jxnl.co

@jxnlco

Systematically Improving RAG Applications

Session 5

Map: Navigating Multimodal RAG

Jason Liu

Progress so far

Sessions 1, 2, 3

- ☐ Generate synthetic data
- ☐ Generate fast evaluations
- ☐ Focus on improving representations
- ☐ Implement product improvements to gain user trust and satisfaction

Session 4

- ☐ Build models to segment the users and queries
- ☐ Prioritize segments and new functionality based on impact, impact, volume of queries and probability of success

Agenda

Multiple indices for RAG applications

- Two Classes of Improvements
- Important metrics for search indices

Handling entity-specific search indices

Sneak peak of next session

Router?

Building a local vs global model

- When segments exist in a population, the assumption we make is that local decision making will outperform global decision making
- Instead of building on search index we might want to split up the problem space and solve each one locally
- Building specific indices allows use to divide the labor into isolated tests that we combine later
- As you learn more about user needs, adding a new index is easier than rebuilding existing systems

Router?

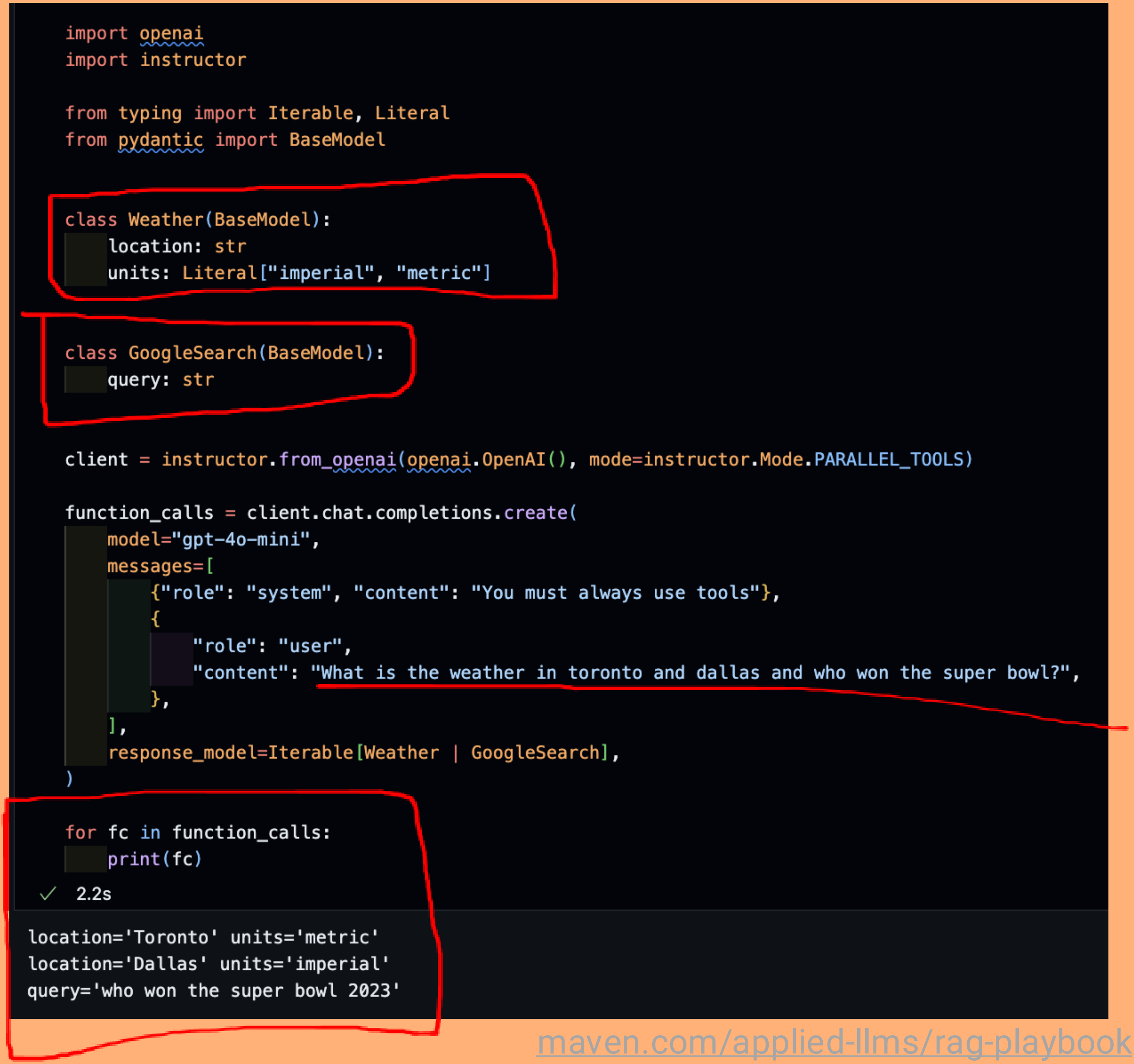
Building a local vs global model

- When segments exist in a population, the assumption we make is that local decision making will outperform global decision making
- Instead of building on search index we might want to split up the problem space and solve each one locally
- Building specific indices allows use to divide the labor into isolated tests that we combine later
- As you learn more about user needs, adding a new index is easier than rebuilding existing systems

Example: Building specific search indices for a hardware store

- **Lexical Search**
  - How does the XZF2000 compare to the XZF3000?
- **Semantic search**
  - What do people tend to think about this saw's durability?
- **Search through table data** extracted from manufacturer provided specs
  - How much does it weigh?
  - What's the latest version?

How to build a simple query router



```python
import openai
import instructor

from typing import Iterable, Literal
from pydantic import BaseModel

class Weather(BaseModel):
    location: str
    units: Literal["imperial", "metric"]

class GoogleSearch(BaseModel):
    query: str

client = instructor.from_openai(openai.OpenAI(), mode=instructor.Mode.PARALLEL_TOOLS)

function_calls = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        {"role": "system", "content": "You must always use tools"},
        {
            "role": "user",
            "content": "What is the weather in toronto and dallas and who won the super bowl?",
        },
    ],
    response_model=Iterable[Weather | GoogleSearch],
)

for fc in function_calls:
    print(fc)
```

✓ 2.2s

```
location='Toronto' units='metric'
location='Dallas' units='imperial'
query='who won the super bowl 2023'
```

@jxnlco

maven.com/applied-llms/rag-playbook

How do we join and improve our multiple search indices?

❶ **Short-term**: Concatenate results, apply a reranker, stuff everything into context

❷ **Long-term**: Train a ranking model using relevancy and citation data as you get it (to take in various different scores and sources of information)

How do we join and improve our multiple search indices?

❶ **Short-term**: Concatenate results, apply a reranker, stuff everything into context

❷ **Long-term**: Train a ranking model using relevancy and citation data as you get it (to take in various different scores and sources of information)

$$\textbf{Score} = (\, a \times \text{cosine\_score} \,) + (\, b \times \text{cohere\_score} \,) + (\, c \times \cdots \,)$$

Tests:

```
Assert eval.chunk_id in search(query).chunk_ids
```

This session, we'll focus on individual tools, followed by routing next session.

However, these topics can be developed concurrently due to their separation of concerns—much like front-end and back-end development.

Agenda

Two approaches to candidate selection:

Process

- Expose existing structured data as additional filtering criteria on the same "chunk" (e.g., document, image)
- Expose new structured datasets

Approach 1: Extracting Metadata

Two approaches to candidate selection:

Process

- Expose existing structured data as additional filtering criteria on the same “chunk” (e.g., document, image)
- Expose new structured datasets

Approach 1: Extracting Metadata

Examples

Finance:

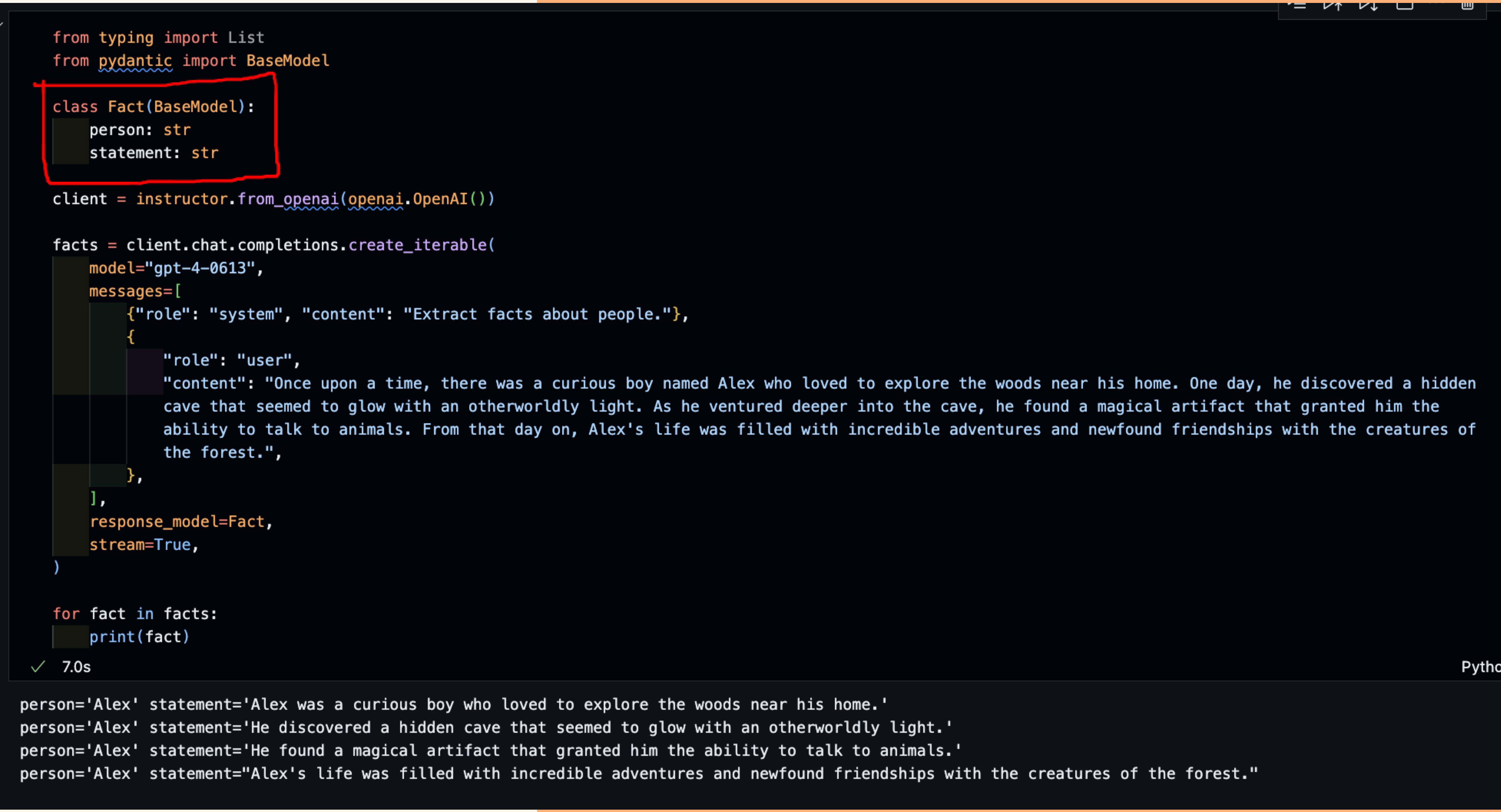
- Map calendar months with industry-specific fiscal years to search financial data in a more targeted way

Contracts:

- Process and create labels for a ‘contract signed’ boolean, for payment dates, and payment terms; return doc_id

Research call transcripts:

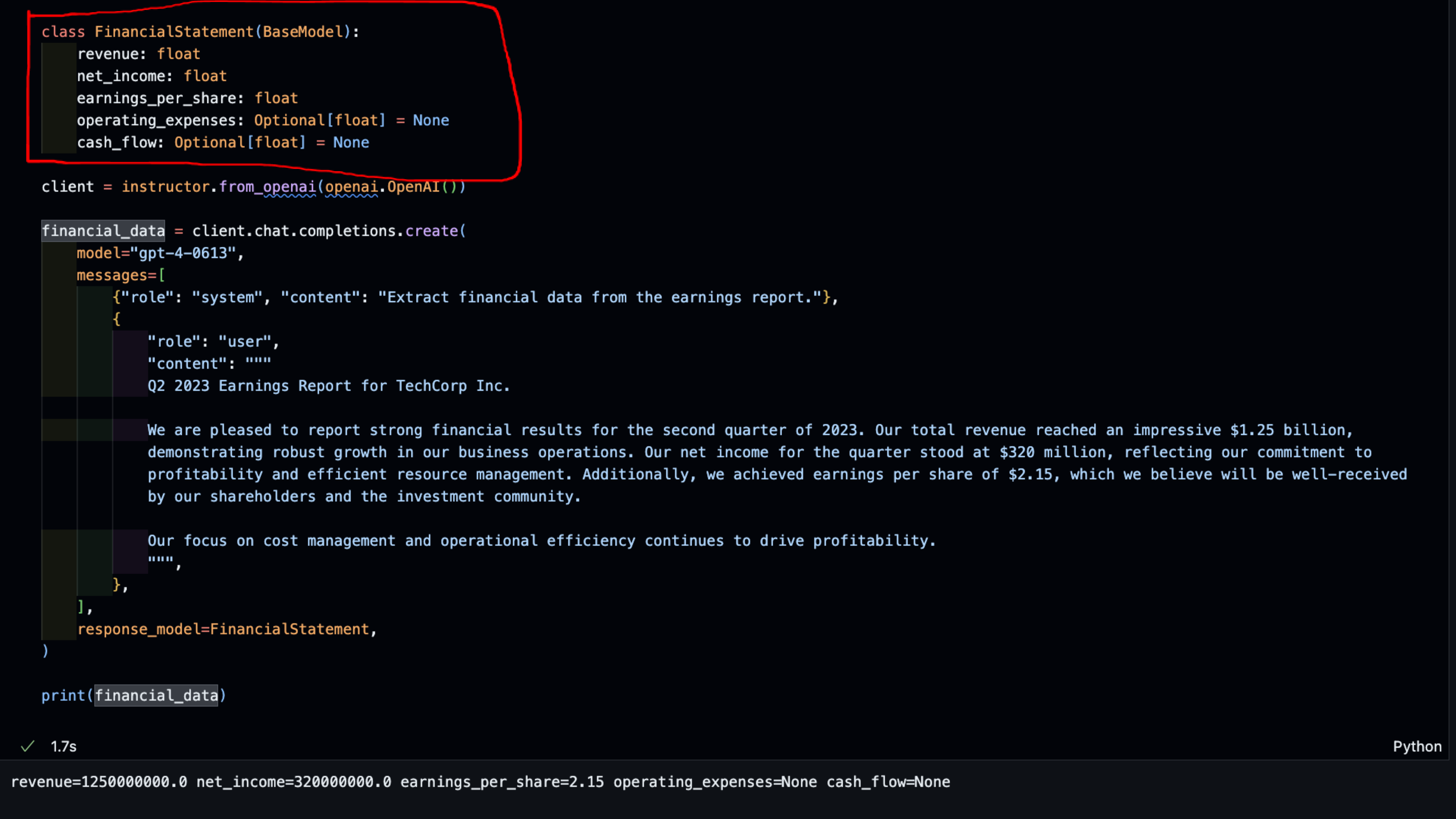
- Classify by type (e.g., job interview, stand-up, design review)
- Extract type-specific metadata to enable additional searches against different reading types



```python
from typing import List
from pydantic import BaseModel

class Fact(BaseModel):
    person: str
    statement: str

client = instructor.from_openai(openai.OpenAI())

facts = client.chat.completions.create_iterable(
    model="gpt-4-0613",
    messages=[
        {"role": "system", "content": "Extract facts about people."},
        {
            "role": "user",
            "content": "Once upon a time, there was a curious boy named Alex who loved to explore the woods near his home. One day, he discovered a hidden
            cave that seemed to glow with an otherworldly light. As he ventured deeper into the cave, he found a magical artifact that granted him the
            ability to talk to animals. From that day on, Alex's life was filled with incredible adventures and newfound friendships with the creatures of
            the forest.",
        },
    ],
    response_model=Fact,
    stream=True,
)

for fact in facts:
    print(fact)
```

✓ 7.0s Pytho

```
person='Alex' statement='Alex was a curious boy who loved to explore the woods near his home.'
person='Alex' statement='He discovered a hidden cave that seemed to glow with an otherworldly light.'
person='Alex' statement='He found a magical artifact that granted him the ability to talk to animals.'
person='Alex' statement="Alex's life was filled with incredible adventures and newfound friendships with the creatures of the forest."
```



```python
class FinancialStatement(BaseModel):
    revenue: float
    net_income: float
    earnings_per_share: float
    operating_expenses: Optional[float] = None
    cash_flow: Optional[float] = None

client = instructor.from_openai(openai.OpenAI())

financial_data = client.chat.completions.create(
    model="gpt-4-0613",
    messages=[
        {"role": "system", "content": "Extract financial data from the earnings report."},
        {
            "role": "user",
            "content": """
            Q2 2023 Earnings Report for TechCorp Inc.

            We are pleased to report strong financial results for the second quarter of 2023. Our total revenue reached an impressive $1.25 billion,
            demonstrating robust growth in our business operations. Our net income for the quarter stood at $320 million, reflecting our commitment to
            profitability and efficient resource management. Additionally, we achieved earnings per share of $2.15, which we believe will be well-received
            by our shareholders and the investment community.

            Our focus on cost management and operational efficiency continues to drive profitability.
            """,
        },
    ],
    response_model=FinancialStatement,
)

print(financial_data)
```

✓ 1.7s Python

```
revenue=1250000000.0 net_income=320000000.0 earnings_per_share=2.15 operating_expenses=None cash_flow=None
```

Two approaches to candidate selection:

Process

1. Use a language model to generate a summary or synthetic text chunk (optimized for recall)
2. During the text search (e.g., BM25, semantic search), search for the synthetic text chunk and return:
   - Source Content
   - Synthetic Chunk
3. Become pointers to source data

Two approaches to candidate selection:

Process

1. Use a language model to generate a summary or synthetic text chunk (optimized for recall)
2. During the text search (e.g., BM25, semantic search), search for the synthetic text chunk and return:
   - Source Content
   - Synthetic Chunk
3. Become pointers to source data

Approach 2: Generate synthetic data text chunks

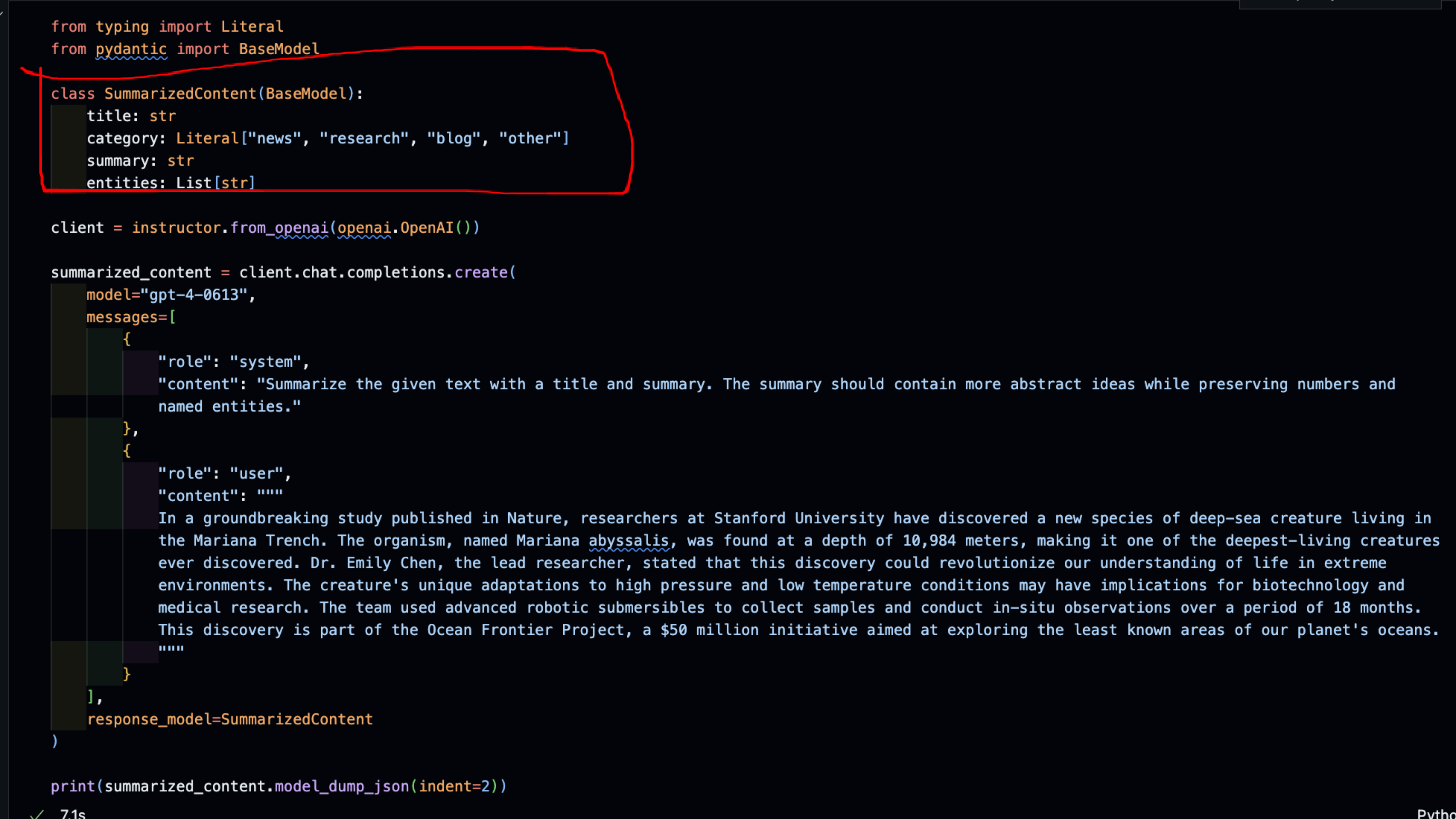
Examples

Complex text data (e.g., interview or research call transcripts):

- Give the LLM a document and ask it to generate an FAQ
- Embed the FAQ as question answer pairs

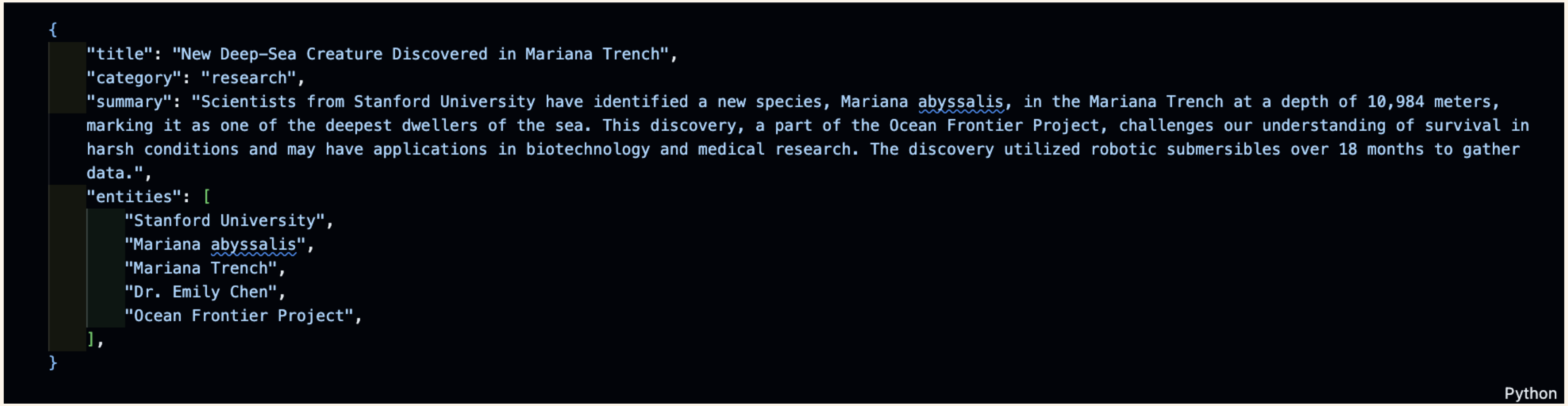
Image data:

- Give the LLM images and ask it to generate detailed descriptions based on sample user queries. For example:
  - Count the number of cones in the image
  - Describe the mood in the image
  - Describe the type of shot (e.g., fish-eye, close-up, wide-angle)
- Embed the specific image descriptions



```python
from typing import Literal
from pydantic import BaseModel

class SummarizedContent(BaseModel):
    title: str
    category: Literal["news", "research", "blog", "other"]
    summary: str
    entities: List[str]

client = instructor.from_openai(openai.OpenAI())

summarized_content = client.chat.completions.create(
    model="gpt-4-0613",
    messages=[
        {
            "role": "system",
            "content": "Summarize the given text with a title and summary. The summary should contain more abstract ideas while preserving numbers and
            named entities."
        },
        {
            "role": "user",
            "content": """
            In a groundbreaking study published in Nature, researchers at Stanford University have discovered a new species of deep-sea creature living in
            the Mariana Trench. The organism, named Mariana abyssalis, was found at a depth of 10,984 meters, making it one of the deepest-living creatures
            ever discovered. Dr. Emily Chen, the lead researcher, stated that this discovery could revolutionize our understanding of life in extreme
            environments. The creature's unique adaptations to high pressure and low temperature conditions may have implications for biotechnology and
            medical research. The team used advanced robotic submersibles to collect samples and conduct in-situ observations over a period of 18 months.
            This discovery is part of the Ocean Frontier Project, a $50 million initiative aimed at exploring the least known areas of our planet's oceans.
            """
        }
    ],
    response_model=SummarizedContent
)

print(summarized_content.model_dump_json(indent=2))
```

✓ 7.1s Pytho



```
{
    "title": "New Deep-Sea Creature Discovered in Mariana Trench",
    "category": "research",
    "summary": "Scientists from Stanford University have identified a new species, Mariana abyssalis, in the Mariana Trench at a depth of 10,984 meters,
    marking it as one of the deepest dwellers of the sea. This discovery, a part of the Ocean Frontier Project, challenges our understanding of survival in
    harsh conditions and may have applications in biotechnology and medical research. The discovery utilized robotic submersibles over 18 months to gather
    data.",
    "entities": [
        "Stanford University",
        "Mariana abyssalis",
        "Mariana Trench",
        "Dr. Emily Chen",
        "Ocean Frontier Project",
    ],
}
```

If you look closely it’s just extracting structured output, either to put into a DB, and some fields are indexed for search directly!

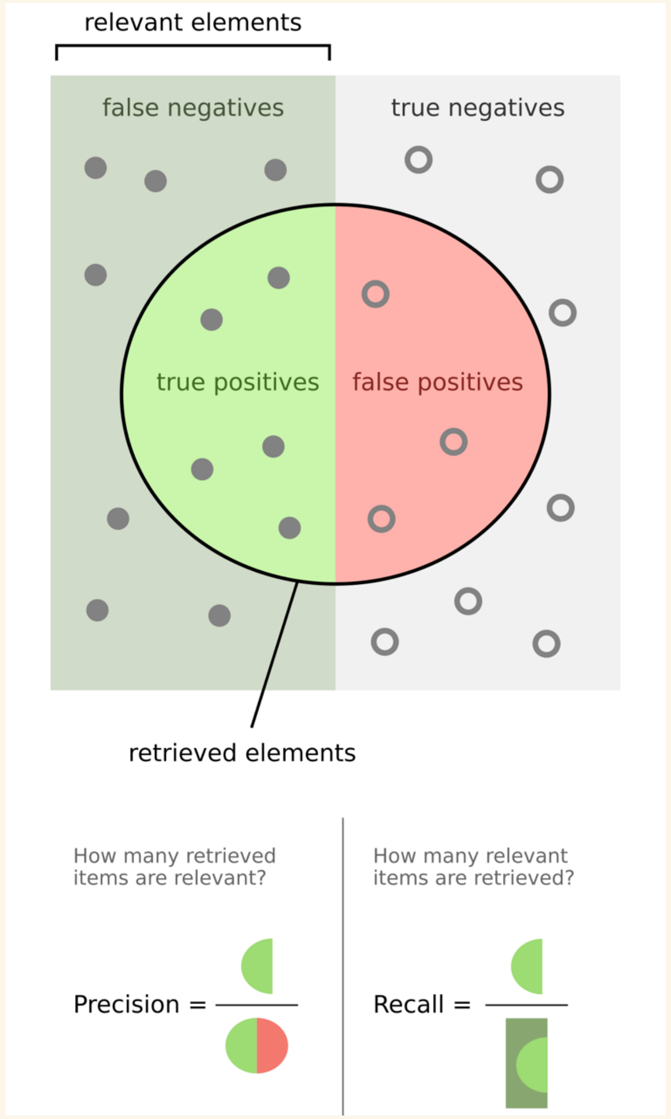
It’s just a materialized view + AI

Agenda

In session 1, we had a single global index, and we generated synthetic data to test the performance of retrieval

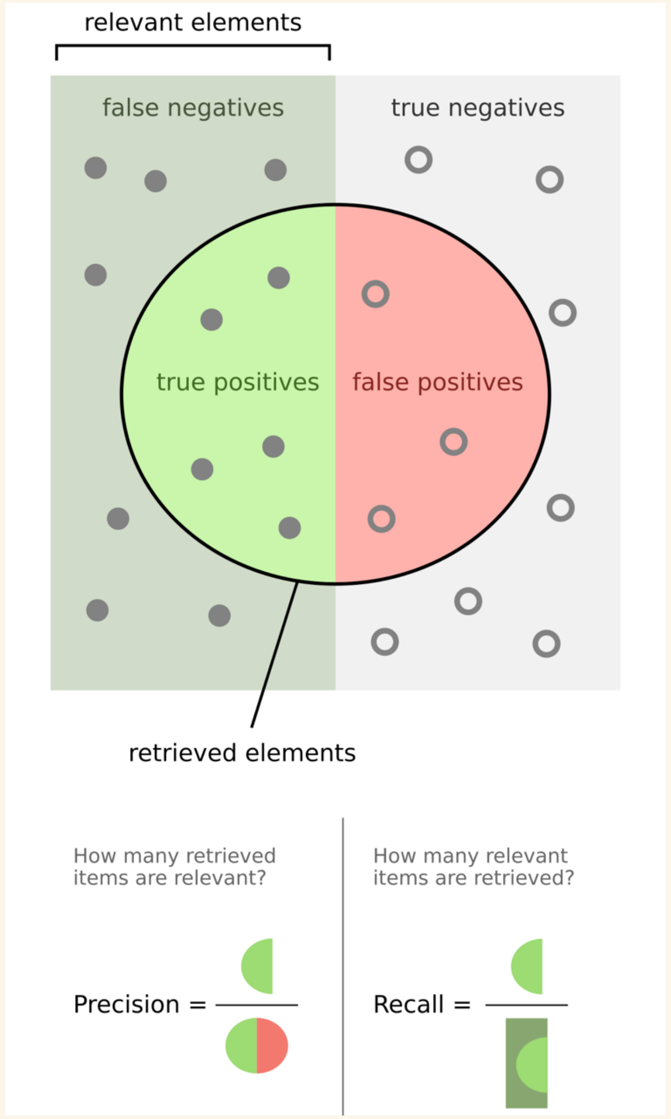
By session 4, we realized we might need local solutions for segments we uncover

Precision and Recall Again!



	Recall	Precision
Definition	*% of relevant search methods that are successfully used*	*% of search methods used relevant to the query*
Formula	$\frac{\text{(Relevant retrieved search methods)}}{\text{(Total relevant search methods)}}$	$\frac{\text{(Relevant retrieved search methods)}}{\text{(Total retrieved search methods)}}$

Precision and Recall Again!



	Recall	Precision
Definition	*% of relevant search methods that are successfully used*	*% of search methods used relevant to the query*
Formula	$\frac{\text{(Relevant retrieved search methods)}}{\text{(Total relevant search methods)}}$	$\frac{\text{(Relevant retrieved search methods)}}{\text{(Total retrieved search methods)}}$
Why is this important to measure?	• High recall means the system finds most of the relevant search methods	• High precision means the system uses mostly relevant search methods
Example	• LLM is able to identify most of the relevant search methods	• LLM chooses to call the correct (2/2) relevant search methods

Applying the Chain Rule

$$P(\text{Correct chunk found}) = \boxed{P(\text{Correct chunk found} \mid \text{correct retriever})} \times \boxed{P(\text{correct retriever})}$$

Applying the Chain Rule

$$P(\text{Correct chunk found}) = \boxed{P(\text{Correct chunk found} \mid \text{correct retriever})} \times \boxed{P(\text{correct retriever})}$$

Key takeaway:

Identify the limiting factor of the system

A) Not using the right retriever

B) The retriever itself is bad

Why is this important?

- This gives you a process to determine what needs to improve across segments
- This might feel boring, and the work required to collect these data points will be challenging. But it is very worth doing for long term systematic improvements to your RAG application

Agenda

Multiple indices for RAG applications

Handling entity-specific search indices

- **Documents**
- Images and answering questions about images
- Tables (SQL, Excel)

Sneak peak of next session

How to handle document search – at a glance

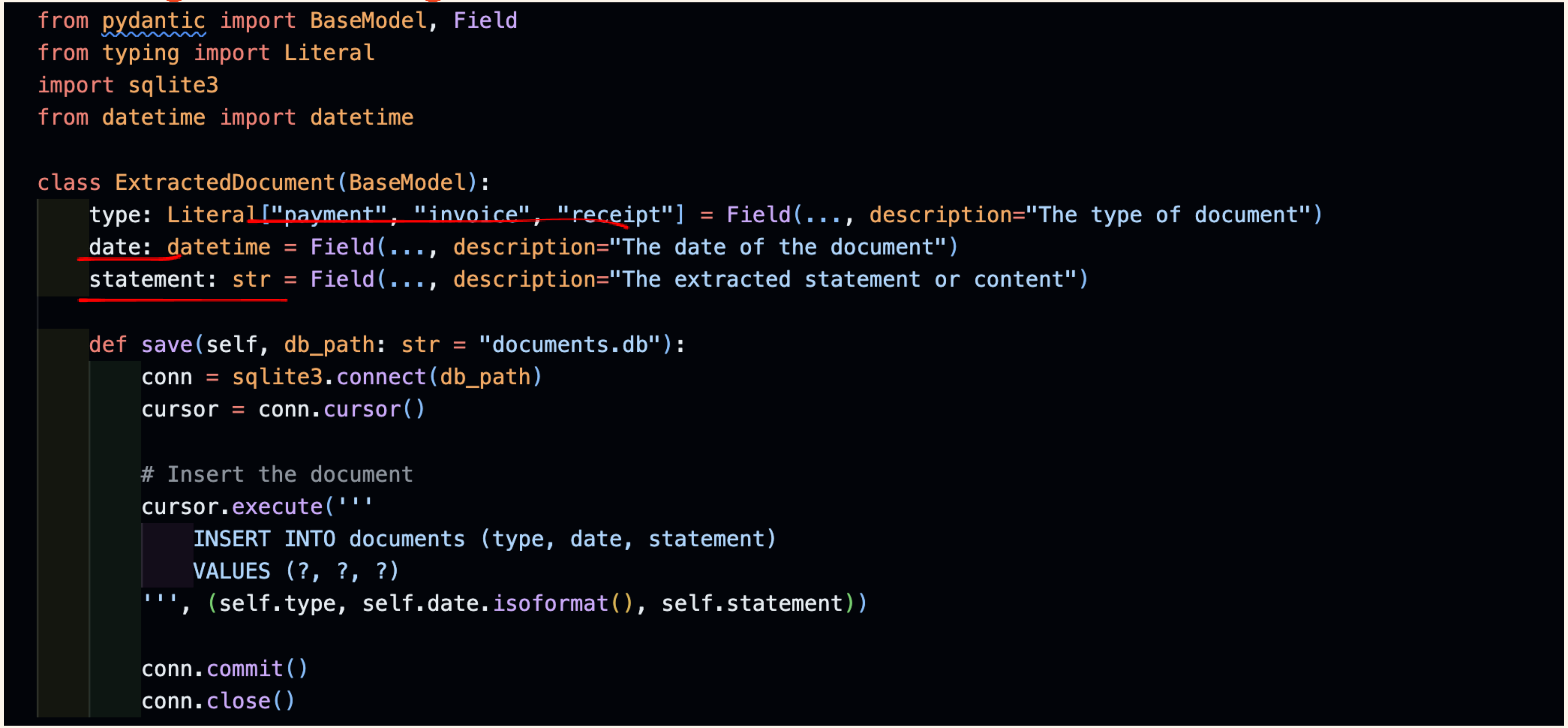
- A Chunking, Metadata, Contextual Retrieval
- B Lexical and Semantic Search
- C Passages and Rerankers
- D Documents and Long Context
- E Summarization and Extraction

How to handle document search – at a glance

- A Chunking, Metadata, Contextual Retrieval
- B Lexical and Semantic Search
- C Passages and Rerankers
- D Documents and Long Context
- E Summarization and Extraction

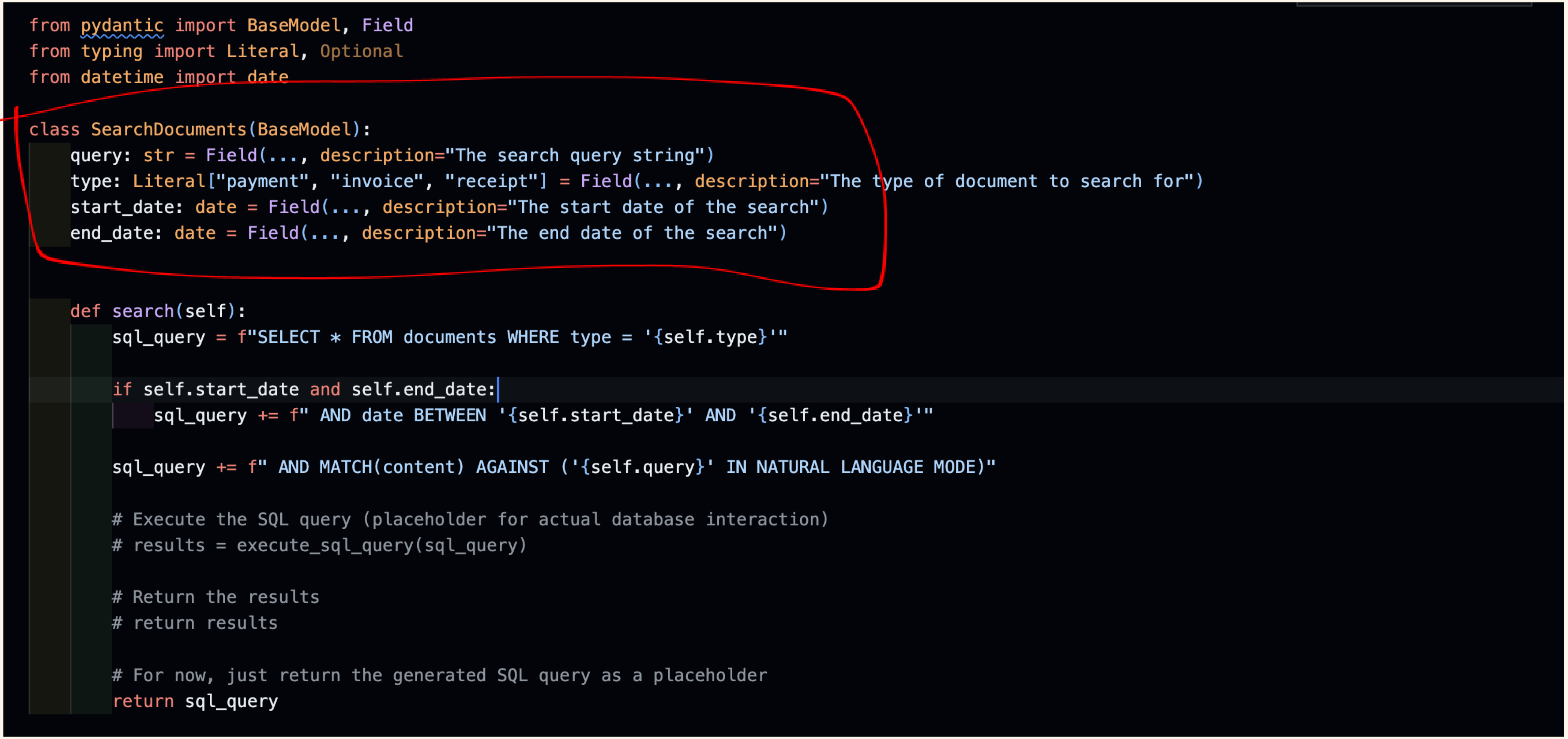
1. *Continue to blend and test multiple parallel search methods and fuse results*
2. *As a result, retrieval might return summaries, documents, and chunks*
3. *As long as data is represented correctly, just tell the LLM that it might receive a mixture of summaries, documents, and chunks. This will elicit a higher quality result*

Extracting and Inserting data



```python
from pydantic import BaseModel, Field
from typing import Literal
import sqlite3
from datetime import datetime

class ExtractedDocument(BaseModel):
    type: Literal["payment", "invoice", "receipt"] = Field(..., description="The type of document")
    date: datetime = Field(..., description="The date of the document")
    statement: str = Field(..., description="The extracted statement or content")

    def save(self, db_path: str = "documents.db"):
        conn = sqlite3.connect(db_path)
        cursor = conn.cursor()

        # Insert the document
        cursor.execute('''
            INSERT INTO documents (type, date, statement)
            VALUES (?, ?, ?)
        ''', (self.type, self.date.isoformat(), self.statement))

        conn.commit()
        conn.close()
```

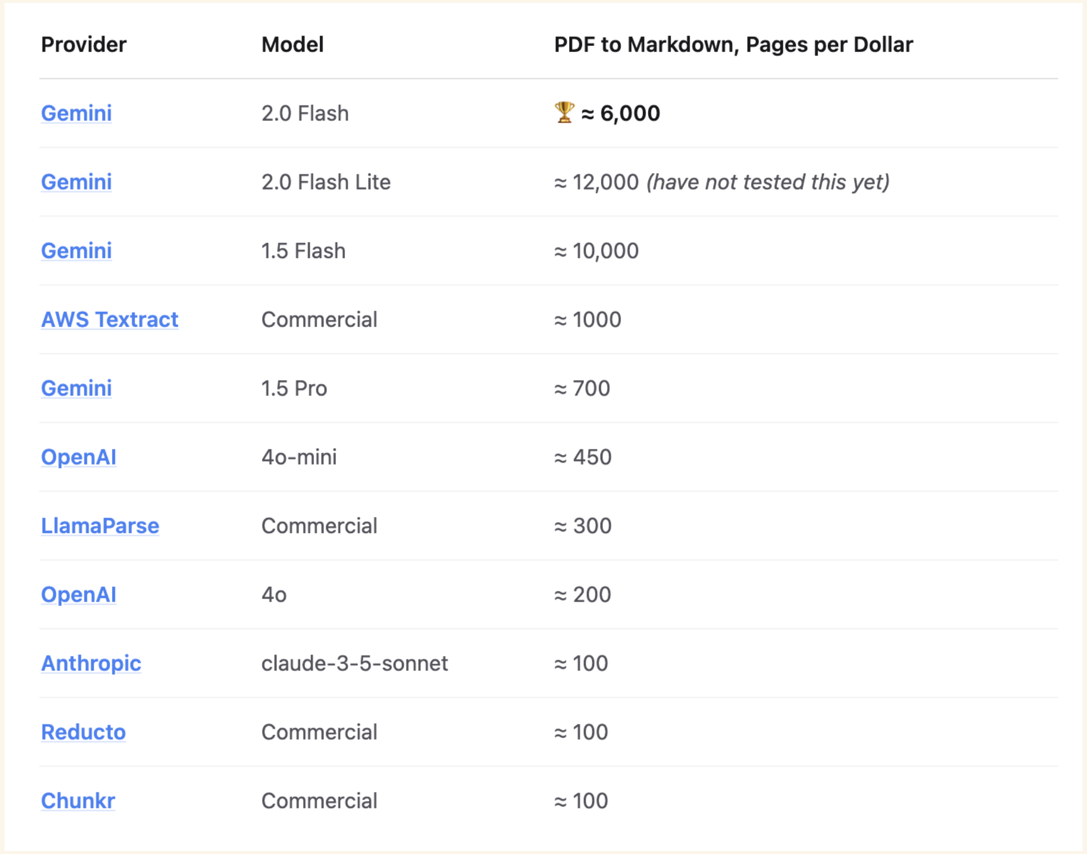
Searching Data



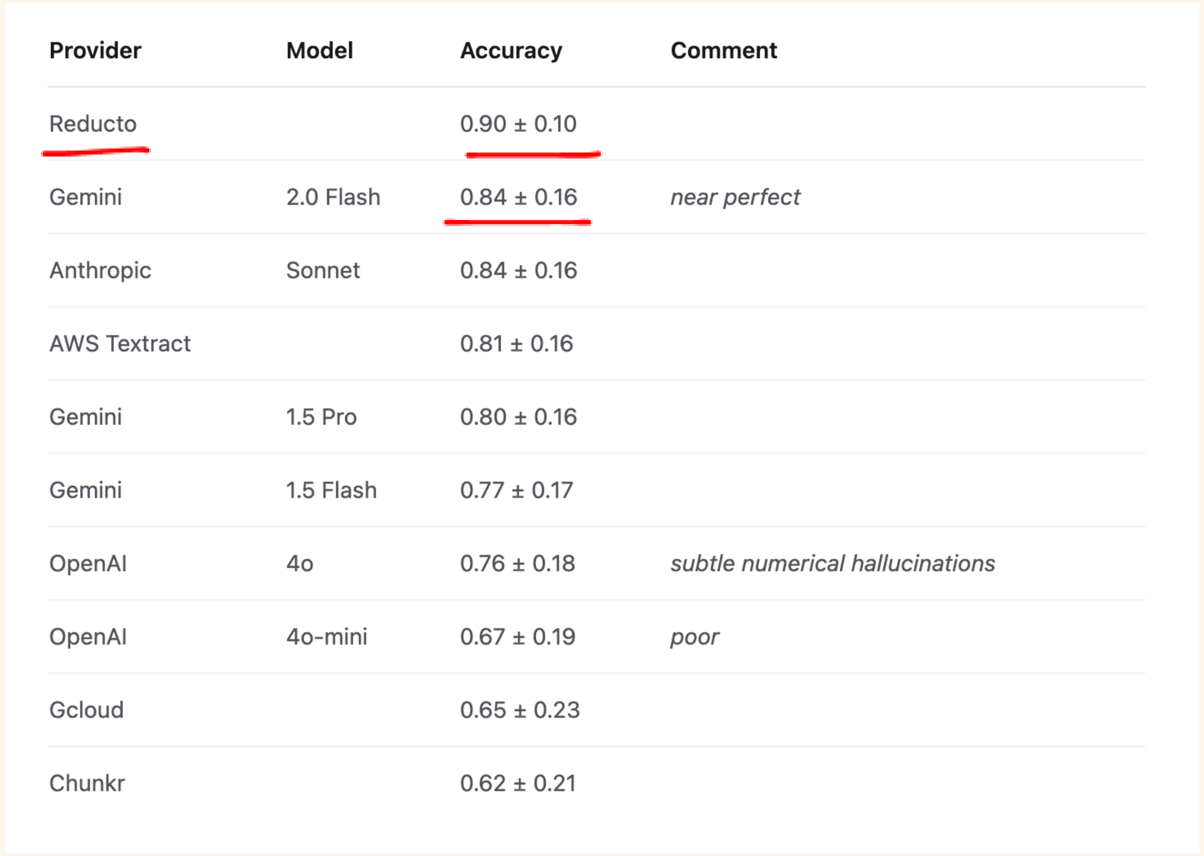
```python
from pydantic import BaseModel, Field
from typing import Literal, Optional
from datetime import date

class SearchDocuments(BaseModel):
    query: str = Field(..., description="The search query string")
    type: Literal["payment", "invoice", "receipt"] = Field(..., description="The type of document to search for")
    start_date: date = Field(..., description="The start date of the search")
    end_date: date = Field(..., description="The end date of the search")

    def search(self):
        sql_query = f"SELECT * FROM documents WHERE type = '{self.type}'"

        if self.start_date and self.end_date:
            sql_query += f" AND date BETWEEN '{self.start_date}' AND '{self.end_date}'"

        sql_query += f" AND MATCH(content) AGAINST ('{self.query}' IN NATURAL LANGUAGE MODE)"

        # Execute the SQL query (placeholder for actual database interaction)
        # results = execute_sql_query(sql_query)

        # Return the results
        # return results

        # For now, just return the generated SQL query as a placeholder
        return sql_query
```

PDF parsing is only gotten better

https://www.sergey.fyi/articles/gemini-flash-2



Provider	Model	PDF to Markdown, Pages per Dollar
Gemini	2.0 Flash	🏆 **≈ 6,000**
Gemini	2.0 Flash Lite	≈ 12,000 *(have not tested this yet)*
Gemini	1.5 Flash	≈ 10,000
AWS Textract	Commercial	≈ 1000
Gemini	1.5 Pro	≈ 700
OpenAI	4o-mini	≈ 450
LlamaParse	Commercial	≈ 300
OpenAI	4o	≈ 200
Anthropic	claude-3-5-sonnet	≈ 100
Reducto	Commercial	≈ 100
Chunkr	Commercial	≈ 100

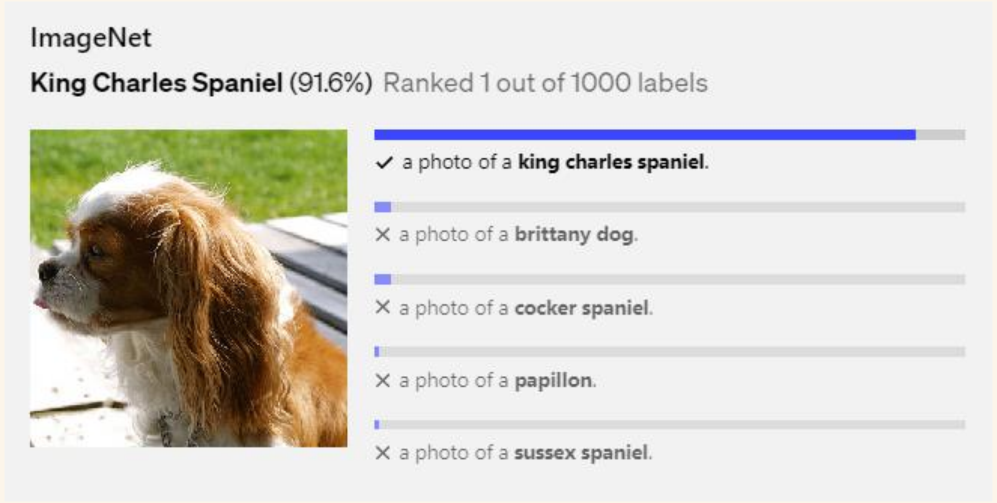


Provider	Model	Accuracy	Comment
Reducto		0.90 ± 0.10	
Gemini	2.0 Flash	0.84 ± 0.16	*near perfect*
Anthropic	Sonnet	0.84 ± 0.16	
AWS Textract		0.81 ± 0.16	
Gemini	1.5 Pro	0.80 ± 0.16	
Gemini	1.5 Flash	0.77 ± 0.17	
OpenAI	4o	0.76 ± 0.18	*subtle numerical hallucinations*
OpenAI	4o-mini	0.67 ± 0.19	*poor*
Gcloud		0.65 ± 0.23	
Chunkr		0.62 ± 0.21	

That said my bet on where the space is going is going to be aroud generation location specific citations

Agenda

Remember: Visual language models are trained on captioning data



ImageNet

King Charles Spaniel (91.6%) Ranked 1 out of 1000 labels

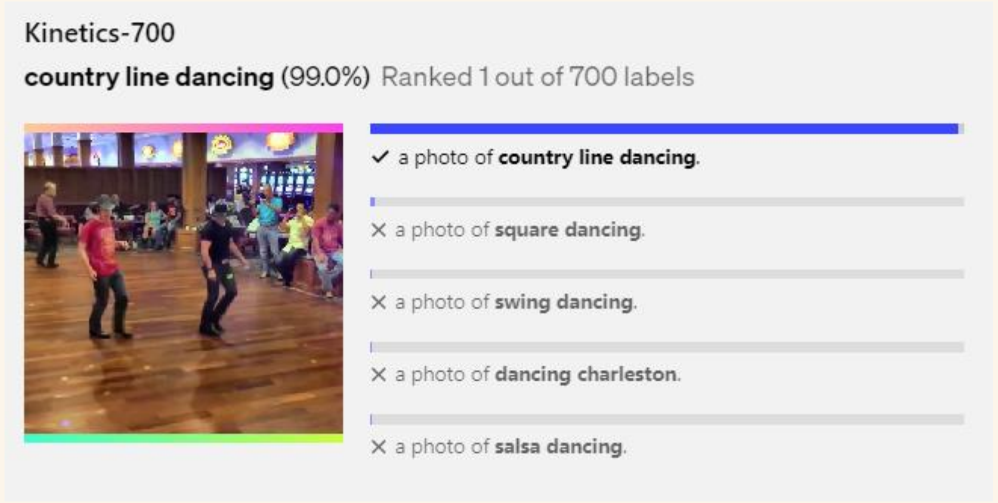
✓ a photo of a **king charles spaniel**.

✗ a photo of a **brittany dog**.

✗ a photo of a **cocker spaniel**.

✗ a photo of a **papillon**.

✗ a photo of a **sussex spaniel**.



Kinetics-700

country line dancing (99.0%) Ranked 1 out of 700 labels

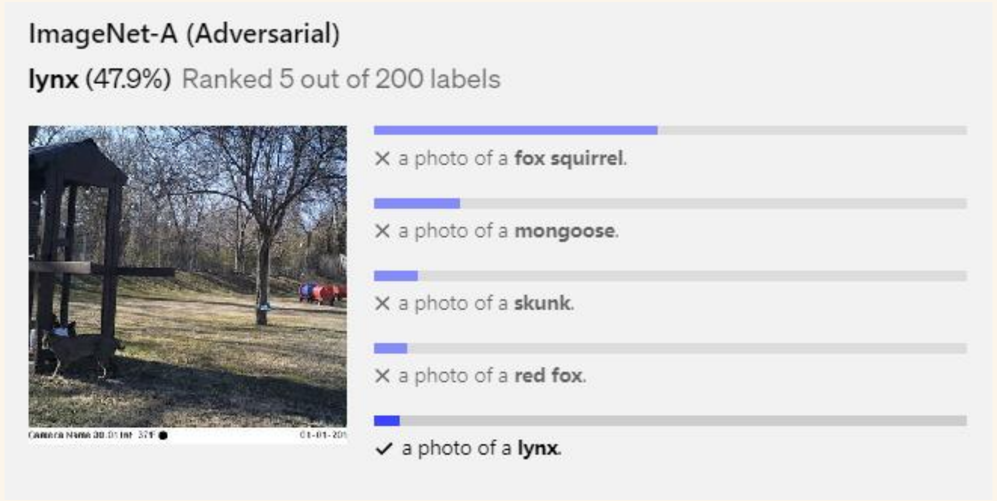
✓ a photo of **country line dancing**.

✗ a photo of **square dancing**.

✗ a photo of **swing dancing**.

✗ a photo of **dancing charleston**.

✗ a photo of **salsa dancing**.



ImageNet-A (Adversarial)

lynx (47.9%) Ranked 5 out of 200 labels

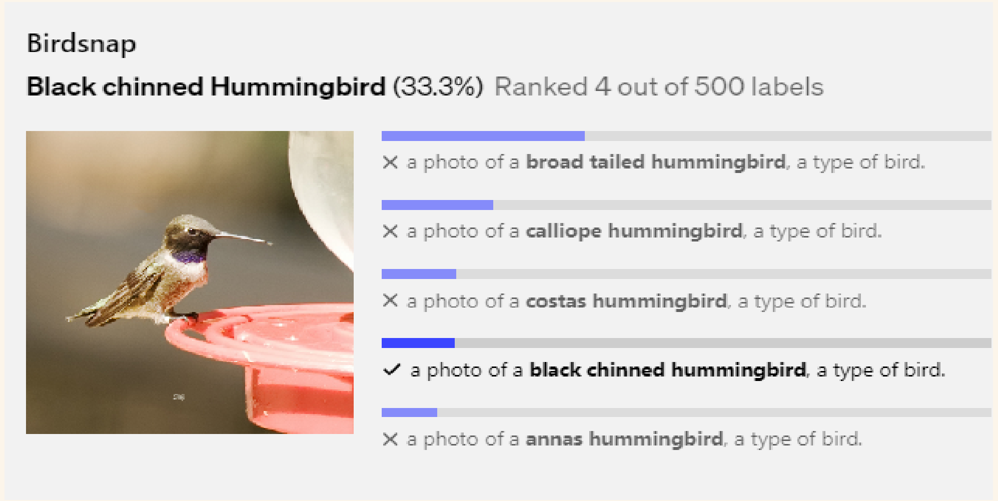
✗ a photo of a **fox squirrel**.

✗ a photo of a **mongoose**.

✗ a photo of a **skunk**.

✗ a photo of a **red fox**.

✓ a photo of a **lynx**.



Birdsnap

Black chinned Hummingbird (33.3%) Ranked 4 out of 500 labels

✗ a photo of a **broad tailed hummingbird**, a type of bird.

✗ a photo of a **calliope hummingbird**, a type of bird.

✗ a photo of a **costas hummingbird**, a type of bird.

✓ a photo of a **black chinned hummingbird**, a type of bird.

✗ a photo of a **annas hummingbird**, a type of bird.

Multimodal Embeddings might make sense for costs, but complex question answering may results in low recall for the same reason as query/passages.

How can we be certain about embeddings matching?

Prompting Descriptions (read: summarization)

Don't simply ask, "What's in this image?"

Be specific and incorporate the questions you anticipate being asked about a system.

- This is an image of two people
- This is an image of two people arguing
- This is an image of Name1, Name2 in a mysterious foggy scene arguing over the dinner table and Name2 has a knife

Prompting Descriptions (read: summarization)

Don't simply ask, "What's in this image?"

Be specific and incorporate the questions you anticipate being asked about a system.

- This is an Image of two people
- This is an image of two people arguing
- This is an image of Name1, Name 2 in a mysterious foggy scene arguing over the dinner table and Name2 has a knife

However, if your search data does not look like it will be or does search across captions, try these search techniques:

1. **Leverage Chain of Thought to 'reason' out the captioning data**
2. **Augment context when extracting images, attach OCR, or even near by passages**
3. **Leverage Bounding boxes and Structured Extraction from Images w/**

Often it can be much better to extract summaries or even structured data out of a visual model

1 Monologues and Additional Context

You are an AI assistant tasked with generating a comprehensive description of an image that can be used for semantic and lexical search. Follow these steps to create your description:

1. If available, review any OCR content extracted from the image:
<ocr_content>
{{OCR_CONTENT}}
</ocr_content

2. Begin by providing a detailed description of the image. Focus on visual elements, colors, composition, and any text or recognizable objects. Use <description> tags for this section.

3. Next, reason about the image. Consider its context, purpose, and potential significance. Use <reasoning> tags for this section.

4. Think about what kinds of questions this image could answer. Consider various aspects such as subject matter, historical context, artistic style, or practical applications. Use <potential_questions> tags for this section.

5. Finally, create a concise summary that incorporates key elements from your description, reasoning, and potential questions. This summary should be optimized for semantic and lexical search. Use <summary> tags for this section.

2

Augment context when extracting images

Accept the null hypothesis that things won't improve and run experiments to actually find what will impact the system

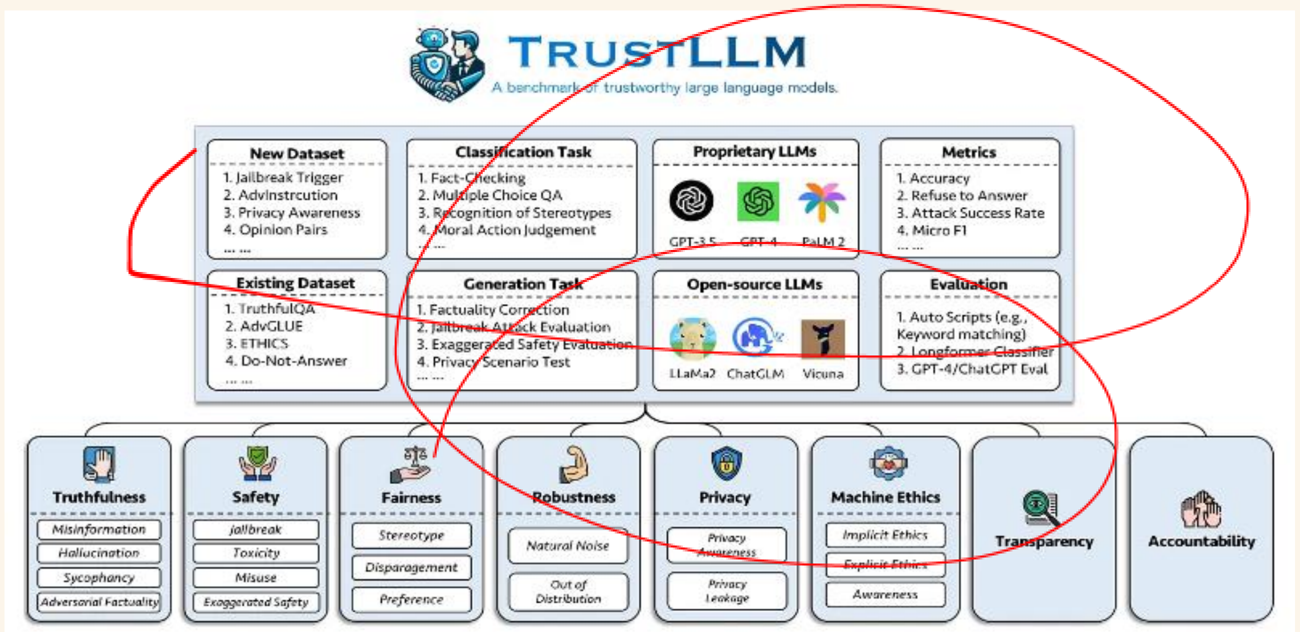


Figure 1. The design of benchmark in TRUSTLLM. Building upon the evaluation principles in prior research (Ma et al., 2021; Wang et al., 2023b), we design the benchmark to evaluate the trustworthiness of LLMs on six aspects: truthfulness, safety, fairness, robustness, privacy, and machine ethics. We incorporate both existing and new datasets first proposed (as shown in Table 4). The benchmark involves categorizing tasks into classification and generation, as detailed in Table 5. Through diverse metrics and evaluation methods, we assess the trustworthiness of a range of LLMs, encompassing both proprietary and open-weight variants.

2 Augment context when extracting images

Accept the null hypothesis that things won't improve and run experiments to actually find what will impact the system

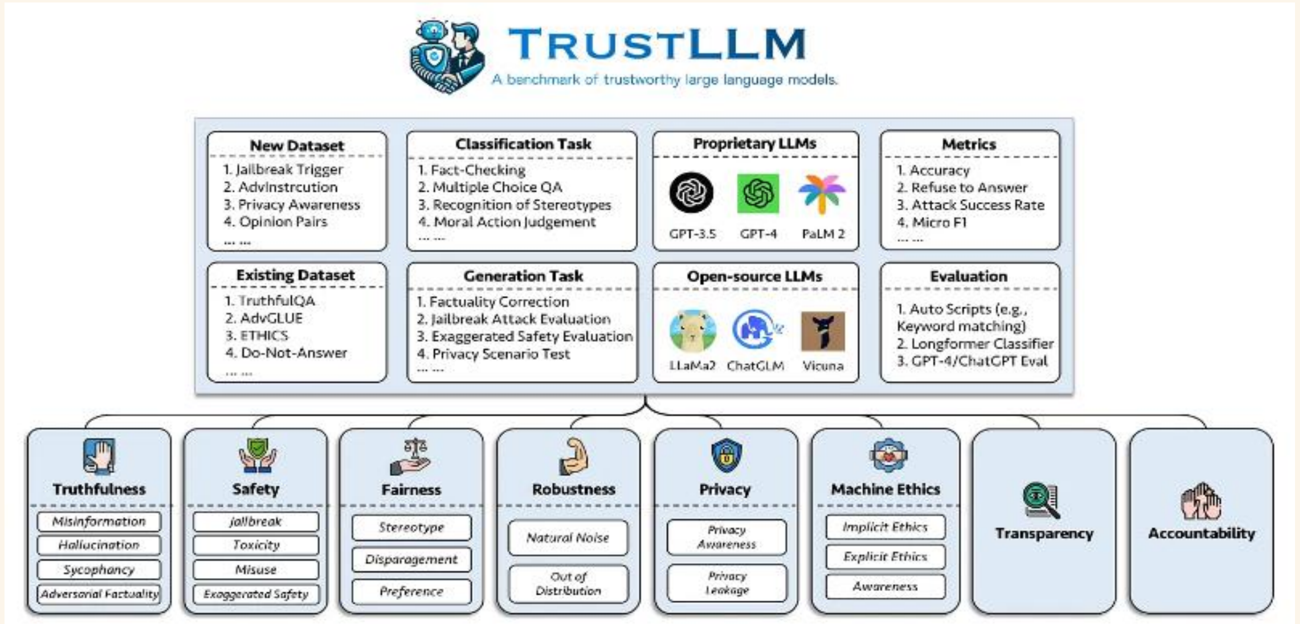
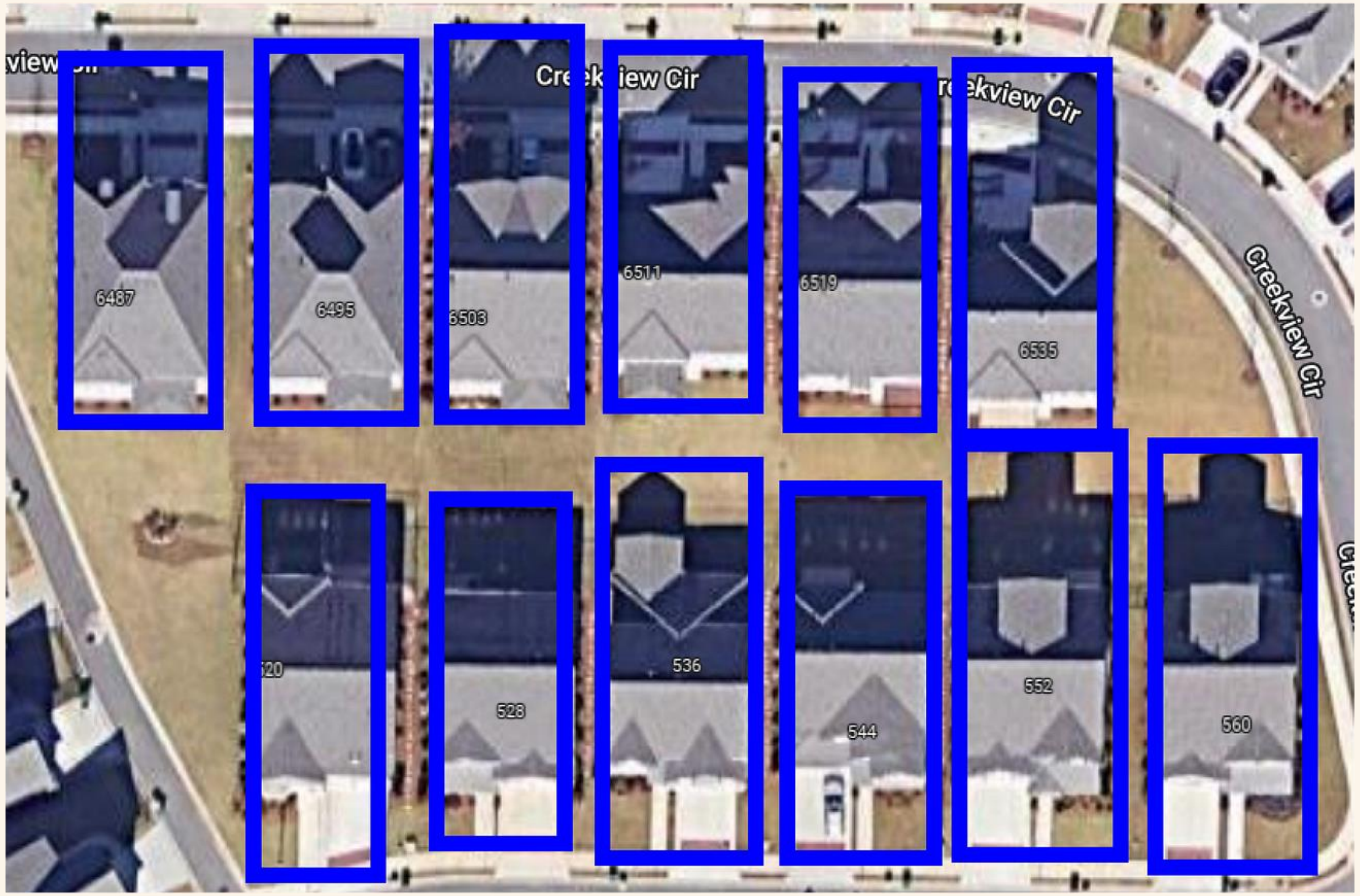


Figure 1. The design of benchmark in TRUSTLLM. Building upon the evaluation principles in prior research (Ma et al., 2021; Wang et al., 2023b), we design the benchmark to evaluate the trustworthiness of LLMs on six aspects: truthfulness, safety, fairness, robustness, privacy, and machine ethics. We incorporate both existing and new datasets first proposed (as shown in Table 4). The benchmark involves categorizing tasks into classification and generation, as detailed in Table 5. Through diverse metrics and evaluation methods, we assess the trustworthiness of a range of LLMs, encompassing both proprietary and open-weight variants.

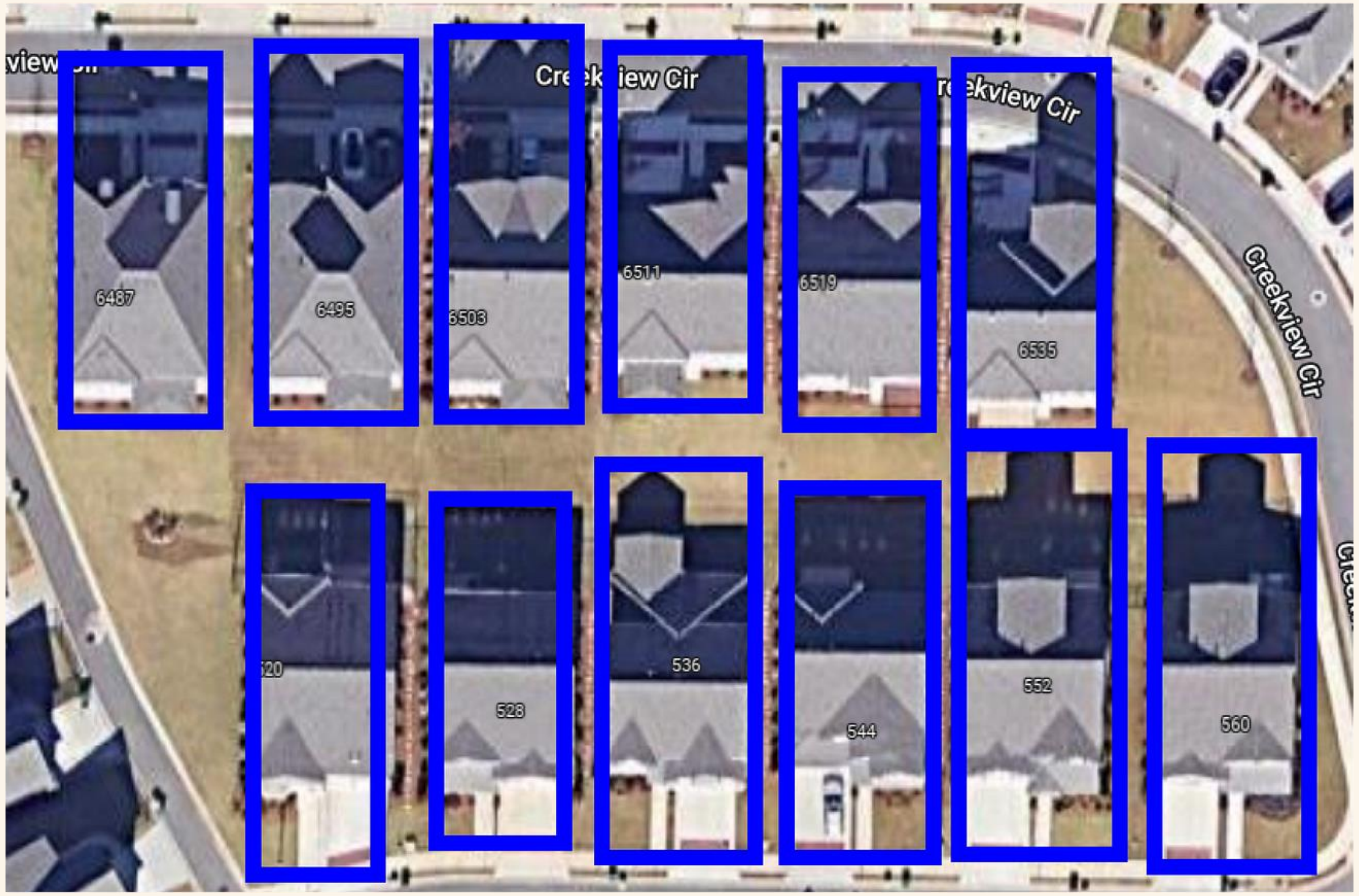
Process

1. **As a part of chunking**:
   - Provide the image to the model
   - Consider adding the text above and below to the model
   - Provide OCR data when applicable
2. **Potentially include all these in the prompt for better extraction**
3. **Continue to experiment and test with synthetic data sets.** For example, if the methods below don't improve recall or precision, save yourself money:
   1. Running OCR to generate question-retrieval pairs does not improve metrics
   2. Additional text added to the language models do not improve recall

3 Consider using bounding box regression and visual grounding



3 Consider using bounding box regression and visual grounding



We already consider:

- Different chunking mechanisms
- Different data extraction mechanisms

We should also consider:

- Explicitly do visual grounding to do better object detection to solve issues (e.g., counting, facial recognition)

Beyond the scope of the class, but if it's relevant to you, let's talk on slack

slack

@jxnlco

maven.com/applied-llms/rag-playbook

Now, you have image-description indices to generate synthetic data to test recall.

You can also segment questions and image types to evaluate any needs for specialization.

Remember: Capabilities vs. Inventory

Agenda

How to handle tables – at a glance

Process	Use case	Approach	Example
Chunk table	Semantic search on tables	1. Generate summaries of table statements and descriptions to ‘search’ against using lexical and semantic search 2. Search over these summaries and use precision recall metrics to improve from benchmark metrics 3. Make sure to preserve headers as part of chunking	Find specific table across many years or specifiation documents
Directly incorporate table metadata	Table is just another search index	1. Standardize Schemas and build into semantic search index 2. Incorporate the table as metadata for an additional search index 3. Query this search index when receiving relevant search queries	Normalizing Financial reports into a specific index

How to handle tables – at a glance

Process	Use case	Approach	Example
Chunk table	Semantic search on tables	1. Generate summaries of table statements and descriptions to 'search' against using lexical and semantic search 2. Search over these summaries and use precision recall metrics to improve from benchmark metrics 3. Make sure to preserve headers as part of chunking	Find specific table across many years or specification documents
Directly incorporate table metadata	Table is just another search index	1. Standardize Schemas and build into semantic search index 2. Incorporate the table as metadata for an additional search index 3. Query this search index when receiving relevant search queries	Normalizing Financial reports into a specific index
Long context window with full table	Treat table data as just another document	1. Include the context above and below the table as additional information 2. Generate summary indices and test precision recall metrics to determine if you're retrieving the correct tables	Process specific financial information or metrics from PDFs
Code Generation	Run SQL queries on the tables	1. Prepare tables into SQL or Pandas and do code generation 2. Determine the main query types people have. 3. Generate summaries of table statements and descriptions to 'search' against using lexical and semantic search 4. With new queries, search for the right table and the correct codesnippets	

Baseline

- **Focus on text-to-SQL on generation**. Main challenges:
  - **Multiple correct answer**: You can test the SQL statements but some SQL statements that generate more/less columns will still get the right answer
  - **Time consuming**: You run the SQL query and compare results, but this will take time
- Main challenges:
  - In most practical situations, people want to query many tables with complex schemas with 10s or 100s of columns

Baseline

- **Focus on text-to-SQL on generation**. Main challenges:
  - **Multiple correct answer**: You can test the SQL statements but some SQL statements that generate more/less columns will still get the right answer
  - **Time consuming**: You run the SQL query and compare results, but this will take time
- Main challenges:
  - In most practical situations, people want to query many tables with complex schemas with 10s or 100s of columns

Proposed new approach

- **Focus on inventory:**
  - Do we have the right tables?
  - Do we have the right columns
- **Focus on capabilities**:
  - Do we have the ability to write the right queries and find the right joins?
- **Define and measure metrics**:
  - Precision and recall metrics on tables, columns, and segments
- **Generate synthetic datasets**:
  - Precision-recall datasets
  - Synthetic testing data sets

Retrieve the right assets

Objective

Test if the system can correctly return the right tables for a search query when you have many tables

Process

1. Create statements with descriptions for tables
2. Generate a summary / pointer references the table
3. Test if the system retrieves the correct table(s) for questions

Retrieve the right assets

Objective

Test if the system can correctly return the right tables for a search query when you have many tables

Process

1. Create statements with descriptions for tables
2. Generate a summary / pointer references the table
3. Test if the system retrieves the correct table(s) for questions

Examples

Question: "How many users generate more than 10K revenue?"

Relevant Tables: users, finance

For situations involving joins, when given two or three tables, we can look at whether the question we generates can resolve this

Testing

1. Create question and table name pairs to evaluate retrieval
2. Test precision and recall (as always)
3. Strive for high recall

Potential Improvement: Find Table co-occurances and use that to augment table retrieval

Example prompt template for Text-to-SQL generation

Try to bake as much domain knowledge into these prompts, change prompts based on table schema, be specific

```
You are an AI assistant designed to convert natural language queries into SQL. Use the following information about available tables and common SQL patterns to generate accurate SQL queries.

## Available Tables
{dynamically_loaded_table_schemas}

## Common SQL Patterns
{dynamically_loaded_sql_snippets}

## Instructions
1. Analyze the user's question carefully.
2. Identify the relevant tables from the schema information provided.
3. Determine if any of the common SQL patterns are applicable to the query.
4. Generate a SQL query that answers the user's question, utilizing the appropriate tables and SQL patterns.
5. Provide a brief explanation of your reasoning, including why you chose certain tables or patterns.

Now, please convert the following natural language query into SQL:
User Query: {user_query}
```

Example prompt template for Text-to-SQL generation: Dynamically loaded table schemas

Try to bake as much domain knowledge into these prompts, change prompts based on table schema, be specific

~~~
## Available Tables

1. users

   ```sql
   CREATE TABLE users (
     user_id INT PRIMARY KEY,
     username VARCHAR(50),
     email VARCHAR(100),
     signup_date DATE,
     last_login TIMESTAMP
   );
   ~~~

   Summary: Contains user account information including signup and login data.

2. orders

   ```sql
   CREATE TABLE orders (
     order_id INT PRIMARY KEY,
     user_id INT,
     order_date TIMESTAMP,
     total_amount DECIMAL(10, 2),
     status VARCHAR(20),
     FOREIGN KEY (user_id) REFERENCES users(user_id)
   );
   ```

   Summary: Stores order details including total amount and status, linked to users.

3. products

   ```sql
   CREATE TABLE products (
     product_id INT PRIMARY KEY,
     name VARCHAR(100),
     category VARCHAR(50),
     price DECIMAL(10, 2),
     stock_quantity INT
   );
   ```

   Summary: Contains product information including category, price, and stock levels.
```
```

Retrieve the right capabilities

Objective

Test if the system retrieves correct SQL queries, especially for situations involving joins and specific common analysis patterns

Process

1. Identify patterns in existing SQL queries
2. Create SQL snippets for common patterns (e.g., computing month-over-month statistics) and brief descriptions of snippets
3. Generate natural language versions of these queries and summaries of table CREATE statements
4. Test if the system can retrieve and use the correct snippets

Retrieve the right capabilities

Objective

Test if the system retrieves correct SQL queries, especially for situations involving joins and specific common analysis patterns

Process

1. Identify patterns in existing SQL queries
2. Create SQL snippets for common patterns (e.g., computing month-over-month statistics) and brief descriptions of snippets
3. Generate natural language versions of these queries and summaries of table CREATE statements
4. Test if the system can retrieve and use the correct snippets

Examples

Question: "What is the month-over-month revenue growth?"

Relevant Snippet: [SQL code for calculating month-over-month growth]

Description: "Use this snippet when computing month-over-month or week-over-week statistics"

Testing

1. Dynamically load relevant snippets into the prompt when it makes sense, or leave them in forever if they're widely applicable

Additional improvement

1. Build UI to collect +1 snippets to use as few shot examples in the future

Example prompt template for Text-to-SQL generation: Dynamically loaded SQL snippets

Try to bake as much domain knowledge into these prompts, change prompts based on table schema, be specific

1. Month-over-Month Growth

```sql
WITH monthly_data AS (
  SELECT
    DATE_TRUNC('month', order_date) AS month,
    SUM(total_amount) AS monthly_revenue
  FROM orders
  GROUP BY DATE_TRUNC('month', order_date)
)
SELECT
  month,
  monthly_revenue,
  LAG(monthly_revenue) OVER (ORDER BY month) AS prev_month_revenue,
  (monthly_revenue - LAG(monthly_revenue) OVER (ORDER BY month)) /
    LAG(monthly_revenue) OVER (ORDER BY month) * 100 AS month_over_month_growth
FROM monthly_data
ORDER BY month;
```

Summary: Use this snippet when computing month-over-month or week-over-week statistics for revenue or other metrics.

2. Customer Repeat Purchase Analysis

```sql
WITH customer_purchases AS (
  SELECT
    user_id,
    order_date,
    LEAD(order_date) OVER (PARTITION BY user_id ORDER BY order_date) AS next_order_date
  FROM orders
)
SELECT COUNT(DISTINCT user_id) AS repeat_customers
FROM customer_purchases
WHERE DATEDIFF(day, order_date, next_order_date) <= 30;
```

Summary: Use this pattern to analyze customer repeat behavior within a specific timeframe, such as repeat purchases within 30 days.

maven.com/applied-llms/rag-playbook

Summary: how to handle tables

1 **Recognize inventory vs. capabilities issues for you use case**

2 **Establish precision and recall metrics and datasets**

- Question to Table Recall
- Question to Snippets Recall

3 **Continuous improvement**:

1. Analyze queries to identify underperforming areas
2. Identify missing inventory (tables) or capabilities (snippets)
3. Update the system with new tables or snippets as needed

Summary: how to handle tables

1 **Recognize inventory vs. capabilities issues for you use case**

---

2 **Establish precision and recall metrics and datasets**

- Question to Table Recall
- Question to Snippets Recall

---

3 **Continuous improvement**:

1. Analyze queries to identify underperforming areas
2. Identify missing inventory (tables) or capabilities (snippets)
3. Update the system with new tables or snippets as needed

With a relatively simple system setup, the **general RAG playbook can be re-applied to any sub-system** in a smaller setting (e.g., text-to-SQL)

The Playbook

- Evals provides us with an estimate of the P(success) ~ recall
- Segmentation helps us prioritize different types of problems.
- The solution will involve splitting the monolithic retriever into multiple retrievers.

The Playbook

- Evals provides us with an estimate of the P(success) ~ recall
- Segmentation helps us prioritize different types of problems.
- The solution will involve splitting the monolithic retriever into multiple retrievers.
- These retrievers will likely resemble structured data extraction, which we can either:
    - Insert into a new table to query
    - Allow for new metadata fields
    - We do this with the assumption that an LLM will be able 'fill in the tool call'
    - Some new entities might be pointers to the source data.
- After this, we can continue to apply the synthetic data, segmentation, topics, and capabilities workflow to dig deeper into the problem.
    - The easiest example would be text to sql where we need to better retrieve tables and code snippets, using additional new tables and metadata.

Agenda

Multiple indices for RAG applications

Handling entity-specific search indices

Sneak peak of next session

Food for thought: try this at work or in your own projects

- ☐ For a specific query segment, explore a new index or new metadata you'd need to answer these questions better
- ☐ Test out specific methods we've outlined across specific entity types
- ☐ Generate questions for this new index using the skills you developed in session 1

Summary

- **Focus for last four sessions:**
  - Understand synthetic evals, segmentation to fine specific inventory and capabilities issues
  - Embeddings and representations, product design
  - Topic modeling and query segmentation
- **Focus for this session**:
  - Any specific index we build will leverage what we learned in sessions 1, 2, applying it recursively
  - Once we segment we can solve each segment individually while also working on the router (session 4)
  - Outline how to improve each specific type of retrieval index (e.g., documents, images, text-to-SQL)
  - Review how to use the RAG playbook steps (e.g., generate synthetic data, focus on recall-precision metrics) to tackle sub-problems in improving RAG applications

Focus for next session:

- **Session 6**: Apply: Routing queries