Multimodal Video-Only RAG - Detailed Development Plan

Project Scope

Goal: Build a fully-local, video-only Retrieval-Augmented Generation (RAG) pipeline that answers natural-language queries and returns (a) a textual answer citing timestamps and (b) clipped video segments corresponding to those timestamps.

All code in, managed in virtualenv.

Directory Layout (Monorepo)

```
repo_root/
├─ data/                  # raw + processed
│   ├─ raw_videos/
│   ├─ frames/            # extracted .jpg
│   └─ clips/             # returned snippets
├─ src/
│   ├─ phase1_audio/
│   ├─ phase2_visual/
│   ├─ phase3_db/
│   ├─ phase4_retriever/
│   ├─ phase5_generation/
│   └─ phase6_clipper/
├─ tests/
├─ conf/                  # .env, model paths, settings.yaml
└─ README.md
```

Each sub-package has its **own** `** entry point** so phases can be run as standalone modules `.

Phase 1 – Audio Processing & Embedding

1-A | Audio Extraction & Transcription

Item	Detail
Tool	OpenAI Whisper (CPU, `medium` model) via `openai-whisper`
Script	`src/phase1_audio/extract_transcribe.py`
Algorithm	1. `ffmpeg -i video.mp4 -vn -acodec pcm_s16le temp.wav`

2. Run Whisper **with** `to obtain JSON with per-word timings. | | **Output** |data/transcripts/{video_id}.json– list of words withstart,end,word`.|

1-B | 10-Second Segmentation & Normalisation

- `segment_transcript.py` converts word list → 10 s buckets.
- Normalise text: lower-case, strip punctuation, collapse whitespace. **Silent buckets** (≤ 2 words) retain empty string but keep times.

```
{
  "video_id": "demo.mp4",
  "segments": [
    {"start": 0.0,   "end": 10.0,  "text": "introduction of the device ..."},
    {"start": 10.0,  "end": 20.0,  "text": ""},
    ...
  ]
}
```

1-C | Embedding

- `embed_text.py` loads CLIP **ViT-B/32** text encoder (`open_clip`).
- Batched forward pass (≤ 32 segments/batch, CPU).
- Persist embeddings → `phase3_db` via gRPC interface (see Phase 3 API).

Deliverables Phase 1

Deliverable	Acceptance Criteria
`extract_transcribe.py`	JSON transcript matches Whisper CLI output; unit test verifies ≥ 95 % coverage for timing fields.
`segment_transcript.py`	Exactly ⌈duration/10⌉ segments with correct start/end; silent buckets kept.
`embed_text.py`	Embeds N segments in ≤ (N / 8) s on dev laptop (benchmark).

Phase 2 – Visual Frame Extraction & Embedding

2-A | Frame Sampling (10 s Grid)

- `ffmpeg -ss {t} -i video.mp4 -frames:v 1 frames/{id}_{t}.jpg` for t = 0,10,20, …
- Metadata: `start = t`, `end = t + 10`.

2-B | Frame Embedding

- Use **same CLIP model** image encoder.
- Images resized to 224×224, center crop.
- Batch size = 64.
- Store embedding + metadata.

Deliverables Phase 2

Deliverable	Criterion
`sample_frames.py`	All frames exist; naming matches regex `videoid_\d+.jpg`.
`embed_frames.py`	Embedding file or direct DB push validated by checksum.

Phase 3 – Vector Store Service (ChromaDB)

3-A | Containerised DB

- Docker service in `docker-compose.yml` – Chroma with `CHROMA_DB_IMPL=duckdb+parquet`, volume-mounted under `data/chroma/`.

3-B | Schema & API

One **collection** `video_segments` with schema:

```
{
  "id": str,                    # UUID
  "embedding": List[float],
  "metadata": {
       "video_id": str,
       "modality": "audio"|"frame",
       "start": float,          # seconds
       "end": float,
       "path": str|null         # jpeg for frames
  }
}
```

Expose a thin gRPC layer (`src/phase3_db/server.py`) so Phase 1 & 2 import `DbClient` without raw Chroma dependency.

3-C | Batch Ingestion

- `DbClient.add_batch(vectors, metadatas)` with **batch_size=20**.
- Unit test: ingest 100 dummy vectors, assert count==100.

Deliverables Phase 3: Docker compose, `DbClient`, ingestion tests.

Phase 4 – Retrieval Service

4-A | Query Embedding

- `embed_query.py` → CLIP text encoder.

4-B | Search Endpoint

`Retriever.search(query: str, k:int=10) -> List[Document]`

- Executes single cosine-similarity search in Chroma.
- Returns **rank-ordered** list; each `Document` holds `text` (for audio) *or* `"<IMAGE_FRAME>"` placeholder plus metadata.

Deliverables Phase 4

- `src/phase4_retriever/retriever.py` with pydantic models and coverage tests.

Phase 5 – LLM Generation Micro-service

5-A | Prompt Template (LangChain)

```
SYSTEM = "You are a helpful assistant ... always cite timestamp and video
ID."  # in .jinja
DOC_PROMPT = "[{metadata[start]:.2f}-{metadata[end]:.2f}s |
{metadata.video_id}] {page_content}"
QA_CHAIN = RetrievalQA.from_chain_type(
    llm=OpenAI(...),
    retriever=Retriever(...),
    chain_type="stuff",
    chain_type_kwargs={"document_prompt": DOC_PROMPT, "prompt": SYSTEM}
)
```

5-B | API Wrapper

- FastAPI endpoint `/ask` → returns JSON: `{ "answer": str, "sources": List[metadata] }` (sources are the same objects given to the LLM).

Deliverables Phase 5: prompt files, `qa_service.py`, e2e test with canned video and expected timestamp in answer.

Phase 6 – Clip Builder

6-A | Timestamp Parser

- Regexes for `HH:MM:SS`, `MM:SS`, `SS.S` → seconds.
- Merge intervals with ≤ 2 s gap.

6-B | Clip Extraction

- Use `moviepy.VideoFileClip(video).subclip(start,end).write_videofile(outfile, codec="libx264")`.

- Runs asynchronously (`asyncio`) so multiple clips export in parallel.

6-C | Clip Registry

- SQLite table `clips(id, video_id, start, end, path, created_at)` for re-use.

Deliverables Phase 6: `clipper.py`, integration test – feed answer JSON, verify clips exist and duration matches.

Milestone Timeline & Team Allocation

Week	Phase / Component	Owner	Key Review
1	Phase 1-A/B	Backend 1	Transcript JSON contract
2	Phase 1-C + Phase 2-A	Backend 1 / ML 1	Embedding fidelity test
3	Phase 2-B + Phase 3	ML 1 / DevOps 1	DB ingestion benchmark
4	Phase 4	Backend 2	Retrieval precision@10
5	Phase 5	ML 2	Answer quality & token costs
6	Phase 6	Backend 2	Full e2e demo

Readiness Checklist per Phase

Future Enhancements (Post-MVP)

1. **UI**: React front-end with embedded `<video>` + timeline markers.
