jxnl.co

@jxnlco

Systematically Improving RAG Applications

Session 6

Apply: Function Calling Done Right

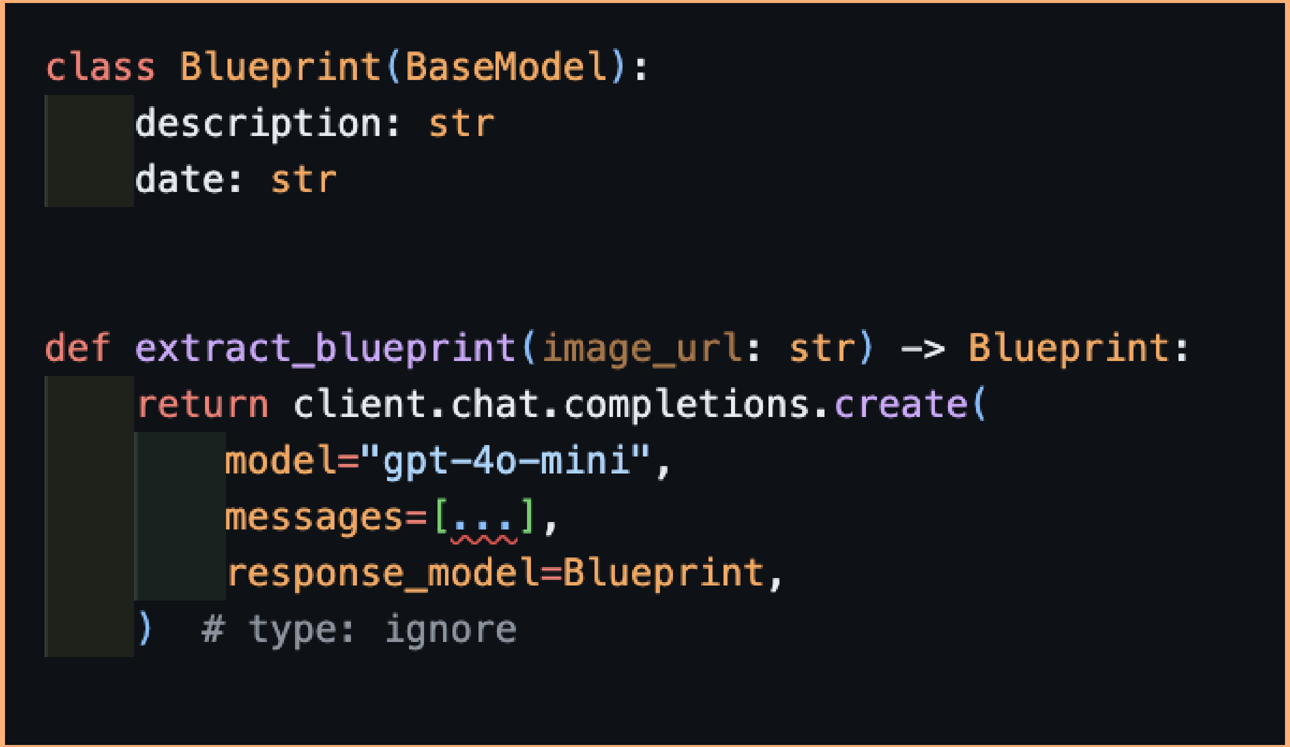
Jason Liu

Agenda

Extract into Index

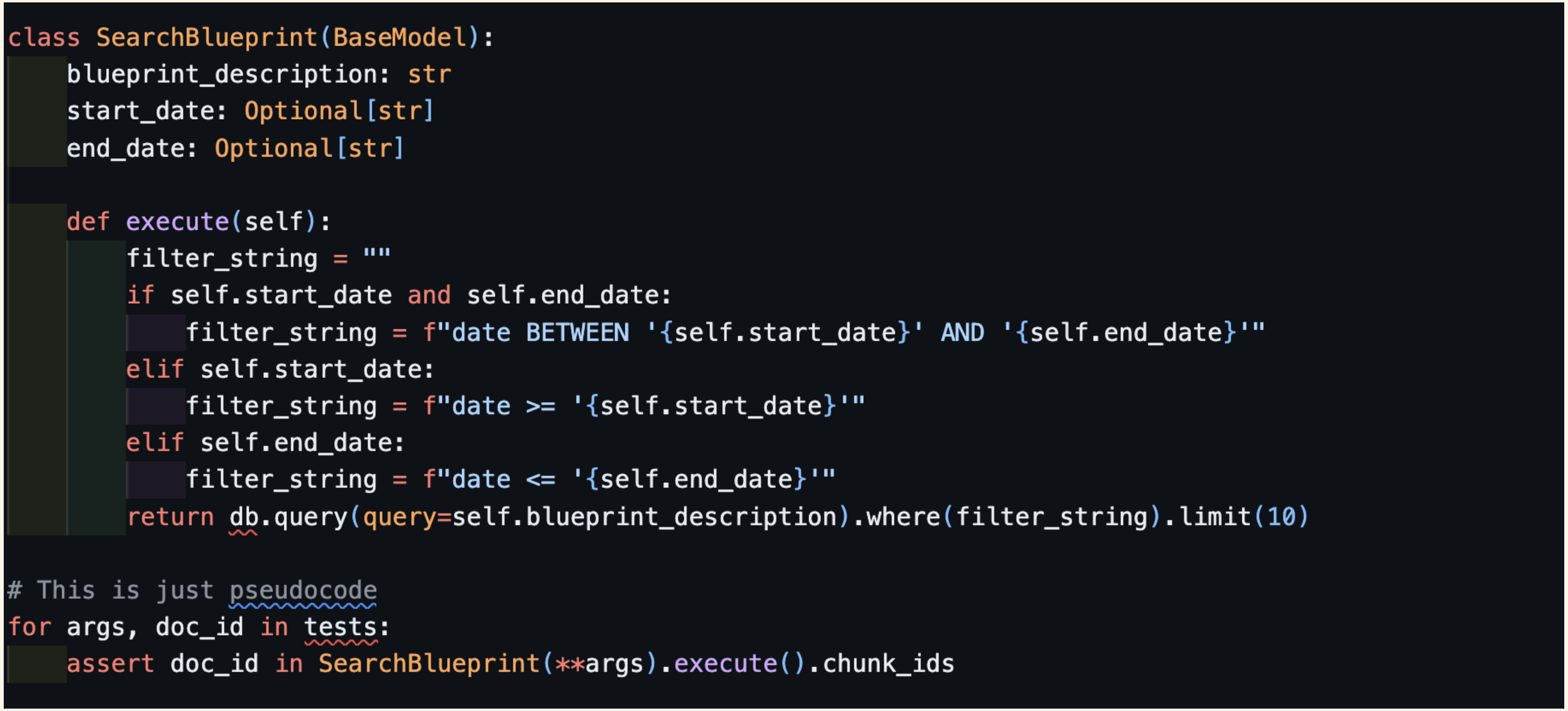
So maybe, based on what you did last session, we constructed a hypothetical new image index that searches blueprints.

Now we have a new db

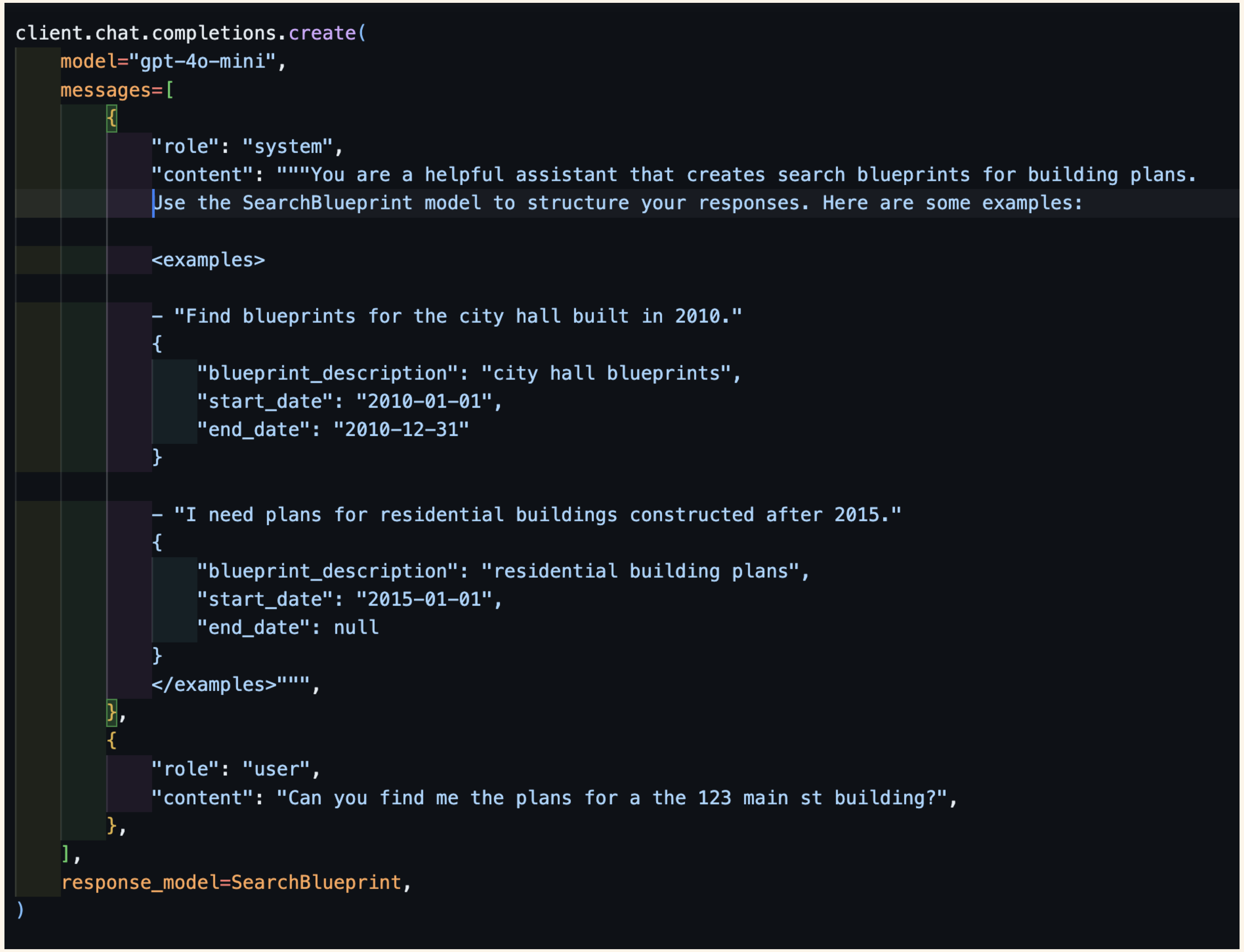


```python
class Blueprint(BaseModel):
    description: str
    date: str


def extract_blueprint(image_url: str) -> Blueprint:
    return client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[...],
        response_model=Blueprint,
    )  # type: ignore
```

Defining the Tool



```python
class SearchBlueprint(BaseModel):
    blueprint_description: str
    start_date: Optional[str]
    end_date: Optional[str]

    def execute(self):
        filter_string = ""
        if self.start_date and self.end_date:
            filter_string = f"date BETWEEN '{self.start_date}' AND '{self.end_date}'"
        elif self.start_date:
            filter_string = f"date >= '{self.start_date}'"
        elif self.end_date:
            filter_string = f"date <= '{self.end_date}'"
        return db.query(query=self.blueprint_description).where(filter_string).limit(10)

# This is just pseudocode
for args, doc_id in tests:
    assert doc_id in SearchBlueprint(**args).execute().chunk_ids
```



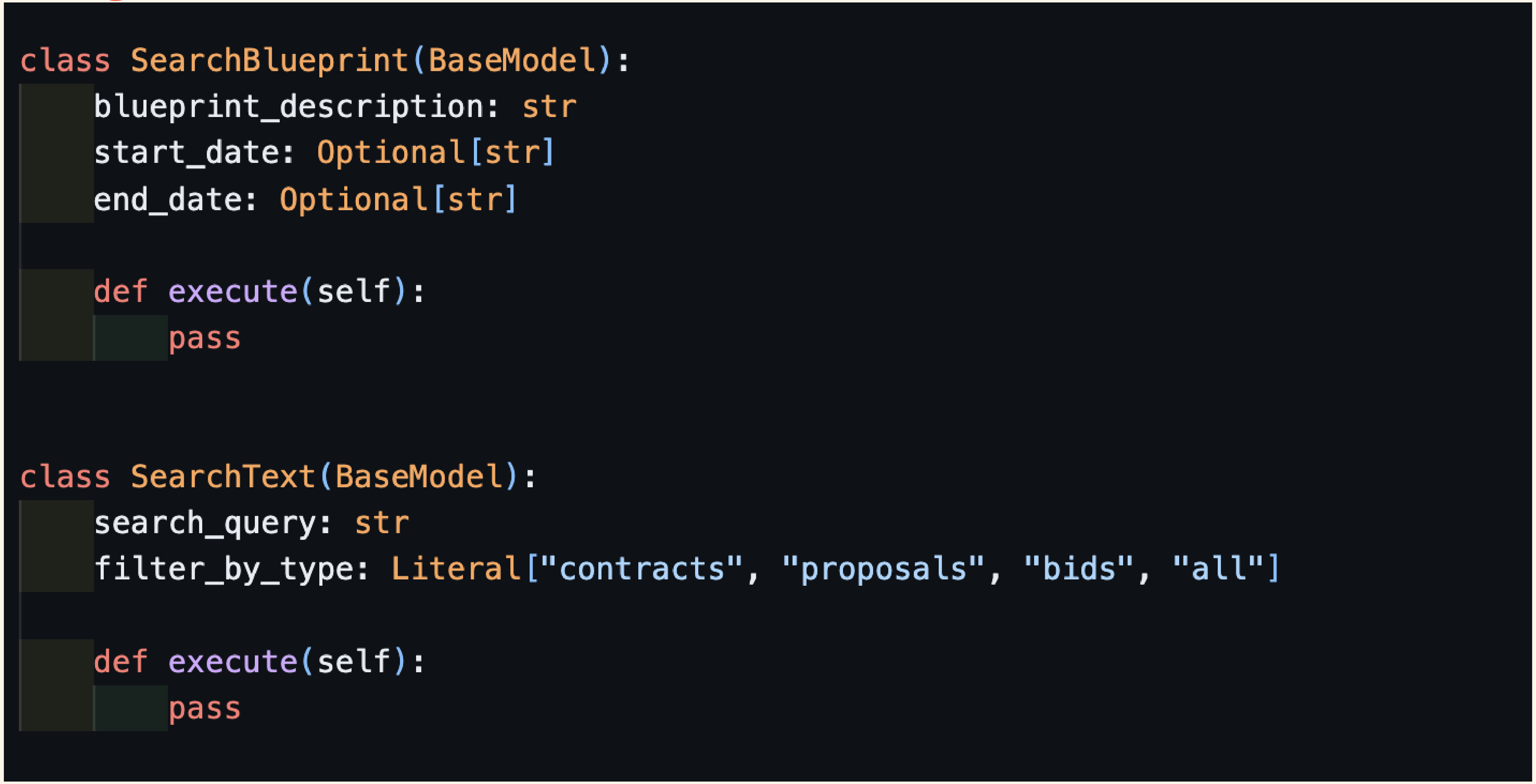
```
client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        {
            "role": "system",
            "content": """You are a helpful assistant that creates search blueprints for building plans.
            Use the SearchBlueprint model to structure your responses. Here are some examples:

            <examples>

            - "Find blueprints for the city hall built in 2010."
            {
                "blueprint_description": "city hall blueprints",
                "start_date": "2010-01-01",
                "end_date": "2010-12-31"
            }

            - "I need plans for residential buildings constructed after 2015."
            {
                "blueprint_description": "residential building plans",
                "start_date": "2015-01-01",
                "end_date": null
            }
            </examples>""",
        },
        {
            "role": "user",
            "content": "Can you find me the plans for a the 123 main st building?",
        },
    ],
    response_model=SearchBlueprint,
)
```

A retrieval method should feel more like a method in a REST API rather than the database itself.

Many APIS could hit the DB in different ways

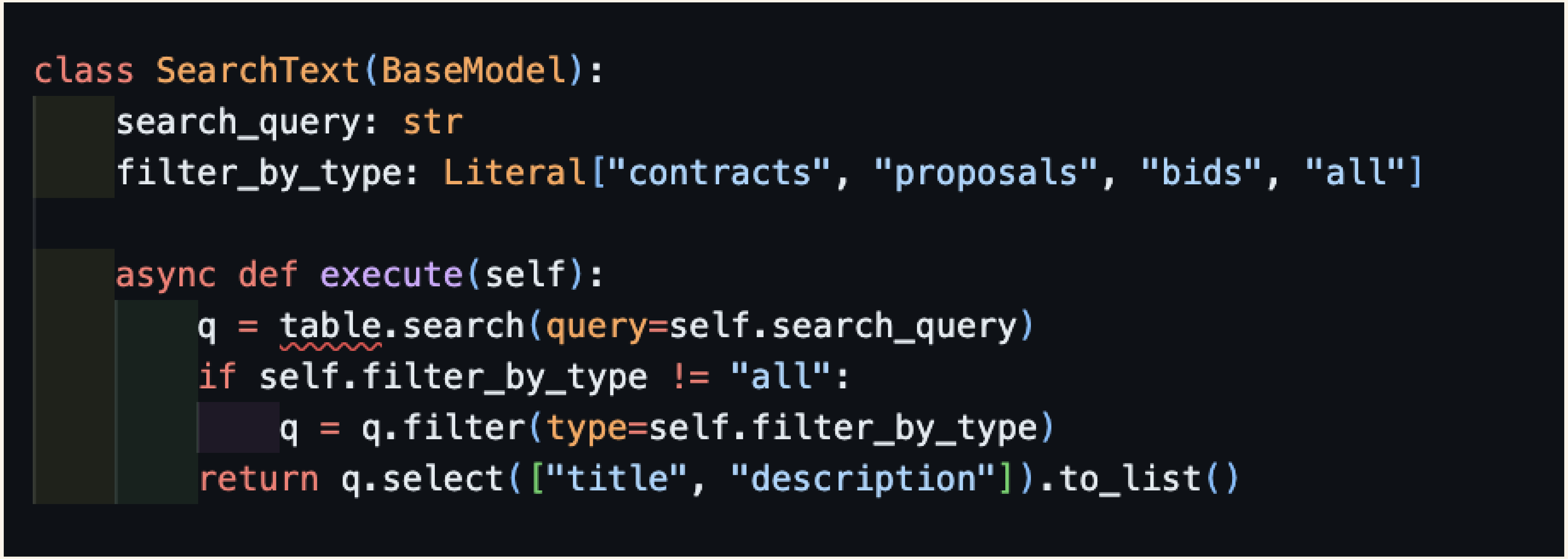
You are a framework developer for an LLM!

Defining the Interface / Front end



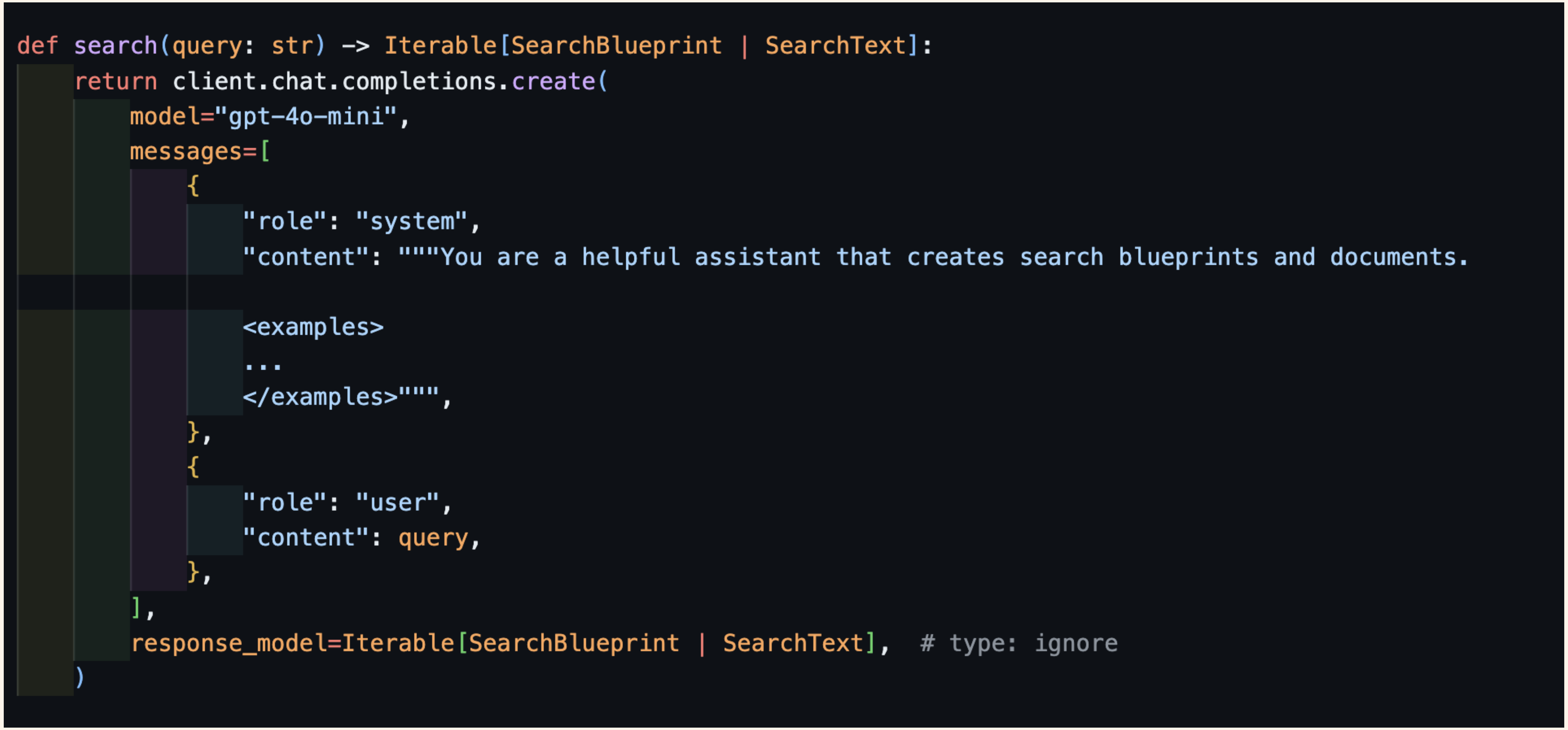
```python
class SearchBlueprint(BaseModel):
    blueprint_description: str
    start_date: Optional[str]
    end_date: Optional[str]

    def execute(self):
        pass


class SearchText(BaseModel):
    search_query: str
    filter_by_type: Literal["contracts", "proposals", "bids", "all"]

    def execute(self):
        pass
```

Defining the Implementation / Backend



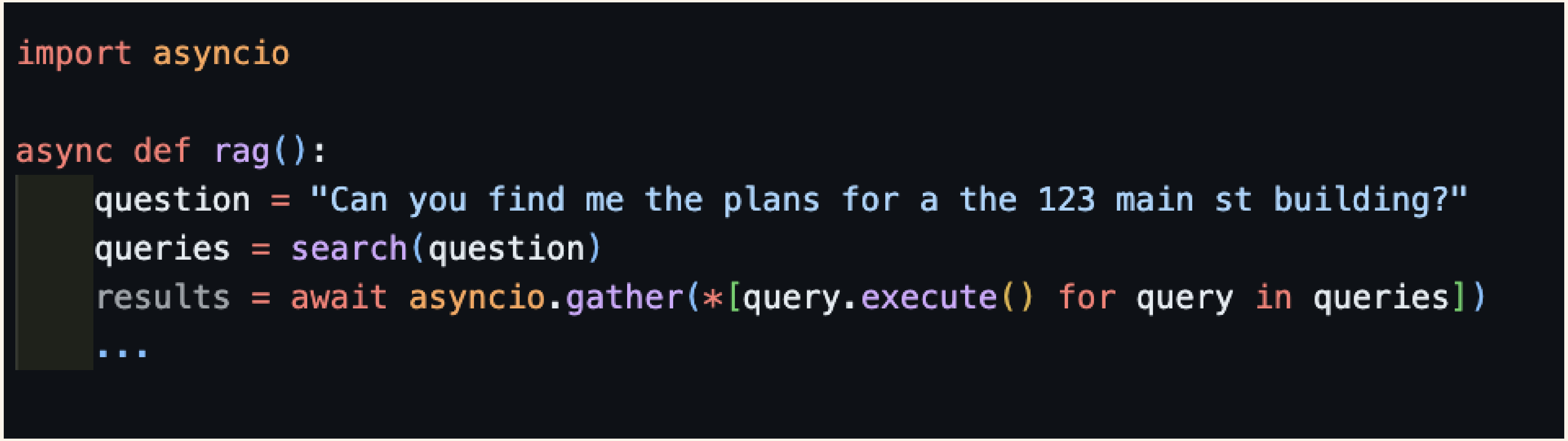
```python
class SearchText(BaseModel):
    search_query: str
    filter_by_type: Literal["contracts", "proposals", "bids", "all"]

    async def execute(self):
        q = table.search(query=self.search_query)
        if self.filter_by_type != "all":
            q = q.filter(type=self.filter_by_type)
        return q.select(["title", "description"]).to_list()
```

Defining the Gateway / Router



```
def search(query: str) -> Iterable[SearchBlueprint | SearchText]:
    return client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[
            {
                "role": "system",
                "content": """You are a helpful assistant that creates search blueprints and documents.

                <examples>
                ...
                </examples>""",
            },
            {
                "role": "user",
                "content": query,
            },
        ],
        response_model=Iterable[SearchBlueprint | SearchText],  # type: ignore
    )
```

Putting it together



```python
import asyncio

async def rag():
    question = "Can you find me the plans for a the 123 main st building?"
    queries = search(question)
    results = await asyncio.gather(*[query.execute() for query in queries])
    ...
```

Separation of concerts

Here, you might notice that there are a few standard abstractions:

1. **Interface**: Defining tools
2. **Implementation**: Implementing the execution of a single tool
3. **Gateway**: Putting it all together using a parallel tool called an API, like the one you see from OpenAI

Leveraging Parallel Tools

- By leveraging parallel tool calling, we are able to extract tools to improve our RAG
- We can also then run the API
- Available in many implementations in both Llamaindex and Langchain as well!

***Initial setup**: Setup precision recall to test whether we are calling the right tool*

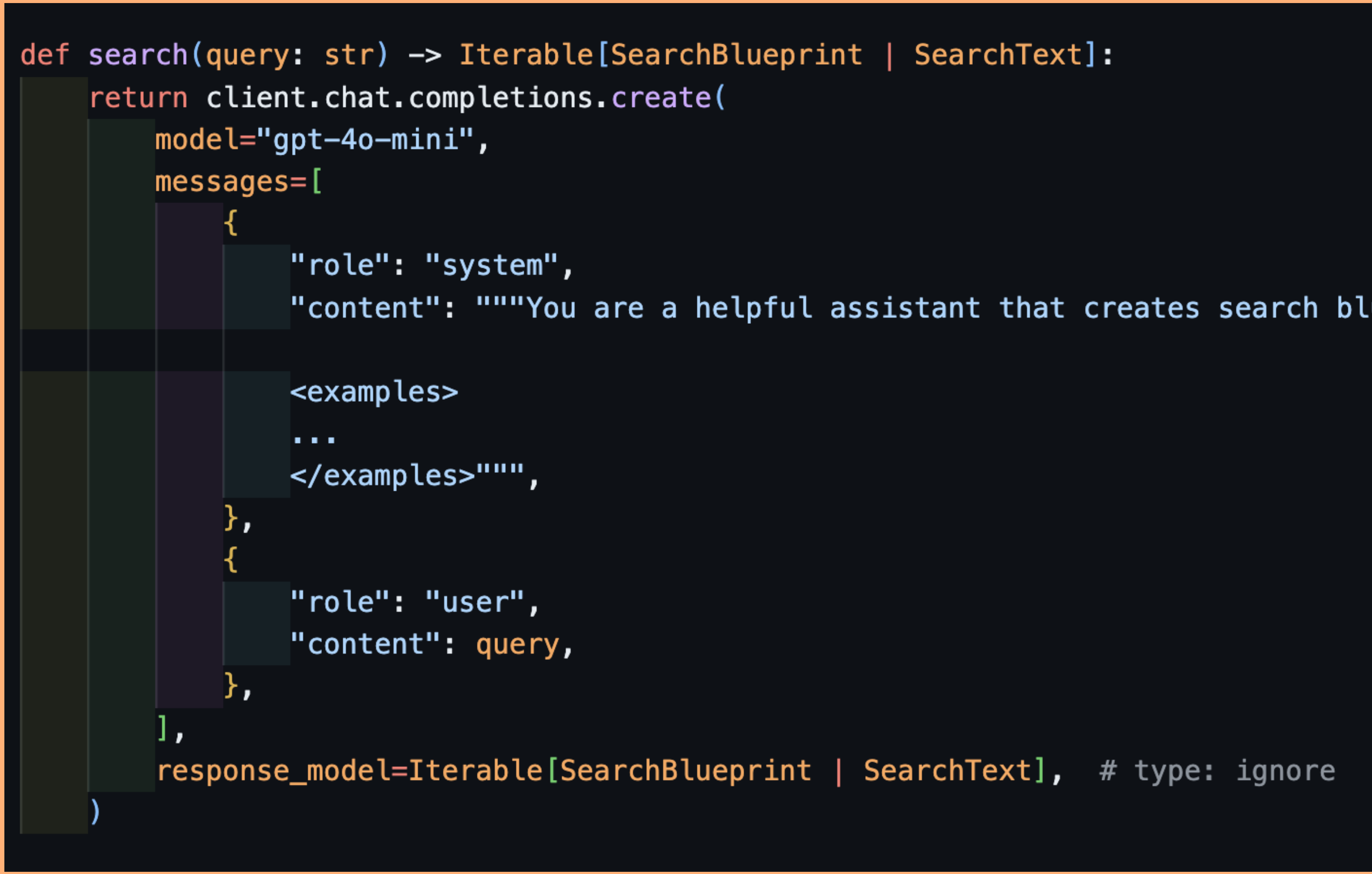
***Continuous improvement**: Focus on better prompting and different types of arguments (e.g., separate deterministic vs. probabilistic arguments)*

Leveraging Parallel Tools

- By leveraging parallel tool calling, we are able to extract tools to improve our RAG
- We can also then run the API
- Available in many implementations in both Llamaindex and Langchain as well!

***Initial setup**: Setup precision recall to test whether we are calling the right tool*

***Continuous improvement**: Focus on better prompting and different types of arguments (e.g., separate deterministic vs. probabilistic arguments)*

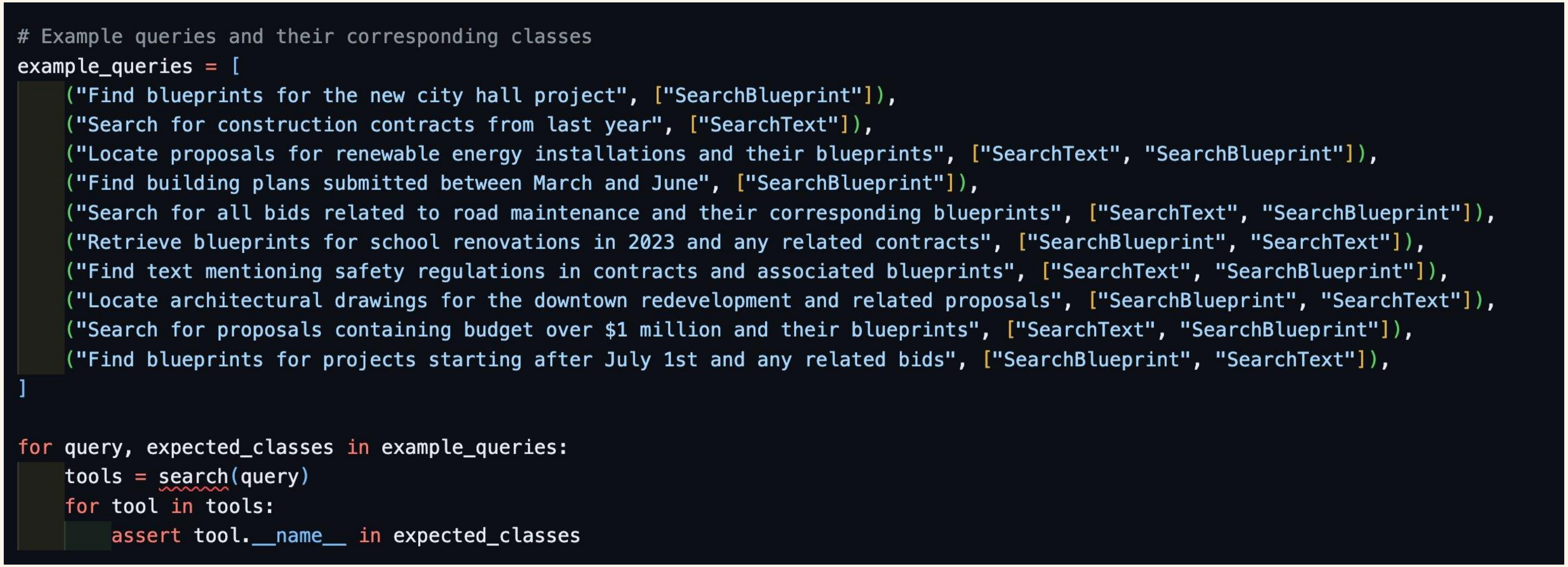


```
def search(query: str) -> Iterable[SearchBlueprint | SearchText]:
    return client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[
            {
                "role": "system",
                "content": """You are a helpful assistant that creates search bl

                <examples>
                ...
                </examples>""",
            },
            {
                "role": "user",
                "content": query,
            },
        ],
        response_model=Iterable[SearchBlueprint | SearchText],  # type: ignore
    )
```

@jxnlco

maven.com/applied-llms/rag-playbook

V0 Tests, tool recall



```python
# Example queries and their corresponding classes
example_queries = [
    ("Find blueprints for the new city hall project", ["SearchBlueprint"]),
    ("Search for construction contracts from last year", ["SearchText"]),
    ("Locate proposals for renewable energy installations and their blueprints", ["SearchText", "SearchBlueprint"]),
    ("Find building plans submitted between March and June", ["SearchBlueprint"]),
    ("Search for all bids related to road maintenance and their corresponding blueprints", ["SearchText", "SearchBlueprint"]),
    ("Retrieve blueprints for school renovations in 2023 and any related contracts", ["SearchBlueprint", "SearchText"]),
    ("Find text mentioning safety regulations in contracts and associated blueprints", ["SearchText", "SearchBlueprint"]),
    ("Locate architectural drawings for the downtown redevelopment and related proposals", ["SearchBlueprint", "SearchText"]),
    ("Search for proposals containing budget over $1 million and their blueprints", ["SearchText", "SearchBlueprint"]),
    ("Find blueprints for projects starting after July 1st and any related bids", ["SearchBlueprint", "SearchText"]),
]

for query, expected_classes in example_queries:
    tools = search(query)
    for tool in tools:
        assert tool.__name__ in expected_classes
```

Notice again…

Data is crucial for evaluating the performance of our tools.

We should focus on the appropriate recall metrics to ensure we're selecting the right tool for the job.

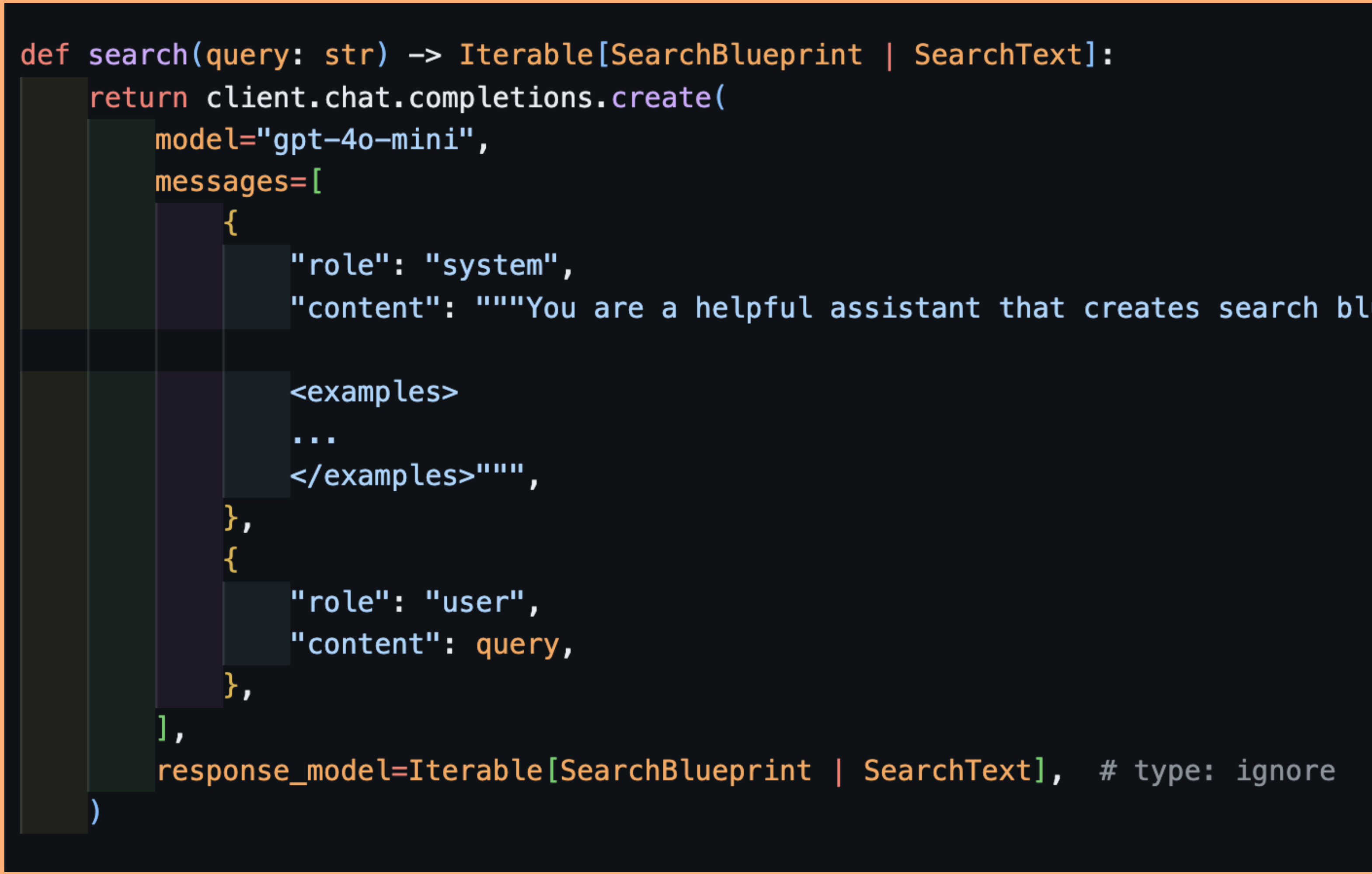
Precision will matter more since we won't want wasted compute

Descriptions and Few-shot examples

Once we have a baseline, we can experiment with tool descriptions and few-shot examples in the prompt.

Descriptions and Few-shot examples

Once we have a baseline, we can experiment with tool descriptions and few-shot examples in the prompt.



```
def search(query: str) -> Iterable[SearchBlueprint | SearchText]:
    return client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[
            {
                "role": "system",
                "content": """You are a helpful assistant that creates search bl

                <examples>
                ...
                </examples>""",
            },
            {
                "role": "user",
                "content": query,
            },
        ],
        response_model=Iterable[SearchBlueprint | SearchText],  # type: ignore
    )
```

@jxnlco

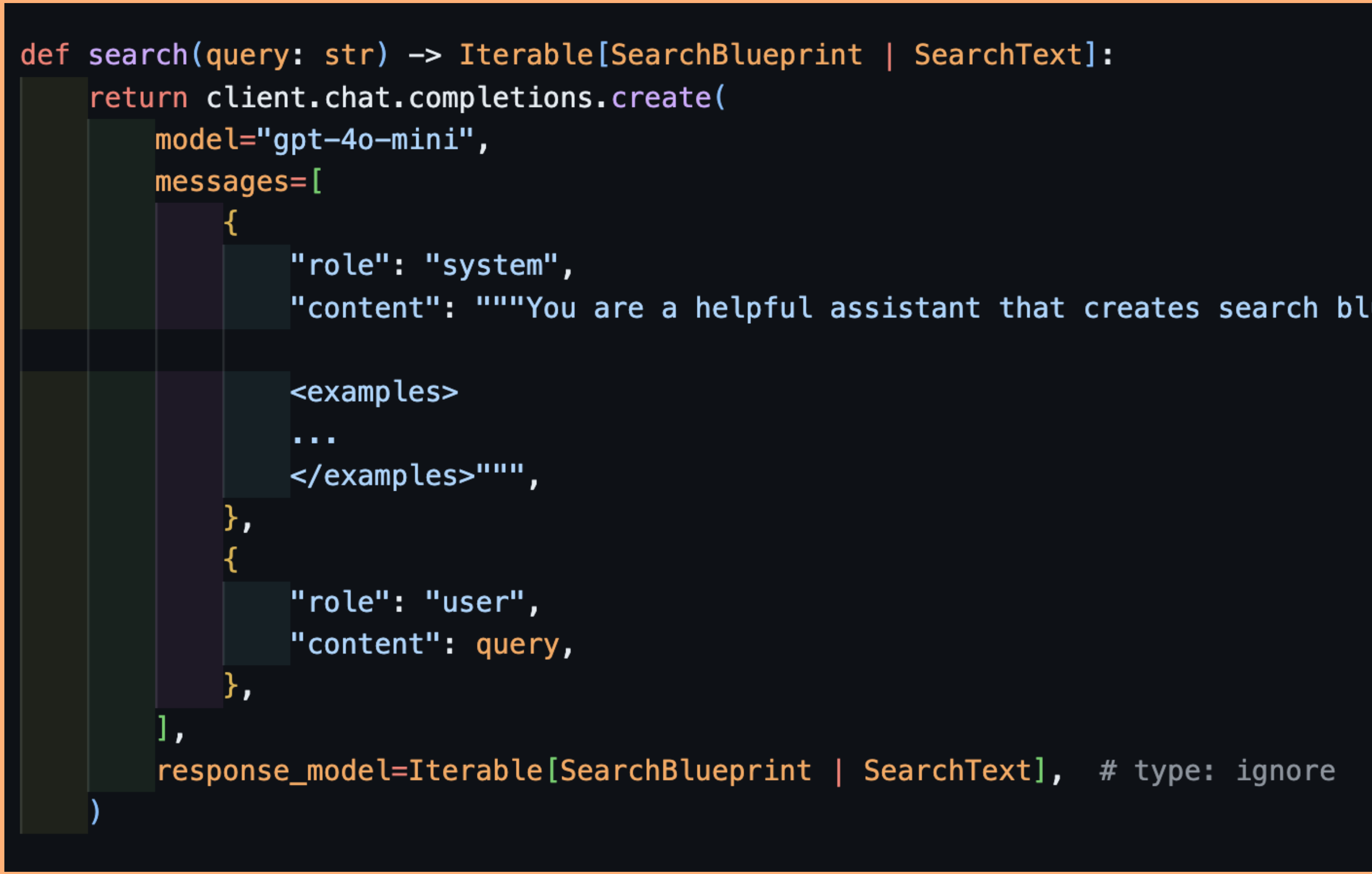
maven.com/applied-llms/rag-playbook

Descriptions and Few-shot examples

Once we have a baseline, we can experiment with tool descriptions and few-shot examples in the prompt.

In v0, we may hard code those examples.

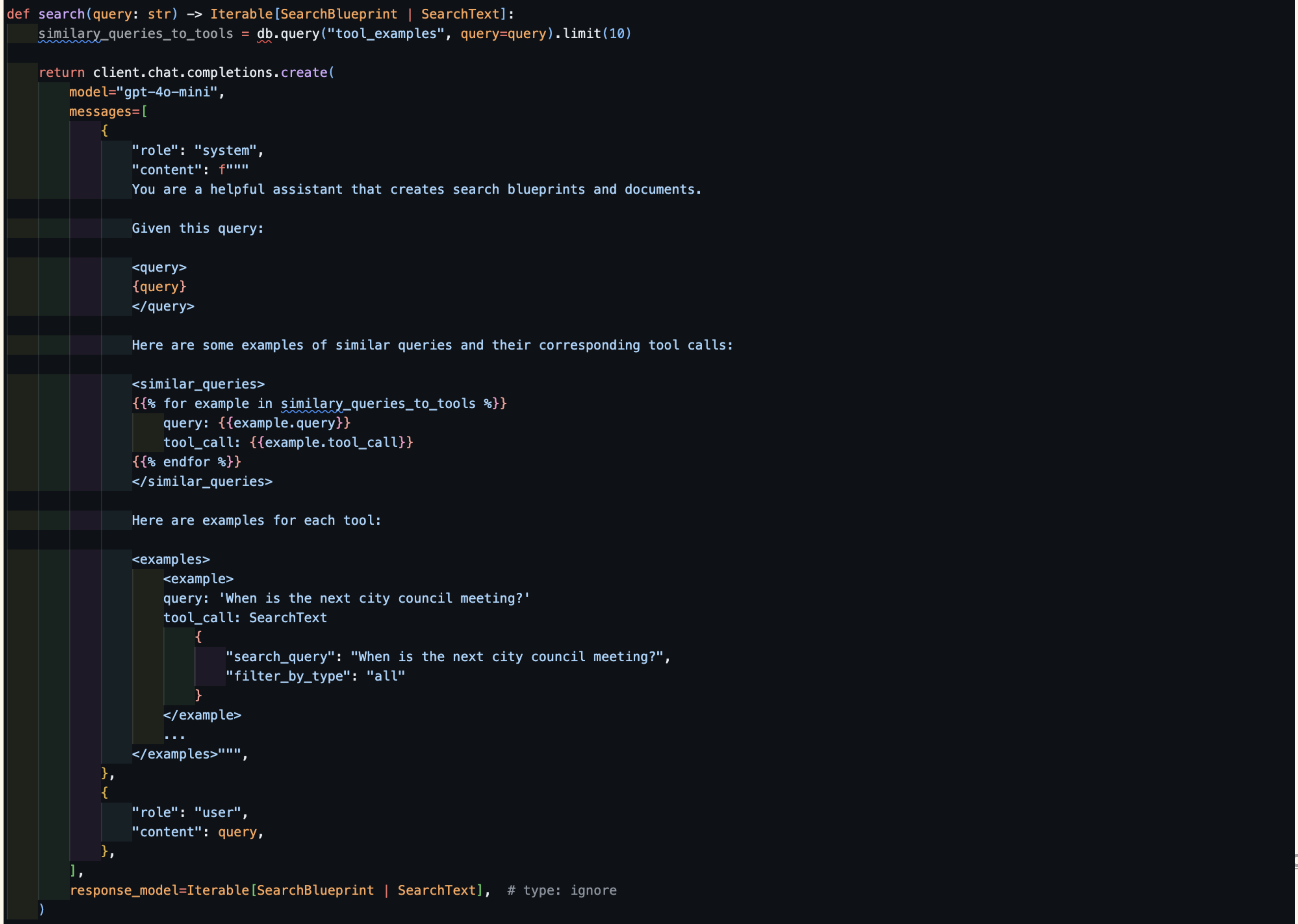
As we get more complex, we can do something similar to our Text2SQL and use search to fill in ideal few-shot examples per tool, evaluating the system to ensure improved recall.



```python
def search(query: str) -> Iterable[SearchBlueprint | SearchText]:
    return client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[
            {
                "role": "system",
                "content": """You are a helpful assistant that creates search bl

                <examples>
                ...
                </examples>""",
            },
            {
                "role": "user",
                "content": query,
            },
        ],
        response_model=Iterable[SearchBlueprint | SearchText],  # type: ignore
    )
```

A step by step summary:

1. Use Synthetic Data flywheel to produce query-to-tool or tool-to-query data
2. Produce Recall metrics (more on this later)
3. Iterate on few shot examples
   - For each tool list out:
     - Examples of queries -> Tool arguments
4. Iterate on a search system to find optimal examples per user query.
   - You can leverage the initial dataset or production data to search over queries and their respective tools.
   - Similar to a summary index but instead of summary, its query mappings



```python
def search(query: str) -> Iterable[SearchBlueprint | SearchText]:
    similary_queries_to_tools = db.query("tool_examples", query=query).limit(10)

    return client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[
            {
                "role": "system",
                "content": f"""
                You are a helpful assistant that creates search blueprints and documents.

                Given this query:

                <query>
                {query}
                </query>

                Here are some examples of similar queries and their corresponding tool calls:

                <similar_queries>
                {{% for example in similary_queries_to_tools %}}
                    query: {{example.query}}
                    tool_call: {{example.tool_call}}
                {{% endfor %}}
                </similar_queries>

                Here are examples for each tool:

                <examples>
                    <example>
                    query: 'When is the next city council meeting?'
                    tool_call: SearchText
                        {
                            "search_query": "When is the next city council meeting?",
                            "filter_by_type": "all"
                        }
                    </example>
                    ...
                </examples>""",
            },
            {
                "role": "user",
                "content": query,
            },
        ],
        response_model=Iterable[SearchBlueprint | SearchText],  # type: ignore
    )
```

Agenda

Query routing vs. retrieval indices

Query routing main challenges

Further testing

UI considerations

Food for thought for this session

Course Conclusion

Low per-class recall

We can segment questions by which tools should be recovered

Each tool can be considered a class

We evaluate the recall for each class individually

How to improve per-class recall

- Better, more-specific tool descriptions
- Improve few-shot examples to let the LLM know what exactly is going on

Low per-class recall

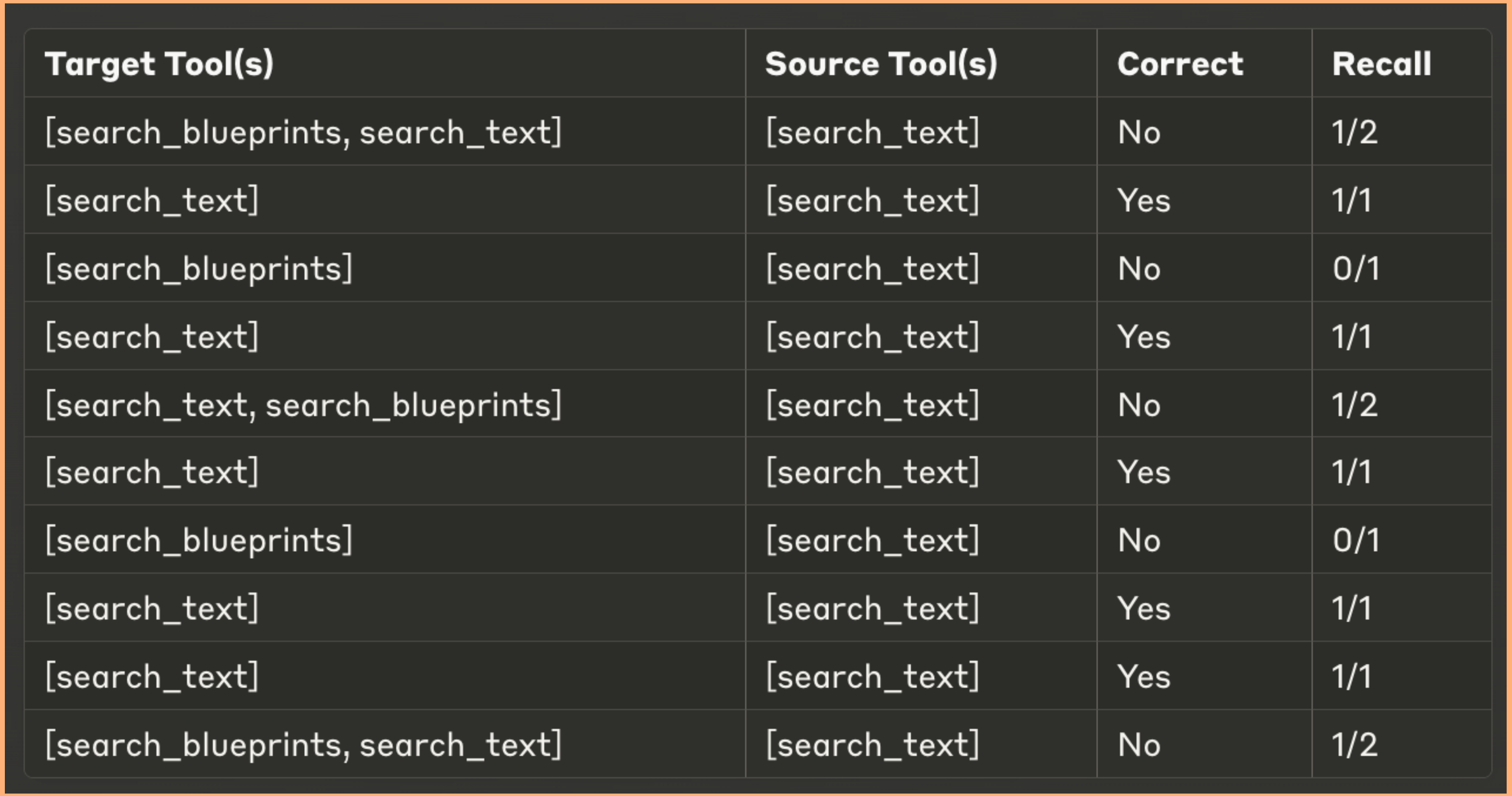
We can segment questions by which tools should be recovered

Each tool can be considered a class

We evaluate the recall for each class individually

How to improve per-class recall

- Better, more-specific tool descriptions
- Improve few-shot examples to let the LLM know what exactly is going on



Target Tool(s)	Source Tool(s)	Correct	Recall
[search_blueprints, search_text]	[search_text]	No	1/2
[search_text]	[search_text]	Yes	1/1
[search_blueprints]	[search_text]	No	0/1
[search_text]	[search_text]	Yes	1/1
[search_text, search_blueprints]	[search_text]	No	1/2
[search_text]	[search_text]	Yes	1/1
[search_blueprints]	[search_text]	No	0/1
[search_text]	[search_text]	Yes	1/1
[search_text]	[search_text]	Yes	1/1
[search_blueprints, search_text]	[search_text]	No	1/2

Recall is 65% for this test suite

Low per-class recall

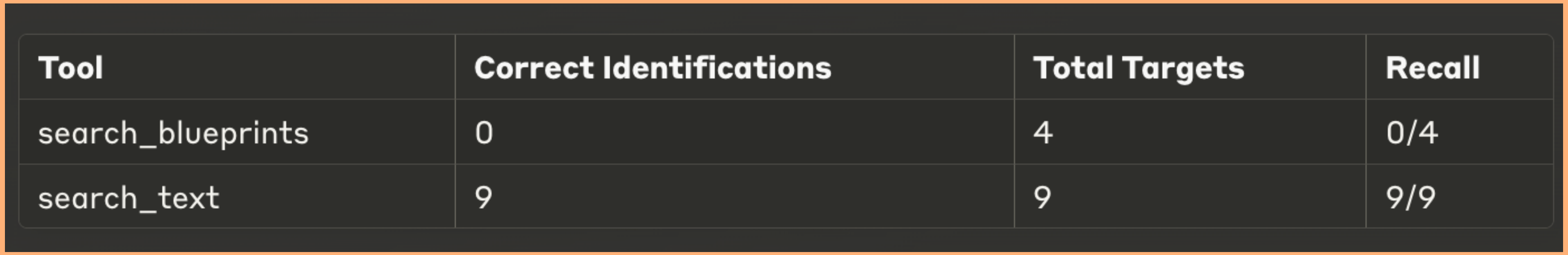
We can segment questions by which tools should be recovered

Each tool can be considered a class

We evaluate the recall for each class individually

How to improve per-class recall

- Better, more-specific tool descriptions
- Improve few-shot examples to let the LLM know what exactly is going on



Tool	Correct Identifications	Total Targets	Recall
search_blueprints	0	4	0/4
search_text	9	9	9/9

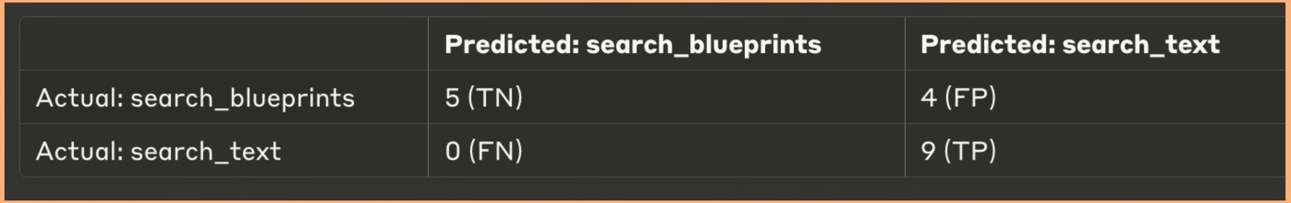
But this is because we're failing to select the blue print functions!

@jxnlco

maven.com/applied-llms/rag-playbook

Tool confusion

We can build tool confusion matrices to determine which tool is used in place of another



	Predicted: search_blueprints	Predicted: search_text
Actual: search_blueprints	5 (TN)	4 (FP)
Actual: search_text	0 (FN)	9 (TP)

How to reduce tool confusion

- Look at your data, filter for failures
- Build examples that can help delineate between tools
- Include more positive examples for each tool
- Find examples where the LLM confuses tool A for tool B; give explicit negative examples on when to use each tool
- Look through data and find examples where LLM is confused and explicitly include these in few-shot examples
- In extreme cases, merge tools

Warning on leakage

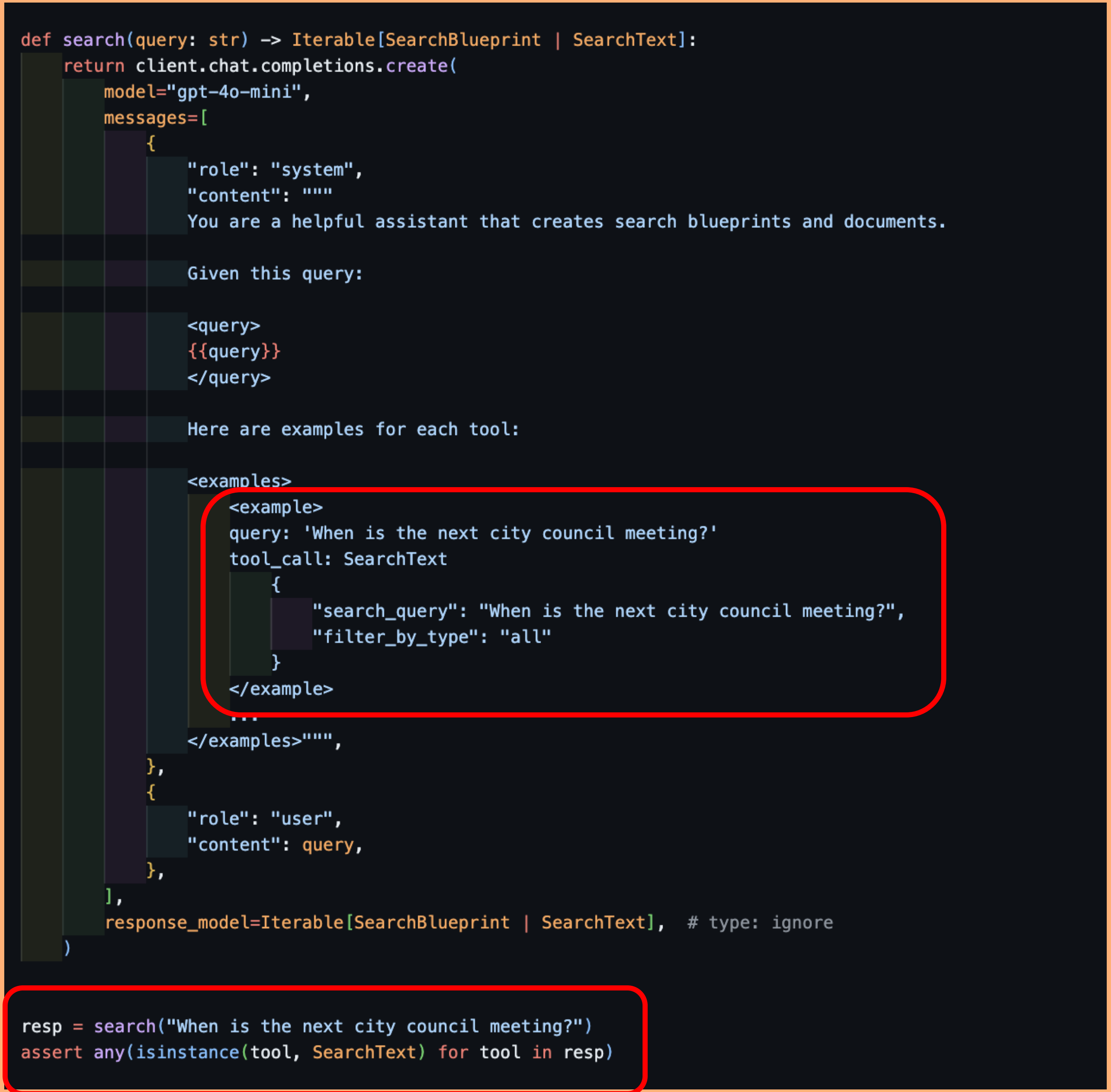
Ensure that few-shot prompt examples are not included in the test set!

- **How people typically start**: dozens of question types with higher than avg performance (depending on how synthetic data is generated)
- Due to limited examples we might overlap examples and tests

Warning on leakage

Ensure that few-shot prompt examples are not included in the test set!

- **How people typically start**: dozens of question types with higher than avg performance (depending on how synthetic data is generated)
- Due to limited examples we might overlap examples and tests
- More important to verify in the example retrieval context



```
def search(query: str) -> Iterable[SearchBlueprint | SearchText]:
  return client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
      {
        "role": "system",
        "content": """
        You are a helpful assistant that creates search blueprints and documents.

        Given this query:

        <query>
        {{query}}
        </query>

        Here are examples for each tool:

        <examples>
          <example>
          query: 'When is the next city council meeting?'
          tool_call: SearchText
            {
              "search_query": "When is the next city council meeting?",
              "filter_by_type": "all"
            }
          </example>
          ...
        </examples>""",
      },
      {
        "role": "user",
        "content": query,
      },
    ],
    response_model=Iterable[SearchBlueprint | SearchText],  # type: ignore
  )

resp = search("When is the next city council meeting?")
assert any(isinstance(tool, SearchText) for tool in resp)
```

@jxnlco

maven.com/applied-llms/rag-playbook

Agenda

How do we measure the quality of the retrieval subsystem?

		Session 4, 5		Session 6
P(Correct chunk found)	=	P(Correct chunk found | correct retriever)	×	P(correct retriever)
		Does a single search method find the correct chunks?		*Does the LLM choose the correct search method?*

How do we measure the quality of the retrieval subsystem?

		Session 4, 5		Session 6
P(Correct chunk found)	=	P(Correct chunk found | correct retriever)	×	P(correct retriever)
		Does a single search method find the correct chunks?		*Does the LLM choose the correct search method?*

Key takeaway*:*

This equation can help identify the limiting factor of the system:

- *Not using the right retriever (query not routed to the right retriever)*
- *The retriever itself is bad*

To improve P(correct retriever), continue to experiment on

- Precision vs. Recall trade-off
  - If precision didn't matter, we can get 100% recall by calling every tool if we don't care about latency
- Examples and prompts for tool selection
- Examples and prompts for arguments
- LLM models used

Interpreting the Formula (Extended)

P(success) = P(success | correct tool selected) × P(correct tool selected | query) × P(query)

Where:

P (query) = UI, Education

P (success) = Quality of application

P (success | correct tool selected) = Retrieval Quality & Generation

P (correct tool selected | query) = Router Quality

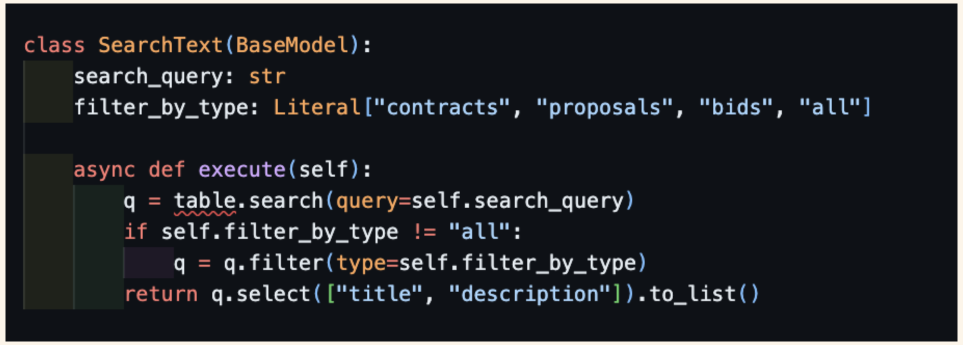
Agenda

Thinking about UI

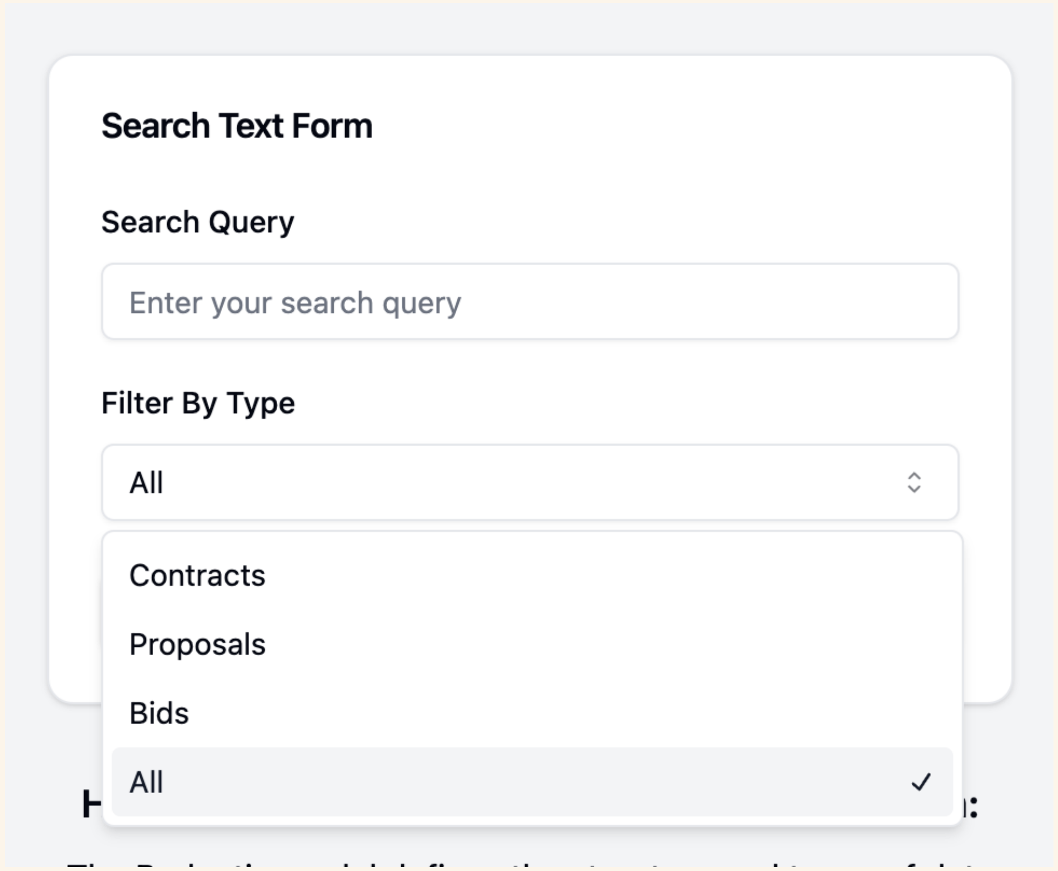
- As we build more tools, each one can be fully specified as a JSON schema, which can be used to model forms
- There are also opportunities to allow either the tool spec to represent forms that the user can review and correct.

Thinking about UI

- As we build more tools, each one can be fully specified as a JSON schema, which can be used to model forms
- There are also opportunities to allow either the tool spec to represent forms that the user can review and correct.



```python
class SearchText(BaseModel):
    search_query: str
    filter_by_type: Literal["contracts", "proposals", "bids", "all"]

    async def execute(self):
        q = table.search(query=self.search_query)
        if self.filter_by_type != "all":
            q = q.filter(type=self.filter_by_type)
        return q.select(["title", "description"]).to_list()
```



Search Text Form

Search Query

Enter your search query

Filter By Type

All

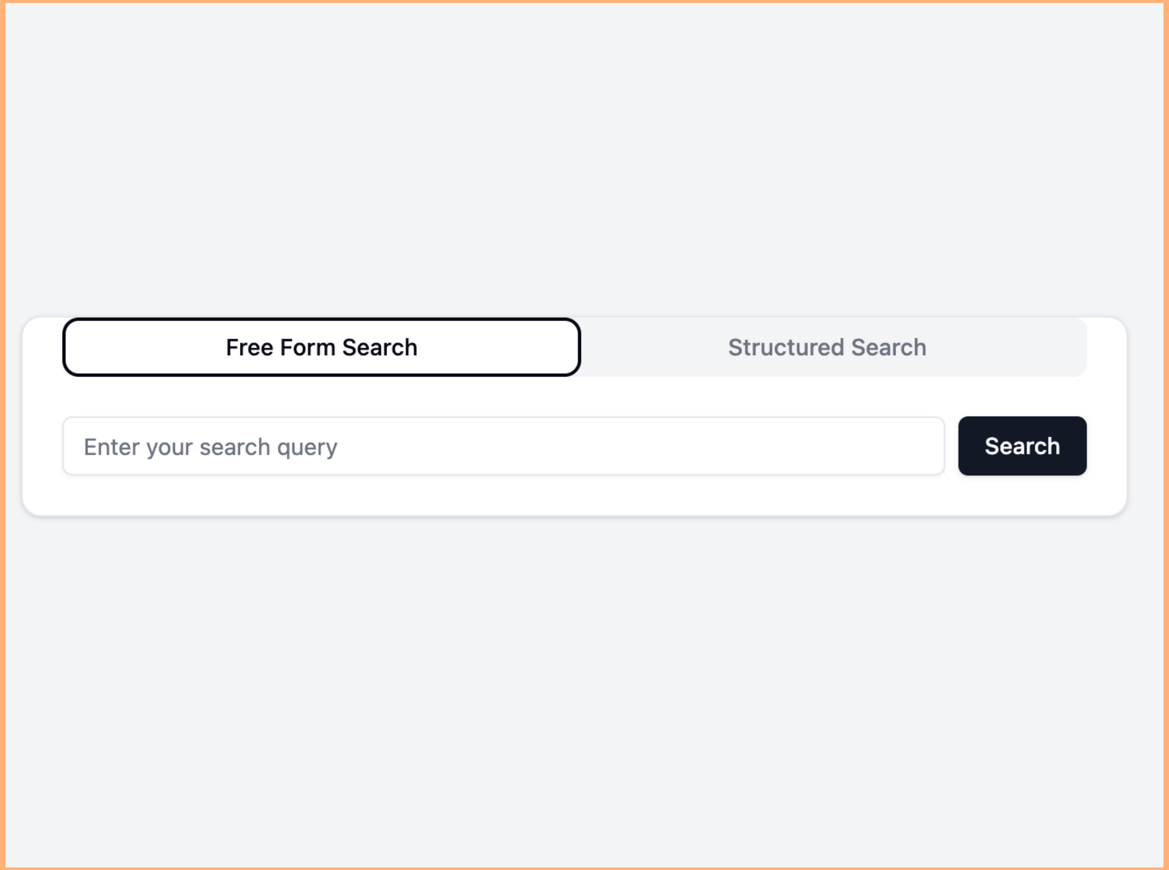
Contracts

Proposals

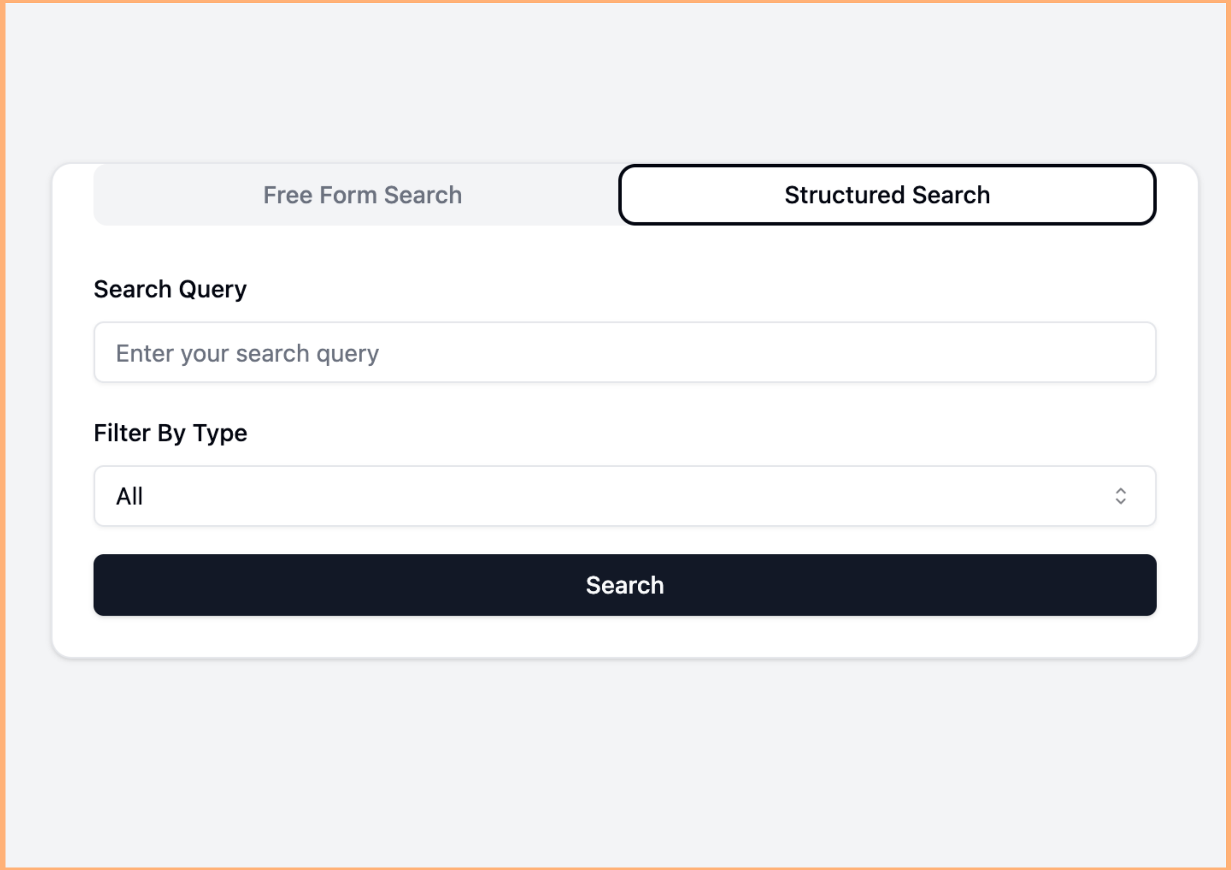
Bids

All ✓

Build for both Humans and AI



Build for both Humans and AI

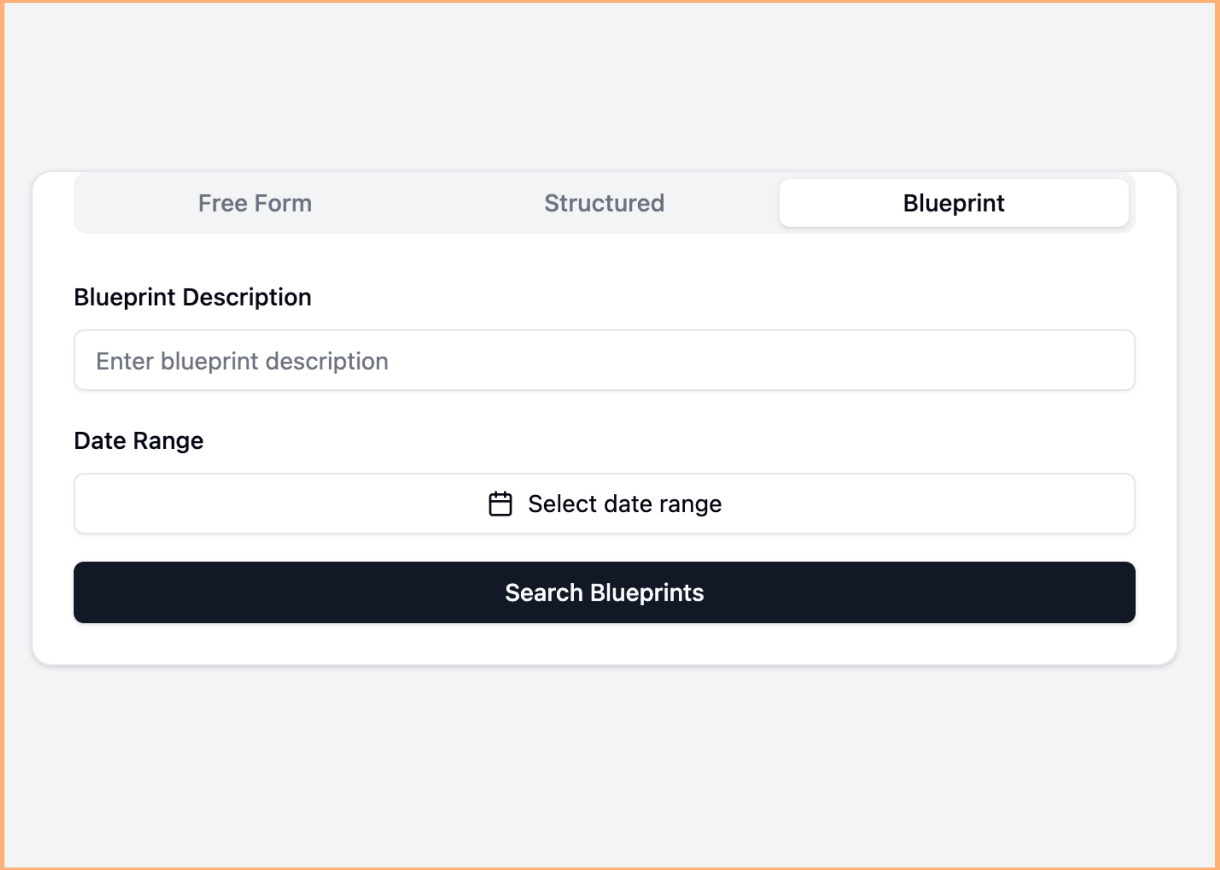


@jxnlco

maven.com/applied-llms/rag-playbook

Build for both Humans and AI

When I want directions, I open maps, if I want videos I visit youtube, don't force users to 'chat with x' when they don't have to



Agenda

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

☐ **Work on Food for thought from last few sessions**

- Generate synthetic data to test your system
- Analyze user queries through topic modeling and use this to inform inventory or capabilities issues
- Implement user feedback mechanisms
- Focus on building more robust retrieval indices based on entity types (e.g., documents, images, text-to-SQL)

☐ **Focus on query routing**

- Ask yourself: if you assume your search methods work well, which search methods would you want to work?
- Should any of these tools be explicitly exposed to the user?
- Can our tools be executed in parallel?

☐ **Additional resources**

- https://python.useinstructor.com/examples/search
- https://python.useinstructor.com/examples/planning-tasks/

Overview

- **Focus for past two sessions**:
  - The importance of segmentation and topic modeling, understanding where there may be a lack of inventory vs. lack of capabilities
  - Review how to extract and improve each specific type of retrieval index (e.g., documents, images, text-to-SQL)
  - Discuss the importance of query routing and how to separate out the query routing step vs. the retrieval index
- **Focus for this session**:
  - Outline tools and example approaches to building a simple query routing and corresponding metrics
  - Review how to use the RAG playbook steps (e.g., generate synthetic data, focus on recall-precision metrics) to tackle the query routing sub-problem while improving RAG applications

Agenda

Query routing vs. retrieval indices

Query routing main challenges

Further testing

UI considerations

Food for thought for this session

Course Conclusion

Course conclusion

What I hope to distill into you

- Evaluations are what you need to understand how to improve.
  - Evaluations are a dataset that can inform your decisions.
  - They power few-shot examples, which can be used with retrieval to improve performance.
  - They are also the finetuning datasets… (Train vs. Test Split)

Course conclusion

What I hope to distill into you

- Evaluations are what you need to understand how to improve.
  - Evaluations are a dataset that can inform your decisions.
  - They power few-shot examples, which can be used with retrieval to improve performance.
  - They are also the finetuning datasets… (Train vs. Test Split)
- Synthetic data with large language models and customer feedback from engagement are two sides of the same coin
  - It's all data and data augmentation.
- These are the fundamental building blocks of creating successful machine learning products.
  - If you refuse to believe this, you're condemning yourself to being lost and confused in this hyped landscape.

Course conclusion

What I hope to distill into you

- Evaluations are what you need to understand how to improve.
  - Evaluations are a dataset that can inform your decisions.
  - They power few-shot examples, which can be used with retrieval to improve performance.
  - They are also the finetuning datasets… (Train vs. Test Split)
- Synthetic data with large language models and customer feedback from engagement are two sides of the same coin
  - It's all data and data augmentation.
- These are the fundamental building blocks of creating successful machine learning products.
  - If you refuse to believe this, you're condemning yourself to being lost and confused in this hyped landscape.

The process:

- A good product generates better evaluations (with strong UX)
- Better evaluations allow you to train / finetune models to create an even better product.
- Data analysis over segments of your evaluations tells you where to focus your product development efforts.

This marks the end of our course.

Please don't hesitate to give any feedback.

My goal is to convey to folks the importance of having strong fundamentals. If you think there's any way this course can be made better for future iterations, please let me know.