**SuperBryn Inc.**

**Task 1:**

**Problem Statement: We have a bunch of videos (say 10 or so). A user asks a query. We need to pick the video and its timestamp that's most relevant for the user's query.  
  
This has to be a really fast operation (within 1 sec)  
  
Research several available options and pick two approaches - the fastest to implement, and the best in performance. Share the reasoning for your choice**

**Solution:**

**Different Possible Approaches**:

1. Basic Transcript-Text based Searching: Using techniques like Fuzzy Search Techniques, Statistical techniques like TF-IDF, Regex based search techniques etc.
2. Transcript-text Embedding-Based Search: Using techniques like vector embedding based search.
3. Frame-based embedding Search: By creating embedding for frames using models like CLIP and then doing search over them by first tokenizing the query text and then embedding that into the same CLIP model and then searching over the data store.

Let’s learn the Transcript-text’s based embedding search and Frame based embedding search in more details.

**Transcript-text Embedding-Based Search**:

Here in this approach, we first need to extract the audio from video file and then we need to transcribe that into text with timestamp.

Then we need to generate the embedding for these transcripts:

Now for generating embedding for the transcript there are different strategic ways we can use:

1. Using a chunk-based approach: In this we divide the whole transcripts of a video into multiple chunks, so this can be helpful for marking the timestamps for the relevant portion
2. Using overlapping chunk-based approach: In this we divide the transcript of a video into overlapping chunks which can help in maintain the context and thus can generate more context aware results.
3. Using Nature Language Based Chunks: The chunks can also be created based on the natural language-based parts of speech tagging like we can generate a single chunk as including 3 sentences together so that it will have overlap of one sentence with previous chunk and similarly with next upcoming chunk.

After generating chunk, we need to generate embedding of these chunks. For this we can employee also use different model.

1. OpenAI based vector Embedding
2. Hugging face-based vector embedding

Now after generating the embedding, we need to store them in a vector store which will be so crafted that will help us for easy retrieval.

So, we can use vector Store like FAISS (Facebook AI Similarity Search) for doing query search here.

Now to access the relevant metadata as respective video id and timestamp for the result embeddings. we need to store the embeddings in FAISS with int64 type IDs. And we separately maintain an external dictionary mapping these IDs to their metadata.

**Frame Based Embedding Search**:

In this technique, we extract some frames from video by deciding the extraction rate as frame /second for the video and create embedding for each frame it using models like CLIP and then store them in models like FAISS and help for faster search.

We also need to apply the similar indexing strategy as we have applied in previous approach for accessing the relevant metadata for each result’s frame.

Here advance strategies can also be applied like;

1. extracting the scenes from video by detecting them then embedding the scene for better contextual resemblance and thus generating higher precision.
2. Extracting the objects from the frames using models like YOLOv5 and then generating embedding for the extracted object’s metadata and then doing over them the similarity search for the query text.
3. Fusion of transcript-text embeddings and frame embeddings, typically by concatenating them into a joint embedding.

My proposed solution:

1. Fastest to Implement: **Transcript Embedding Search with FAISS**

How it works

1. Transcribe videos with an ASR (e.g., Whisper), splitting them into chunks with timestamps.
2. Embed each chunk using a text embedding model (e.g., text-embedding-ada-002).
3. Index these embeddings in a vector database (e.g., FAISS).
4. On query, embed the user’s text and perform nearest-neighbor search → directly returns the most relevant video & timestamp through metadata mapping.

Why it’s fast

* Embedding and indexing are straightforward; no visual processing needed.
* FAISS ensures sub-100 ms retrieval even for thousands of chunks.
* Purely vector-based; no need for keyword or ANN fusion.

2. **Highest Performance: Multimodal CLIP‑Based Retrieval with Transcript + Frame Embedding**

How it works

1. Sample frames and transcripts together per video chunk.
2. Generate:
   * Frame embeddings via CLIP (image encoder)
   * Transcript embeddings via text encoder
3. Concatenate these into joint embeddings for each chunk.
4. Index them in FAISS.
5. At query:
   * Embed the query text and embed-zero or partially text-based joint vector
   * Perform ANN search to retrieve semantically and visually relevant chunks
   * Return video & timestamp via metadata mapping.

Why it’s high performance

* Leverages CLIP’s zero-shot cross-modal capability, capturing both spoken and visual cues.
* Provides superior precision via joint semantic-visual context.