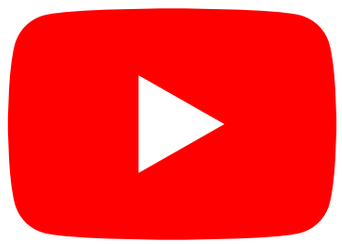
Talk with  Youtube without paying a cent

No Frameworks BS 😎

Francesco Zuppichini

Me

Francesco Saverio Zuppichini

Senior Full Stack ML Engineer

Goal

Full local Youtube RAG

youtube-dl download video

Hugging Face embeddings

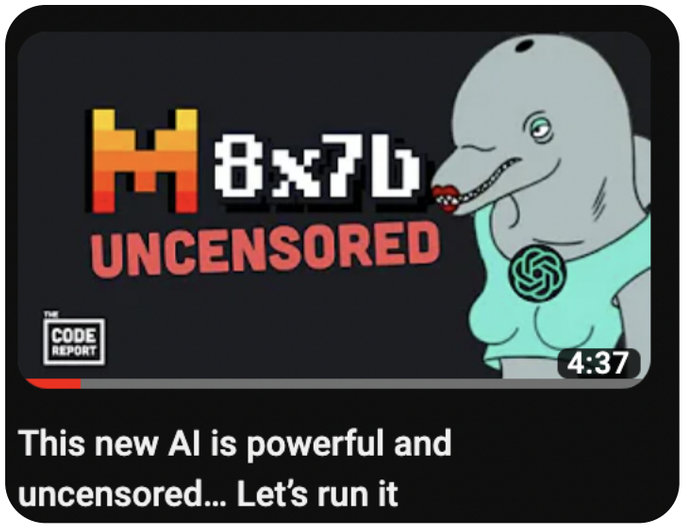
drant store/search

SQLite state

LLM

Example

All from the Python




```
^C
(.venv) (base) ➜  youtube-rag-cli git:(main) ✗ python main.py
[15:09:36] INFO     Load pretrained SentenceTransformer: all-MiniLM-L6-v2        SentenceTransformer.py:110
[15:09:37] INFO     Use pytorch device_name: cuda                                SentenceTransformer.py:216
[15:09:38] INFO     HTTP Request: PUT http://localhost:6333/collections/embeddings          _client.py:1027
                    "HTTP/1.1 400 Bad Request"
           INFO     Connected to DB!                                                        main.py:102
           INFO     [GyllRd2E6fg] Processing ...                                            main.py:105
>what is this model?
Batches: 100%|████████████████████████████████████████████████████| 1/1 [00:00<00:00,  1.02it/s]
[15:09:41] INFO     HTTP Request: POST                                                      _client.py:1027
                    http://localhost:6333/collections/embeddings/points/search "HTTP/1.1
                    200 OK"
[15:09:42] INFO     HTTP Request: POST http://localhost:11434/v1/chat/completions "HTTP/1.1 _client.py:1027
                    200 OK"
 The model mentioned in the context is a language model named "mixl 8X 7B." It requires a machine with suff
icient RAM (approximately 90 GB considering the user's 64 GB and the size of the model being around 26 GB)
to run. This model is from the new open-source Foundation, which users can download and use to generate unc
ensored large language models with approaching GPT performance.
```

Subtitles

```python
def download_subtitles_from_video_url(video_url: str) -> Path:
    video_id = get_id_from_video_url(video_url)
    output_dir = SUBTITLES_DIR / video_id
    output_dir.mkdir(exist_ok=True)
    outtmpl = str(output_dir / "subtitles")

    ydl_opts = {
        "writesubtitles": True,
        "subtitlesformat": "vtt",
        "subtitleslangs": ["en"],
        "writeautomaticsub": True,
        "outtmpl": outtmpl,
        "skip_download": True,
    }

    with YoutubeDL(ydl_opts) as ydl:
        ydl.download([video_url])

    output_path = output_dir / "subtitles.en.vtt"

    if not output_path.exists():
        raise KeyError("Not Subtitles")
    return output_path
```

Embeddings

```python
from sentence_transformers import SentenceTransformer

embeddings = SentenceTransformer("all-MiniLM-L6-v2")
query = "Hi!"
embeddings.encode(query).tolist()
```

Qdrant
Setup/Docker

```yaml
services:
  qdrant:
    image: qdrant/qdrant:latest
    restart: always
    container_name: qdrant
    ports:
      - 6333:6333
      - 6334:6334
    expose:
      - 6333
      - 6334
      - 6335
    volumes:
      - ./qdrant_data:/qdrant_data
```

Qdrant

Setup/Indexes

```python
def setup(db: QdrantClient, vector_size: int):
    try:
        db.create_collection(
            collection_name=COLLECTION_NAME,
            vectors_config=VectorParams(size=vector_size,
distance=Distance.COSINE),
        )
        db.create_payload_index(
            collection_name=COLLECTION_NAME,
            field_name="metadata.video_id",
            field_schema="keyword",
        )
    except UnexpectedResponse:
        pass
```

Qdrant
Store

```python
def embed(
    db: QdrantClient,
    embeddings: SentenceTransformer,
    chunks: list[Document],
):
    embeddings = embeddings.encode(chunks)
    db.upload_records(
        collection_name="embeddings",
        records=[
            models.Record(id=idx, vector=emb.tolist(), payload=doc)
            for idx, (emb, doc) in enumerate(zip(embeddings, chunks))
        ],
    )
```

Qdrant
Search

```python
def search(
    db: QdrantClient,
    embeddings: SentenceTransformer,
    query: str,
    video_id: Optional[str] = None,
    limit: int = 4,
) -> list[Document]:
    query_filter = (
        models.Filter(
            must=[
                models.FieldCondition(
                    key="metadata.video_id", match=models.MatchValue(value=video_id)
                )
            ]
        )
        if video_id
        else None
    )
    hits = db.search(
        collection_name="embeddings",
        query_vector=embeddings.encode(query).tolist(),
        limit=limit,
        query_filter=query_filter,
    )
    documents = [Document(**hit.payload) for hit in hits]
    return documents
```

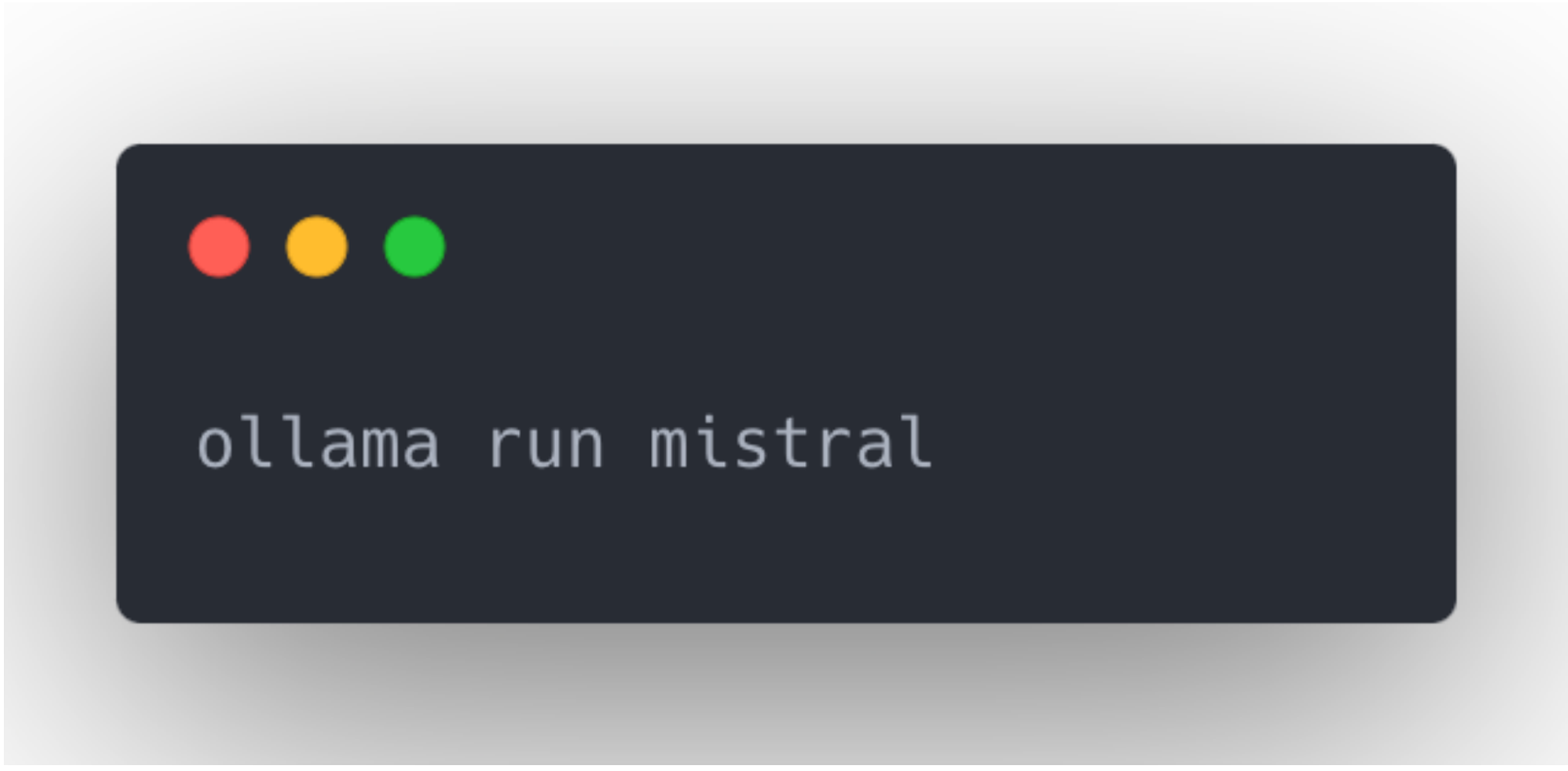
SQlite
Setup

```python
def setup() -> Connection:
    SUBTITLES_DIR.mkdir(exist_ok=True, parents=True)
    conn = sqlite3.connect("main.db")

    conn.execute(
        """
    create table if not exists videos (
        id TEXT primary key,
        title TEXT
    )
    """
    )

    conn.commit()
    return conn
```

SQlite
Crud

```python
def get_video_from_db(video_id: str, conn: Connection) -> Optional[Video]:
    curr = conn.execute("SELECT * FROM videos WHERE id=?", (video_id,))
    row = curr.fetchone()
    if row:
        columns = [column[0] for column in curr.description]
        return dict(zip(columns, row))
    else:
        return None


def save_video_to_db(video: Video, conn: Connection):
    conn.execute(
        "INSERT OR IGNORE INTO videos (id, title) VALUES (?, ?)",
        (video["id"], video["title"]),
    )
    conn.commit()
```

Ollama
Setup



https://ollama.com/library

Ollama
Prompt

```
Use the following pieces of context to answer the question at the end. If you don't
know the answer, just say that you don't know, don't try to make up an answer.

{context}

Question: {question}
Helpful Answer
```

Ollama
LLMs

```python
def get_answer(
    db: QdrantClient,
    embeddings: SentenceTransformer,
    model_client: OpenAI,
    question: str,
    video_id: str,
) -> Tuple[str, list[Document]]:
    prompt = Path("prompts/qa.prompt").read_text()
    documents = search(db, embeddings, question, video_id)
    context = "\n".join([document["page_content"] for document in documents])
    prompt = prompt.format(question=question, context=context)
    # logger.info(prompt)
    chat_completion = model_client.chat.completions.create(
        messages=[
            {
                "role": "system",
                "content": prompt,
            }
        ],
        model="mistral",
        max_tokens=1024,
    )
    return chat_completion.choices[0].message.content, document
```

Result

```
def main(video_url: str):
    embeddings = SentenceTransformer("all-MiniLM-L6-v2")
    conn = setup()
    vector_db = get_vector_db(vector_size=embeddings.get_sentence_embedding_dimension())
    logger.info("Connected to DB!")
    model_client = get_model_client()
    video_id = get_id_from_video_url(video_url)
    logger.info(f"[{video_id}] Processing ... ")
    if not get_video_from_db(video_id, conn):
        logger.info(f"[{video_id}] Downloading ... ")
        video_info = get_info_from_video_url(video_url)
        subtitles_path = download_subtitles_from_video_url(video_url)
        logger.info(f"[{video_id}] Splitting ... ")
        chunks = split_subtitles(subtitles_path, metadata=DocumentMetadata(video_id=video_id))
        logger.info(f"[{video_id}] Embeddings ... ")
        embed(vector_db, embeddings, chunks)
        save_video_to_db({"id": video_id, **video_info}, conn)
        logger.info(f"[{video_id}] Updating DB ... ")
    logger.info(f"[{video_id}] Done!")
    while question := input(">"):
        answer, sources = get_answer(
            vector_db, embeddings, model_client, question, video_id
        )
        print(answer)
```

Demo 

Thanks 

Heading
Heading
Subtitle

body

small

Heading
Heading 2
Heading 3

Subtitle Subtitle

body

small

extra small

Spaces

Subtitle

Title

Subtitle

Text

TITLE

Subtitle

Section