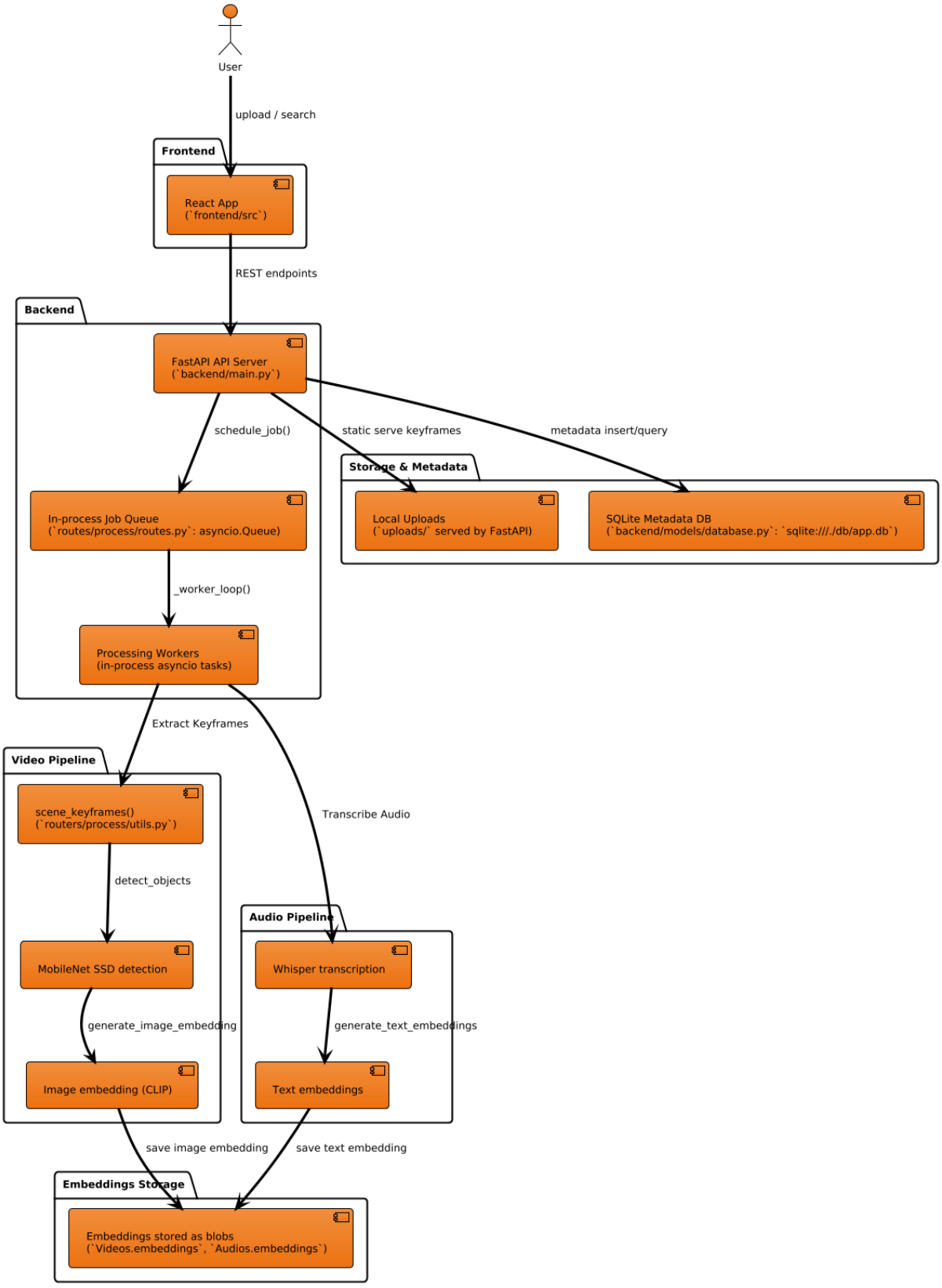
System Overview



Approach for implementation of vector similarity search across different media types

The system implements a hybrid vector search using cosine similarity calculated in-memory. The system stores serialized NumPy arrays in a standard SQLite database and performs brute-force comparison during search (following Option 3 as recommended in the technical test instructions).

1. Embedding Generation

Different models are used to generate vector embeddings depending on the media type:

- **Text & Audio Transcriptions:**
    - **Model:** sentence-transformers/all-MiniLM-L6-v2
    - **Usage:** Used to convert search queries and audio transcription segments into 384-dimensional vectors.
    - **Location:** utils.py (generate_text_embeddings)
- **Video Keyframes (Images):**
    - **Model:** sentence-transformers/clip-ViT-B-32 (CLIP)
    - **Usage:** Used to convert extracted video keyframes into 512-dimensional vectors. This allows for semantic image search.
    - **Location:** utils.py (generate_image_embedding)

2. Storage of embeddings

- **Database:** SQLite
- **Format:** Embeddings are serialized as compressed NumPy arrays (.npz format) using numpy.savez_compressed.
- **Columns:**
    - Videos.embeddings: Stores CLIP embeddings for video keyframes.
    - Audios.embeddings: Stores MiniLM embeddings for transcribed audio segments.
- **Data Type:** LargeBinary (BLOB)

3. Search Implementation

The search logic is in routes.py:

- **Video-to-Video Similarity:**

1. Retrieves the embedding blob of the reference video (specified by db_id).
2. Iterates through all video records in the database.
3. Deserializes the binary blob for each candidate into a NumPy array.
4. Computes cosine similarity between the reference vector and candidate vectors using numpy.dot and numpy.linalg.norm.
5. Sorts result by the calculated score.

- **Text-to-Audio Similarity:**
    1. Converts the user's text query into a vector using the all-MiniLM-L6-v2 model.
    2. Iterates through all audio records.
    3. Deserializes the embeddings for each audio segment.
    4. Computes cosine similarity between the query vector and each segment vector.
    5. Returns the most relevant audio segments sorted by similarity score.

Handling of larger video and audio files

Video Optimizations

- **Keyframe Extraction:**
    - **Frame Skipping:** The system does not process every frame. It samples every N-th frame (default: 5), reducing the workload by 80%. However, there is a drawback where this might miss short events in video.
    - **Scene Change Detection:** It calculates the pixel difference between consecutive sampled frames. If the difference is below a threshold (diff_thresh=10.0), the frame is considered static and discarded. This ensures that expensive object detection and embedding generation only run on distinct, meaningful frames rather than repetitive ones.
- **Streaming I/O:**
    - The system uses OpenCV's VideoCapture to read frames sequentially from the disk. This ensures that the full video file is never loaded into RAM at once, keeping memory usage constant regardless of video length.
- **Lightweight Model Architecture:**

    - **Object Detection:** Utilizes MobileNet-SSD, a model specifically architected for mobile and embedded devices. It is significantly faster and less resource-intensive on CPUs compared to larger models like YOLOv8.

Audio Optimizations

- **Aggressive Downsampling:**
    - Before processing, all audio is resampled to 16,000 Hz. Since standard audio is often 44.1kHz or 48kHz, this reduces the raw data size before it reaches the inference stage, lowering RAM usage.
- **Sliding Window Inference (Chunking):**
    - The system uses the HuggingFace pipeline with stride_length_s=(4, 2). Instead of attempting to process a long file in a single, it breaks the audio into small, overlapping chunks. This allows the system to transcribe arbitrarily long files with a fixed, low memory footprint.
- **Efficient Model Selection:**
    - **Transcription:** Deploys openai/whisper-tiny, the smallest available Whisper model (~39M parameters). It is loaded with low_cpu_mem_usage=True to optimize weight loading, making it viable for systems without a discrete GPU.

Potential areas of improvement with more computational resources

1. Upgrade to State-of-the-Art Models

With powerful GPUs, it will be possible to replace the current lightweight "mobile" models with more powerful models:

- **Object Detection:** Switch from MobileNet-SSD to YOLOv8-Large. This would drastically improve detection accuracy and the variety of recognizable objects from ~20 classes to thousands (depends on pre-training).
- **Transcription:** Upgrade from whisper-tiny to whisper-large-v3. The tiny model trades accuracy for speed; the large model offers better accuracy but requires ~10GB VRAM.
- **Embeddings:** Move from clip-ViT-B-32 to clip-ViT-L-14 for richer semantic understanding of video frames.

2. Dedicated Vector Database (ANN Search)

- **Current Bottleneck:** The system performs a linear, brute-force search ($O(N)$) in Python, calculating cosine similarity against every record in SQLite.
- **Improvement:** Integrate a dedicated vector database like pgvector. Pgvector can use Approximate Nearest Neighbor (ANN) algorithms to search millions of vectors in milliseconds ($O(\log\ N)$), enabling the system to scale to massive datasets without slowing down.

3. Advanced Video Analysis

More compute allows for computationally expensive extraction techniques:

- **OCR (Optical Character Recognition):** Run models like PaddleOCR to index text that appears on screen (e.g., street signs, vehicle license plates).
- **Dense Sampling:** Increase the sampling rate from 1 frame every 5 seconds to every second, ensuring brief events are not missed.

4. Audio Intelligence

- **Speaker Diarization:** Use PyAnnote to distinguish who is speaking (e.g., "Speaker A", "Speaker B").
- **Audio Event Detection:** Implement models like YAMNet to index non-speech sounds (e.g., "glass breaking", "sirens"), allowing users to search for audio events.

5. Horizontal Scalability

- **Current State:** Processed by a single server instance. If the load increases, that one machine becomes a bottleneck.
- **Improvement:** Decouple the processing logic using a distributed task queue like Celery with Redis. This would allow you to spin up multiple worker nodes to process the queue in parallel, which makes it more scalable.

Speed vs Accuracy Trade offs

Video Processing Trade-offs

Feature	Speed Optimization	Accuracy Trade-off
Frame Sampling	Only processes 1 frame every 5 seconds	**Lower Temporal Resolution:** Short events (e.g., a person running past the camera in 2 seconds) might be completely missed if they fall between sampled frames.
Model Choice	Uses **MobileNet-SSD**, designed for mobile devices. It runs very fast on CPUs.	**Lower Detection Accuracy:** MobileNet is less accurate than larger models like YOLOv8. It may fail to detect small objects, objects in poor lighting, or partially occluded objects.
Resolution	Resizes images to **300x300** for the neural network input.	**Loss of Detail:** Small details (like text on a shirt or a distant face) are lost in the downscaling, making them impossible to detect or recognize.

Audio Processing Trade-offs

Feature	Speed Optimization	Accuracy Trade-off
Model Size	Uses **whisper-tiny** (~39M parameters). It is the fastest but smallest available Whisper model.	The tiny model struggles with accents, background noise, overlapping speech, and technical terminology compared to the base, small, or large models.
Resampling	Downsamples audio to **16kHz** before processing to reduce data size.	High-frequency audio cues are lost. While usually fine for speech, this degrades the ability to analyze non-speech background sounds or music.