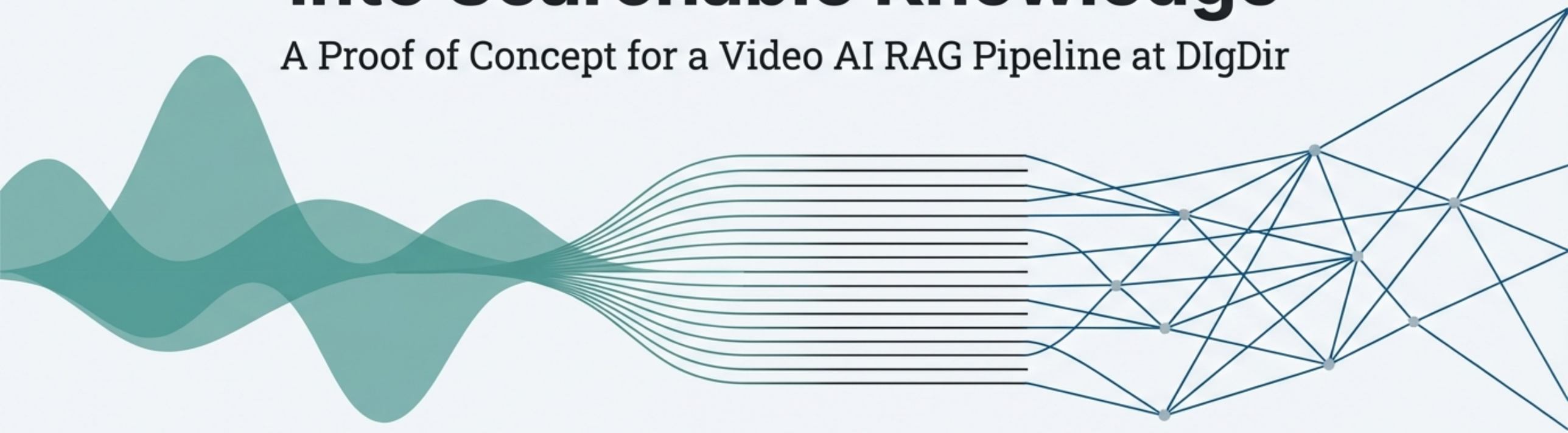
Turning Our Conversations into Searchable Knowledge

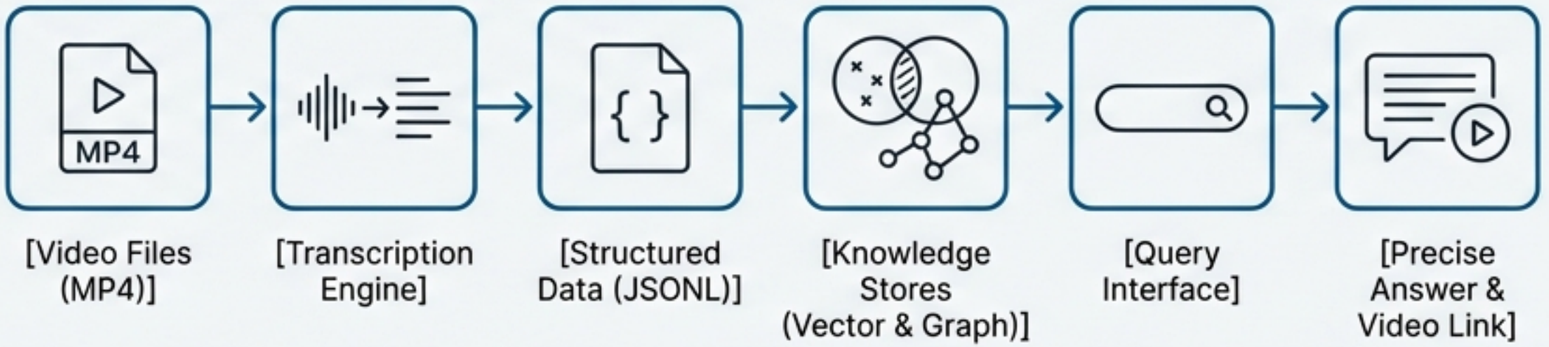
A Proof of Concept for a Video AI RAG Pipeline at DIgDir



Our video library is a vast, untapped asset.

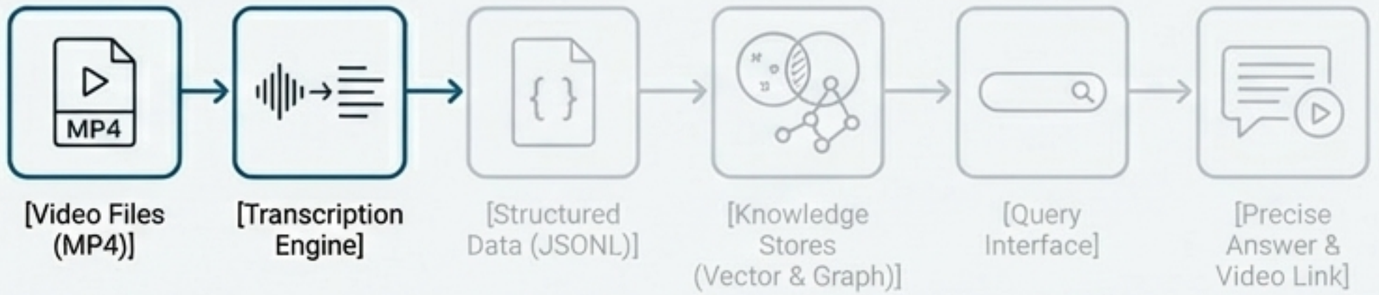
Finding specific information inside hours of video content is slow, manual, and often impossible. Critical knowledge remains locked away.

This POC builds an automated pipeline to transcribe, structure, and query our video content, turning it into a dynamic knowledge base.

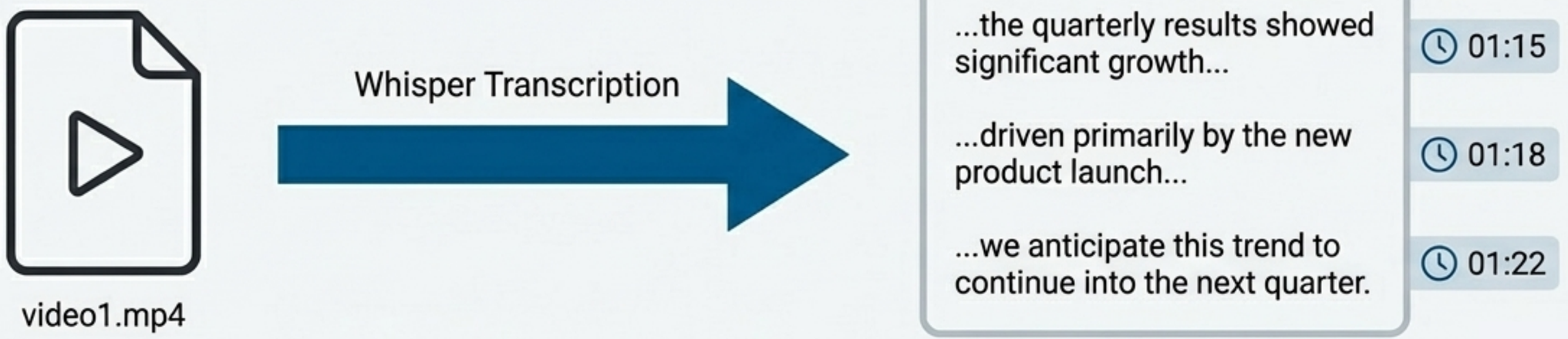


This is the journey our data will take.

First, We Turn Spoken Words into Timestamped Data



We use OpenAI's Whisper (`large` model) for highly accurate speech-to-text conversion. The key is not just transcribing *what* is said, but *when* it is said.



The Output is More Than Text; It's Structured for a Database

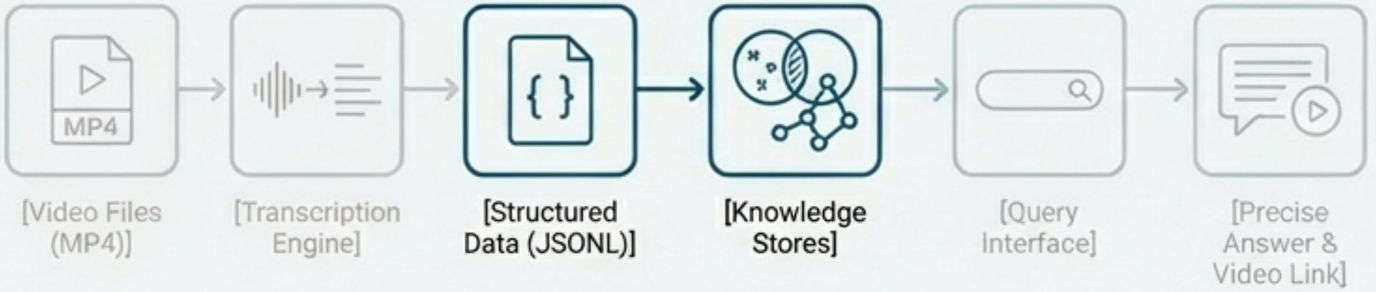

```
{
  "video_id": "video1",
  "video_path": "/Users/adi/digdir-video-
ai/videos/video1.mp4",
  "segment_id": 42,
  "start": 125.5,
  "end": 132.8,
  "text": "The core of the issue is about
data sharing and the platform
 architecture we decide on..."
}
```


Traceability: Every text segment is linked to its source `video_id`.

Precision: `start` and `end` timestamps allow us to pinpoint the exact moment in the video.

Usability: This JSONL format is perfectly designed for ingestion into downstream systems.

LightRAG Orchestrates the Knowledge Base



LightRAG provides the framework to manage the entire Retrieval-Augmented Generation process. It handles data ingestion, indexing, and provides the interface for querying.

```
# Initialize the core RAG engine
rag = LightRAG(
    working_dir="./lightrag_store",
    embedding_func=openai_embed,
    llm_model_func=gpt_4o_mini_complete,
)
await rag.initialize_storages()
```

We configure LightRAG with our chosen embedding models and local storage, creating a self-contained system.

We Prepare Contexts by Fusing Text with Metadata

Raw Transcript Segment

The core of the issue is about data sharing...

Enriched Context for Indexing

[video_id=video1;start=125.5; end=132.8;segment_id=42]

The core of the issue is about data sharing...

Before inserting data into LightRAG, we prepend each text segment with a metadata header. This ensures that every retrieved piece of information carries its origin with it, making the search results fully traceable and actionable.

Our Architecture Supports Both Immediate Search and Future Insights

For lightning-fast semantic search.

Transcript segments are stored in **PostgreSQL**. We will later add a vector column (using **pgvector**) to find text based on conceptual meaning, not just keywords.

For future graph-based analysis.

The same data is modeled as a graph in **Neo4j** (`(Video)-[:HAS_SEGMENT]->(VideoSegment)`). This unlocks future capabilities like topic clustering, speaker identification, and analyzing relationships between concepts across videos.

Ingesting Transcripts into Our Knowledge Stores

Relational & Vector Storage

```sql
-- Create the table to hold video segments
CREATE TABLE video_segments (
    id SERIAL PRIMARY KEY,
    video_id TEXT NOT NULL,
    start DOUBLE PRECISION,
    text TEXT, ...
);

-- Insert records from our JSONL file
INSERT INTO video_segments (...) VALUES (...);
```

Graph Storage


```cypher
// Create nodes and relationships for each segment
MERGE (v:Video {video_id: $video_id})
MERGE (s:VideoSegment {start: $start, ...})
MERGE (v)-[:HAS_SEGMENT]->(s)
```

The structured JSONL output from the transcription stage is loaded directly into both databases, populating our knowledge base.

Now, We Ask a Specific Question

"Når snakker de om datadeling og plattform?"

(When do they talk about data sharing and platform?)

With the knowledge base indexed, we can now perform semantic search. The system looks for segments that are conceptually similar to the query, even if the exact keywords aren't used.

How the System Finds the Answer: A Look at Semantic Retrieval

Step 1. Vectorize Query

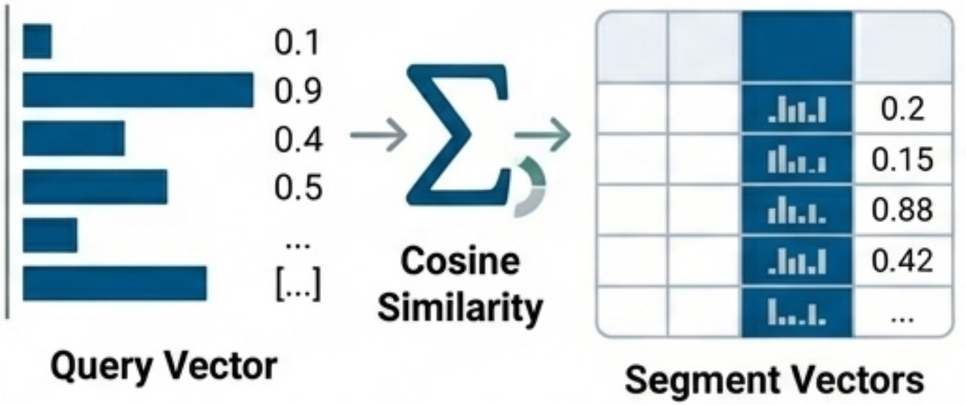
"Når snakker de om...?"

openai_embed()

Query Vector

2. Compare Vectors

PostgreSQL



Query Vector

Segment Vectors

3. Return Top Matches

1. 0.891
2. 0.875
3. 0.854
4. 0.821
5. 0.799

Top 5 Results

The system returns the segments whose vectors are mathematically closest to the query vector.

We find results based on meaning and context, not just keyword matching.

The Result: A Precise, Actionable Answer

video_id	start_time	score	text_snippet
video2	145.21s	0.891	...vi må se på en felles plattform for datadeling, slik at etatene kan samhandle bedre...
video1	88.45s	0.875	...det handler ikke bare om data, men om hvordan plattformen teknisk sett er skrudd sammen for deling...
video1	125.50s	0.862	The core of the issue is about data sharing and the platform architecture we decide on...
video2	210.11s	0.859	...denne plattformen er nøkkelen til effektiv datadeling fremover...

Each result provides the source video, the exact start time, the relevance score, and the relevant text for immediate verification.

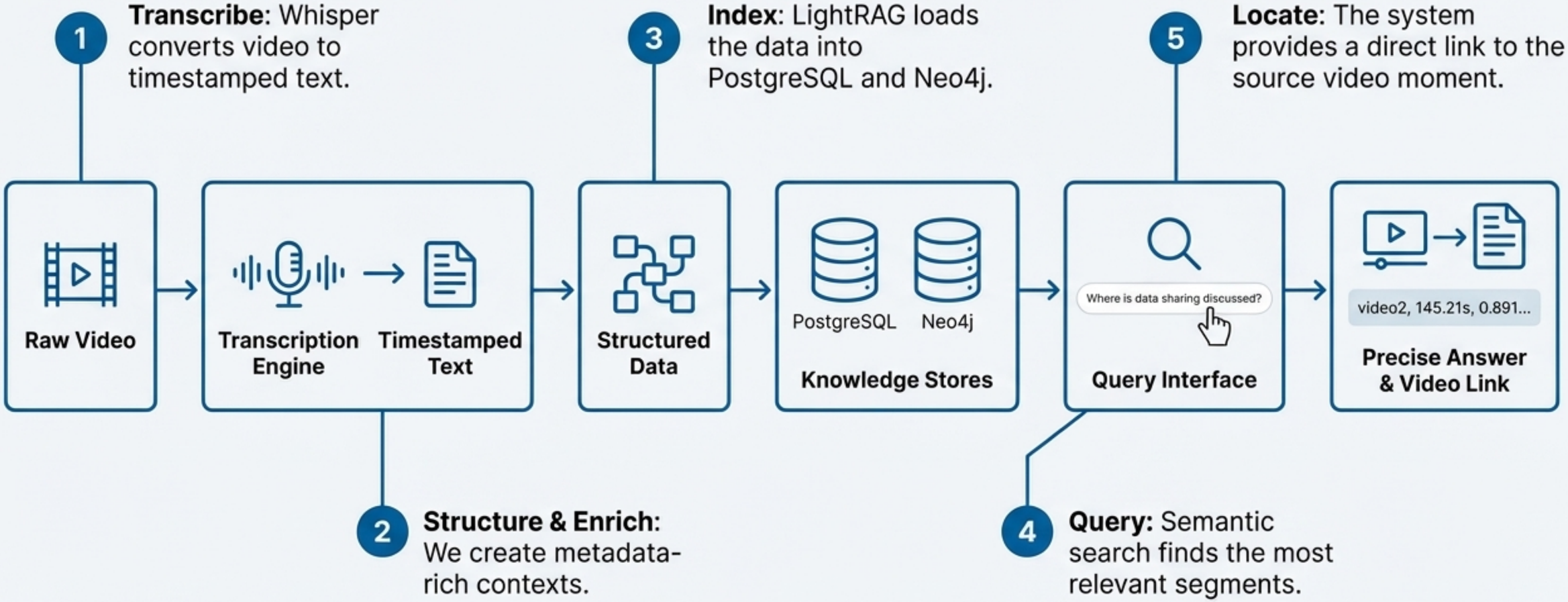
We Don’t Just Tell You the Answer.
We Show You the Exact Moment.


```
# Take the top search result
hit = results[0]

# Instantly generate a video player
# cued to the right time
show_video_segment(
    hit["video_path"],
    hit["start"]
)
```

The entire pipeline culminates in this experience: a direct link from a complex question to the precise moment of discussion in the source media.

The Full Journey: From Raw Video to Verifiable Insight



This POC Unlocks New Capabilities and Defines a Path Forward

What We Can Do Now

- Instantly search across our entire video archive for specific topics.
- Drastically reduce time spent on manual research and knowledge discovery.
- Create a foundational asset for all future AI-driven knowledge tools.
- Provide verifiable, timestamped sources for any retrieved information.

Where We Go From Here

- **Scale**: Ingest the full DIgDir video library.
- **Develop**: Build a user-friendly web interface for querying.
- **Integrate**: Connect the search functionality with existing internal tools.
- **Analyze**: Begin exploring graph-based queries in Neo4j to uncover deeper insights.

Technical Details & Resources

1. Core Components

Whisper Model	`large` (Chosen for accuracy)
Compute Device	`cpu` (Forced for stability during the POC, as Apple Silicon (MPS) can produce NaNs.)
Orchestration	`LightRAG`
Embedding Model	`openai_embed`
Databases	PostgreSQL (for vector search) & Neo4j (for graph analytics)

---

2. System Dependencies

- openai-whisper
- lightrag-hku
- psycopg2-binary
- neo4j

---

3. Resources

- Code Repository: [Link to internal Git repository for this POC]
- Source Notebook: [Link to the source Python Notebook]