:

```
{
  "text": "hello",
  "start": "00:00:10",
  "end": "00:00:11",
  "start_seconds": 10.0,
  "end_seconds": 11.0
}
```

This timestamping allows for precise search and alignment with visuals later in the process.

Step 3: **Named Entity Recognition (NER)**

Function: `extract_named_entities(texts)`
Tool: spaCy NLP (`en_core_web_sm` model)

Breakdown:

- The word-level transcript is grouped into chunks of 10 words.
- Each chunk is processed using spaCy's entity recognizer.
- spaCy extracts entities such as:
  - `PERSON` (e.g., "Elon Musk")
  - `ORG` (e.g., "OpenAI")
  - `GPE` (locations like "Tunisia")
  - `DATE` (e.g., "2025")
- Each entity is labeled and timestamped.

Purpose:

- Enables users to query **who** or **what** is being discussed.
- Improves search accuracy for important subjects.

Step 4: **Object Detection (YOLOv8)**

Function: `detect_objects(video_path)`
Tool: YOLOv8 (from `ultralytics`)

Breakdown:

- Every Nth frame (approx. 5-second intervals) is extracted.
- Each frame is processed using the YOLOv8 deep learning model.
- YOLO returns:
  - Bounding boxes
  - Class labels (like `person`, `car`, `dog`)
- Only unique object names are stored per frame.

Purpose:

- Ties visual evidence to specific timecodes.
- Enables users to ask: "When did a laptop appear?" or "When are people on screen?"

Step 5: **Face Detection**

Function: `detect_faces(video_path)`
Tool: `face_recognition` (HOG-based model)

Breakdown:

- Every Nth frame (~5 seconds) is converted to RGB.
- Faces are detected using HOG (Histogram of Oriented Gradients).
- For each detected face:
  - A counter is incremented.
  - A frame snapshot is saved (e.g., `face_frame_100.jpg`).

Purpose:

- Detects **presence** of individuals, even when they're not speaking.
- Can be extended for future **face recognition** or celebrity detection.
- Helps infer speaker identity based on visual data.

Step 6: **Semantic Embedding and Indexing**

Function: `semantic_indexing(segments)`
Tools: SentenceTransformers (MiniLM-L6-v2), FAISS

Step-by-Step:

📘 **Chunking:**

- Word segments are grouped into blocks of 10 (to create coherent text samples).
- Each chunk contains a small paragraph's worth of speech.

🧠 **Embedding with MiniLM:**

- Each chunk is converted into a **dense vector** (384-d float array) using MiniLM.
- MiniLM captures **semantic relationships**:
  - "plane" ≈ "aircraft"
  - "buy" ≈ "purchase"

⚡ **Indexing with FAISS:**

- FAISS is used to build a vector index that supports **approximate nearest-neighbor search**.
- This allows fast semantic similarity comparisons.
- Enables the user to search by **meaning**, not just exact words.

Outputs:

- `index`: FAISS search object
- `embeddings`: Raw vectors
- `texts`: Chunks of text

- `chunks`: Metadata (start time, end time, etc.)

Step 7: **Search Engine Integration**

Function: `search_in_video(...)`

Breakdown:

- User types a query like: “machine learning,” “dog,” or “Elon Musk.”
- The system:
  1. Encodes the query using MiniLM.
  2. Searches semantic index (FAISS) for similar chunks.
  3. Searches raw text in named entities and object detection results.

Outputs:

- Sorted results with type (`transcript`, `named_entity`, `object`)
- Metadata including timestamps, text content, and match reason.

Step 8: **Saving Results to Disk**

Function: `save_output(...)`

Files Generated:

File	Content
`video_indexed.txt`	Formatted transcript + object/entity summary
`video_indexed.json`	Structured data for programmatic access
`video_indexed_index.faiss`	Vector index for semantic search
`face_frame_X.jpg`	Snapshots of frames with faces

Use:

- For download, re-analysis, or offline querying.
- Can be shared or imported into other systems.

Step 9: **Real-Time Object Detection**

Function: `realtime_object_detection(source)`

Breakdown:

- Loads video (or webcam input).
- Runs YOLOv8 on every frame.
- Draws bounding boxes and confidence scores on frames.

- Outputs an `.mp4` file with visual overlays.

Step 10: **Gradio Web Interface**

Function: `create_interface()`

Features:

- Drag-and-drop video upload
- 3 main tabs:
  1. **Process Video**: Run full pipeline, download results
  2. **Search Video**: Type queries, view matched timestamps, jump to time
  3. **Real-time Detection**: Upload or stream video, view annotated output

Bonus:

- Works in Google Colab and local environments
- Supports forced download buttons if needed

🔎 Summary Table: Component Roles

Component	Tool/Library	Role
Transcription	Whisper	Convert speech to text
Entity Detection	spaCy	Extract persons, orgs, dates
Object Detection	YOLOv8	Detect objects in frames
Face Detection	face_recognition	Detect presence of people
Embedding	MiniLM (SBERT)	Convert chunks to vectors
Indexing/Search	FAISS	Enable fast semantic search
UI	Gradio	Provide interactive search + processing

📦 Output Summary

File	Purpose
`video_indexed.txt`	Readable summary for humans
`video_indexed.json`	Structured results (NER, objects, etc.)
`video_indexed_index.faiss`	Searchable vector index
`face_frame_X.jpg`	Frames saved with face detections
`annotated_video.mp4`	Real-time detection annotated output

💡 Use Case Ideas

- **Lecture Analysis**: Find where terms like “Bayes Theorem” were mentioned.
- **Security**: Detect faces, weapons, or unusual objects.
- **Content Indexing**: Automatically create searchable archives of podcasts or news clips.
- **Accessibility**: Provide text captions and summaries.
- **Marketing Analytics**: Detect brands/logos/people in promotional content.
