GenAI for Meeting Transcripts + RAG Implementation

•••

Data Journalism DC #44

Outline

- Using GenAI with Transcripts
  - Basic Concepts
  - Use Cases
    - Single Meeting
    - Multiple Meetings
    - Interview
    - Classroom
  - Pitfalls
  - Recap
- RAG Implementation
  - Concept / Motivation
  - Overview of Implementation
  - Chunk
  - Generate Embeddings
  - Retrieve
  - Run Prompt
  - Advanced RAG (time permitting)

Be sure to jump in and ask questions or share your experience!

Using GenAI with Transcripts: Basic Concepts

Recording Transcripts

- Always ask for permission to record transcript
- Available in Zoom, Google Meet, etc.
- Better audio quality -> Better transcript quality

Use GenAI to ask questions of transcripts

- We'll use Gemini for Workspace, but there are a lot of alternatives
- Be specific (ask for what you want)
- Ask for when something happened (timestamp or date) so that it's easier to manually inspect the transcript or ask follow up questions
- It really helps to have actually been there for the meeting/class

Using GenAI with Transcripts: Single Meeting

- Summarize this meeting.
- (Use Gemini's suggested questions.)
- What action items was <participant> given?
- What decisions were made in the meeting?
- What was the name of the startup/paper that <participant> mentioned?
- Clarify <topic>
- What was the problem that <participant> ran into and what solutions were suggested?
- Summarize the discussion about <topic> and include important quotes and timestamps.

Using GenAI with Transcripts: Multiple Meetings

- Summarize the work they did between dates <dates>
- (Use Gemini's suggested questions.)
- What did <participant> work on (give two week summaries)? Give every two-week period from July - September.
- When did <event> happen?
- Over the course of the project did the view of <aspect> change?
- Were there any important action items that didn't get completed?
- Specific Question: What database are we using?

Using GenAI with Transcripts: Interview

- Summarize.
- (Use Gemini's suggested questions.)
- What examples did Steve give of using AI for meetings?
- Pull 3 quotes from Steve about <topic> and give timestamps.
- I am writing an article about <specific topic>. What did Steve say about that?
- Give me all the followup questions the interviewer asked with timestamps.
- What was the answer to the question at <timestamp>?
- Find all the timestamps where <topic> was mentioned.

Using GenAI with Transcripts: Classroom

- Summarize this class.
- (Use Gemini's suggested questions.)
- What book, chapter, and pages did the instructor refer to?
- What did the instructor say specifically about <topic>? Include timestamps.
- What vocab words did the instructor mention?
- What sources beside the textbook did the instructor mention?
- Did the instructor mention any websites or other sources of information to look at?
- Were any pieces of information mentioned multiple times or explicitly referred to as important or similar?

Using GenAI with Transcripts: Pitfalls

- Can’t ask about thing not said in the meeting; especially closure on tasks
- When the original ideas are confusing to participants or conflate ideas, the AI will probably have the same problem
- When something is only said once, it’s very easy to get missed by the AI, especially when the quality of the transcript is low
- When an idea is misunderstood and talked about a lot and then clarified with only a small amount of discussion the AI can reiterate the wrong idea
- Like other GenAI use cases, you will always get an answer so “leading the witness” can lead to erroneous results

Using GenAI with Transcripts: Recap

- Ask specific questions
- Low quality transcripts work better than might seem reasonable,
- especially if a topic is talked about at length,
- but there can still be mistakes.
- A very useful tool when meeting participants agree on how they are going to use it
- If GenAI + Transcript made a mistake, maybe so did the participants
- I would still take notes in class and interviews, but for meetings it can help with reducing overhead associated with meetings
- You are now on the hook for what you say (probably a good thing)
- Copy answers into another document to save them

Outline

- ~~Using GenAI with Transcripts~~
  - ~~Basic Concepts~~
  - ~~Use Cases~~
    - ~~Single Meeting~~
    - ~~Multiple Meetings~~
    - ~~Interview~~
    - ~~Classroom~~
  - ~~Pitfalls~~
  - ~~Recap~~
- RAG Implementation
  - Concept / Motivation
  - Overview of Implementation
  - Chunk
  - Generate Embeddings
  - Retrieve
  - Run Prompt
  - Advanced RAG (time permitting)

Be sure to jump in and ask questions or share your experience!

RAG Implementation: Concept / Motivation

RAG = Retrieval Augmented Generation

Basic Idea:

- Retrieve pertinent sections of large or numerous documents
- Insert those chunks into a prompt and generate response

Motivation:

- Fit information into context (limited context length)
- Cost (using context costs $)
- Quality of Results (extraneous information can confuse LLMs)

RAG Implementation: Overview of Implementation

Pre-Process Your Documents:

1. Chunk: Divide the document(s) into small blocks and clean up if necessary
2. Generate Embeddings: For each chunk, use an embedding model to generate an embedding and store the embedding and original text

Run a Prompt:

3. Retrieve: Convert the prompt into an embedding and search for the most closely related chunks
4. Generate: Run the prompt with the retrieved chunks and return a response

RAG Implementation: Chunk

```
meeting-id (YYYY-MM-DD HH:MM GMT)
Attendees: Person A, Person B
Transcript
00:00:00
Person A: Some text.
Person B: More text.
00:05:00
Person A: Text continues.
…
```

```
[
    "Person A (YYYY-MM-DD HH:MM GMT): Some text.",
    "Person B (YYYY-MM-DD HH:MM GMT): More text.",
    ...
]
```

RAG Implementation: Generate Embeddings

```
[

        "Person A (YYYY-MM-DD HH:MM GMT): Some text.",

        "Person B (YYYY-MM-DD HH:MM GMT): More text."

]
```

```
embeddings = []
for document in data:
    response = openai.embeddings.create(
        input=document,
        model="text-embedding-3-small"
    )
    embeddings.append({
                "document": document,
                "embedding": response.data[0].embedding
    })
```

```
[

        {

                “document”: "Person A (YYYY-MM-DD HH:MM GMT): Some text.",

                “embedding”: [0.04146402329206467, 0.0473700575530529, 0.020184028893709183, 0.021964972838759422, (~1500 more)]

        },

        {

                “document”: "Person B (YYYY-MM-DD HH:MM GMT): More text.",

                “embedding”: [-0.008896579965949059, -0.013088001869618893, 0.01451504323631525, 0.005508377216756344, (~1500 more)]

        },

]
```

RAG Implementation: Retrieve


```
[

 {

   “document”: "Person A (YYYY-MM-DD HH:MM GMT): Some text.",

   “embedding”: [0.04146402329206467, 0.0473700575530529,
0.020184028893709183, 0.021964972838759422, (~1500 more)]

  },
```


What did Steve say about database migration?

```python
# Generate an embedding for the prompt
response = openai.embeddings.create(
        input=prompt,
        model="text-embedding-3-small"
)
prompt_embedding = response.data[0].embedding

# Calculate cosine similarity for each transcript
results = []
for transcript in transcripts_data:
        transcript_embedding = transcript["embedding"]
        distance = spatial.distance.cosine(prompt_embedding, transcript_embedding)
        results.append({"document": transcript["document"], "distance": distance})

# Sort the results by cosine similarity
results.sort(key=lambda x: x["distance"])
```


```
[
        {
        "document": "Steve Goldsmith (2024-09-23 14:40 ): I'm looking at those migration, The downside of sequel, the upside of having schemas
generally, rigid schemas is then you're guaranteed to know that you're data follows this format. We're not jus>
        "distance": 0.418447816854445
        },
        {
        "document": "Steve Goldsmith (2024-09-23 14:20 ): where you basically have some window, We're gonna do this. When there's very few users that
are using the site and then, The simplest version is really just like, Okay, we have if the database is>
        "distance": 0.42410594170096827
        },
```

RAG Implementation: Run Prompt

```
[
        {
                "document": "Steve Goldsmith (2024-09-23 14:40 ): I'm
looking at those migration, The downside of sequel,>",
                "distance": 0.418447816854445
        },
        {
                "document": "Steve Goldsmith (2024-09-23 14:20 ):
where you basically have some window, We're gonna do th>",
                "distance": 0.42410594170096827
        }
]
```

What did Steve say about database migration?

```
# Format the documents into a single string
formatted_documents = ""
for doc in retrieved_documents:
        formatted_documents += f"{doc['document']}
({doc['distance']})\n\n"

# Create the user message
user_message = f"{formatted_documents}\n\n{user_prompt}"

# Create the chat completion request
completion = openai.chat.completions.create(
        model="gpt-4o-mini",
        messages=[
        {
                "role": "system",
                "content": "Answer the user's prompt based on the
provided documents.",
        },
        {"role": "user", "content": user_message},
        ],
)
```

Steve Goldsmith shared several insights about database migration emphasizing the complexity, challenges, and strategies involved in the process. Here are the key points he mentioned:

1. **Schema Rigidness and Challenges**: While rigid schemas help ensure data consistency and defined formats, they also create challenges during migration, especially as production databases scale. A consistent approach to handling transactions is cruci>

2. **Migration Strategies**: He discussed different approaches to migration, including performing migrations during low-traffic periods and the idea of full migrations when the database is small enough. For larger databases, he suggested that a complete>

3. **Testing Migrations**: Before executing migrations, it's important to test scripts in isolated development environments. This helps to simulate the migration process with a subset of data to identify potential issues.

RAG Implementation: Advanced RAG

- Pre Process Query (Turn prompt into statement)
- Post Process Document
- Embedding Database (Chroma)
- More Advanced Search (Graph Search)
- Iterative Search (keep trying)
- Recursive Search (drill down on details)
- Ad-Hoc vs Ahead of Time Embedding Generation
- Other Use Cases: Web Search, Company Documents, APIs, Document Databases

https://github.com/goldsmithai/djdc44