

Advanced RAG Concepts



Lukas Spranger

- M.Sc. Computer Science
- Previously at Amazon, Siemens
- Currently ML Tech Lead at SensXPERT
 - ML in industrial contexts
 - LLMs/RAG for knowledge management
 - LLM-powered lead generation
- Personal projects using LLMs

Reach out to talk or collaborate on projects:

<https://www.linkedin.com/in/lukasspranger/>

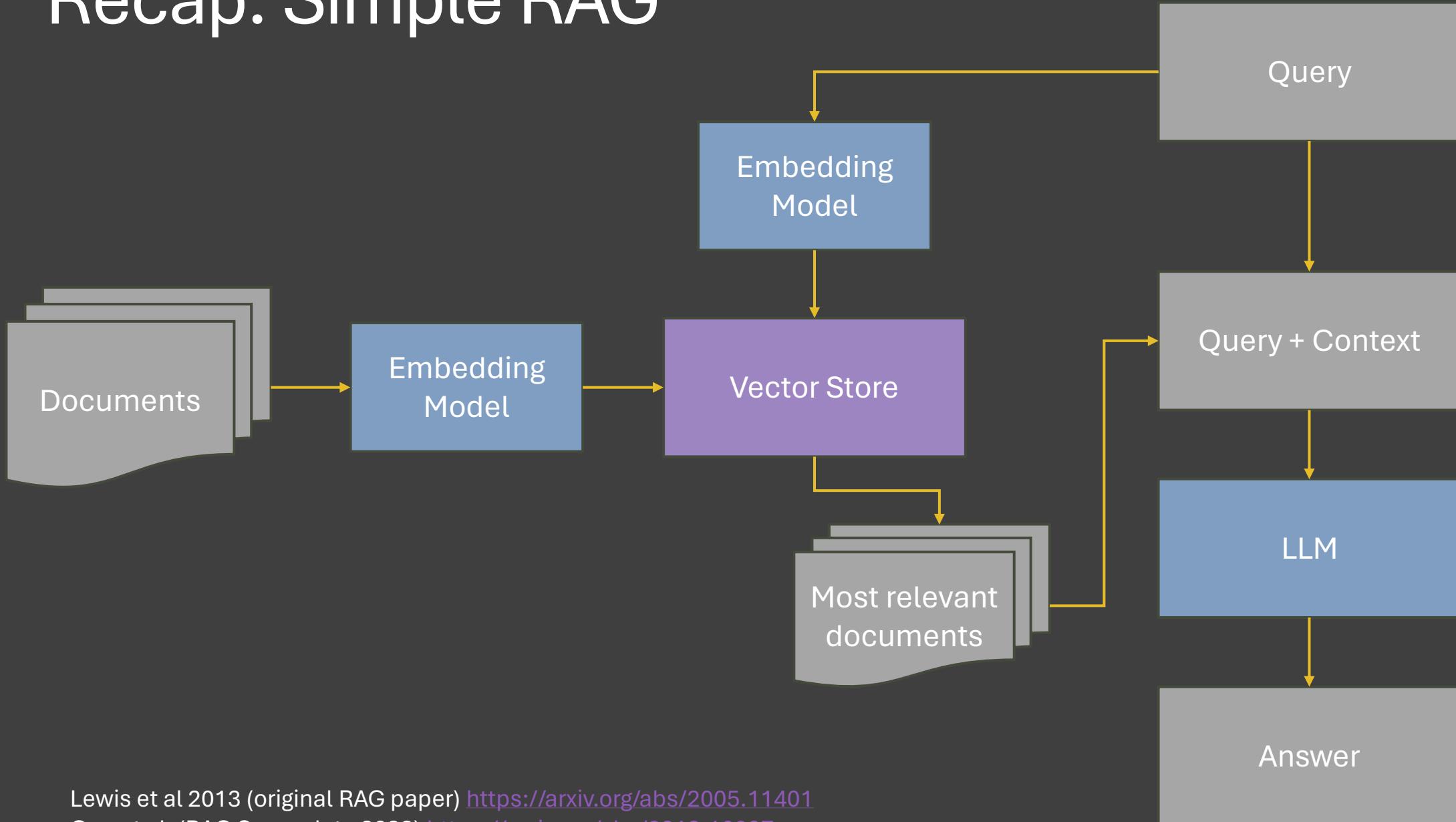
lukas.spranger@gmail.com



2023 Was the Year of RAG

- RAG probably has been the biggest trend in applied AI in 2023
- Many companies have experimented with RAG for internal processes or product features
- Clear value proposition for most companies (knowledge management, chat bots ...)
- It's easy to get started and have a at least somewhat working prototype

Recap: Simple RAG



Lewis et al 2013 (original RAG paper) <https://arxiv.org/abs/2005.11401>

Gao et al. (RAG Survey late 2023) <https://arxiv.org/abs/2312.10997>

Very Simple To Get Started

1. Just get an OpenAI API key
2. Check out my tutorial [RAG From Scratch](#) to build a simple RAG-powered CLI chat bot without any additional frameworks.

```
C:\workspace\rag-tutorial\rag-from-scratch (master -> origin)
(rag) λ python arxivbot.py
Ask me anything about your scientific papers.
What is the cure time of DA 409 as recommended by the manufacturer?

Finding Answer...
1. Obtaining embeddings
100%|██████████| 46/46 [00:00<?, ?it/s]
2. Ranking documents
3. Running queries...
## J_Puentes_dissertation_2017.pdf
page: 51

=====
Answer:
The cure time of DA 409 as recommended by the manufacturer is 1 hour.
=====
```

Very Simple To Get Started

Or use llama-index for very fast prototyping:

```
from llama_index.core import VectorStoreIndex, SimpleDirectoryReader

documents = SimpleDirectoryReader("data").load_data()
index = VectorStoreIndex.from_documents(documents)

query_engine = index.as_query_engine()
response = query_engine.query("What did the author do growing up?")
print(response)
```

Challenges

- Of course, the devil is in the detail and there are several challenges for getting a RAG system production ready
- Reliability of the RAG output
 - Retrieval step
 - Generation step
- Latency
- Cost: Compute and/or API calls

Let's go down the RAG pipeline

- Go through the different stages and design choices
- Of course, they are interrelated with each other, so your choices for one component often effects the others
- We will have a look at some features of LangChain and Llama-Index, but we're not married to a framework
- General concepts and techniques and how to implement them
- We will not be able to cover everything, but some of the “hyperparameters” you can tweak

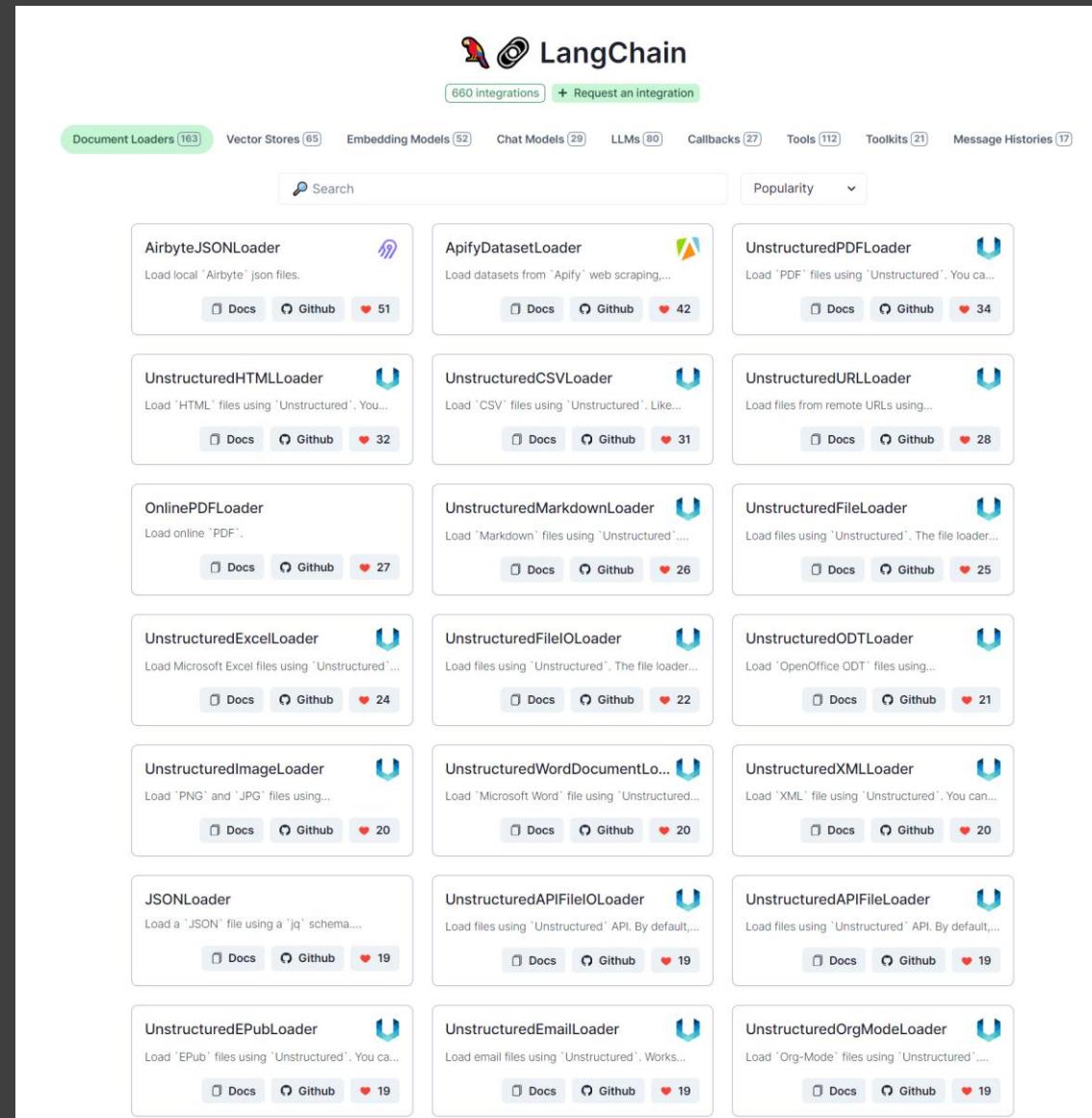


Data Loading & Preprocessing



Data Sources

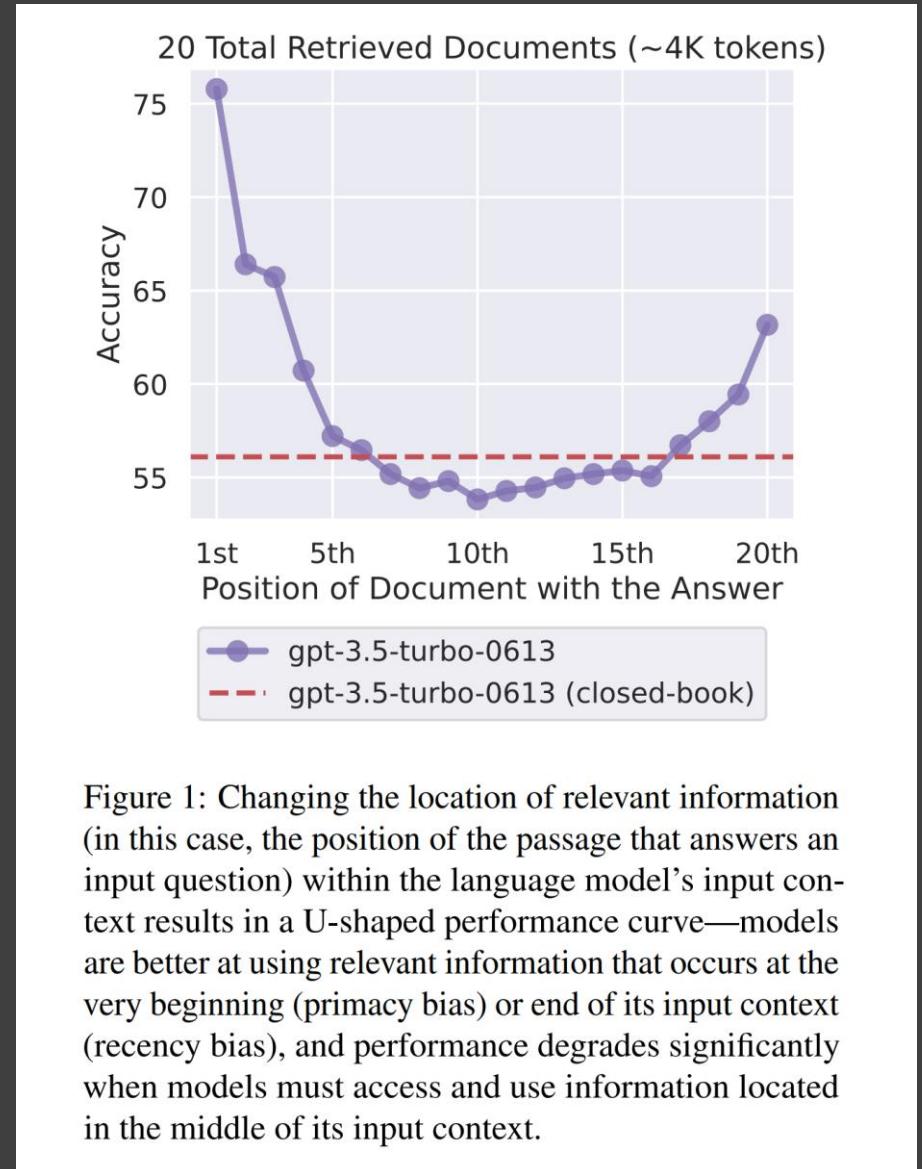
- Which data sources do you have access to
- Which data do you need to solve your problem
- Effects a lot of later decisions
 - One data source vs many that need to be fused
 - How do you choose your data structures
 - How can you use your LLM
- A lot of data loaders and integration are available open source via Llama Hub or Langchain
- Don't be afraid to write your own. Often you need something bespoke



<https://integrations.langchain.com>

Chunking (Text Splitting)

- LLMs have a fixed context window
- Large context windows more costly
- You need to chunk your data for best results and efficiency
- Lost in the middle phenomenon (Liu et al., <https://arxiv.org/abs/2307.03172>)
- Your chunks should be well chosen and relevant (this starts with the chunking algorithm)
- Your chunking strategy is an important hyperparameter you can, and should, tweak



Use Document Format for Chunking

- The format, or structure, of a document encodes semantic information
- We can use the format of a document to split it into chunks
- Examples:
 - Markdown headers and sections
 - Word document chapters and headings
 - Powerpoint slides

```
### A GPU-Ready Tensor Library

If you use NumPy, then you have used Tensors (a.k.a. ndarray).

![Tensor illustration](./docs/source/_static/img/tensor_illustration.png)

PyTorch provides Tensors that can live either on the CPU or the GPU and accelerates the computation by a huge amount.

We provide a wide variety of tensor routines to accelerate and fit your scientific computation needs such as slicing, indexing, mathematical operations, linear algebra, reductions. And they are fast!

### Dynamic Neural Networks: Tape-Based Autograd

PyTorch has a unique way of building neural networks: using and replaying a tape recorder.

Most frameworks such as TensorFlow, Theano, Caffe, and CNTK have a static view of the world. One has to build a neural network and reuse the same structure again and again. Changing the way the network behaves means that one has to start from scratch.

With PyTorch, we use a technique called reverse-mode auto-differentiation, which allows you to change the way your network behaves arbitrarily with zero lag or overhead. Our inspiration comes from several research papers on this topic, as well as current and past work such as [torch-autograd](https://github.com/twitter/torch-autograd), [autograd](https://github.com/HIPS/autograd), [Chainer](https://chainer.org), etc.

While this technique is not unique to PyTorch, it's one of the fastest implementations of it to date. You get the best of speed and flexibility for your crazy research.
```

Example: LangChain Text Splitters

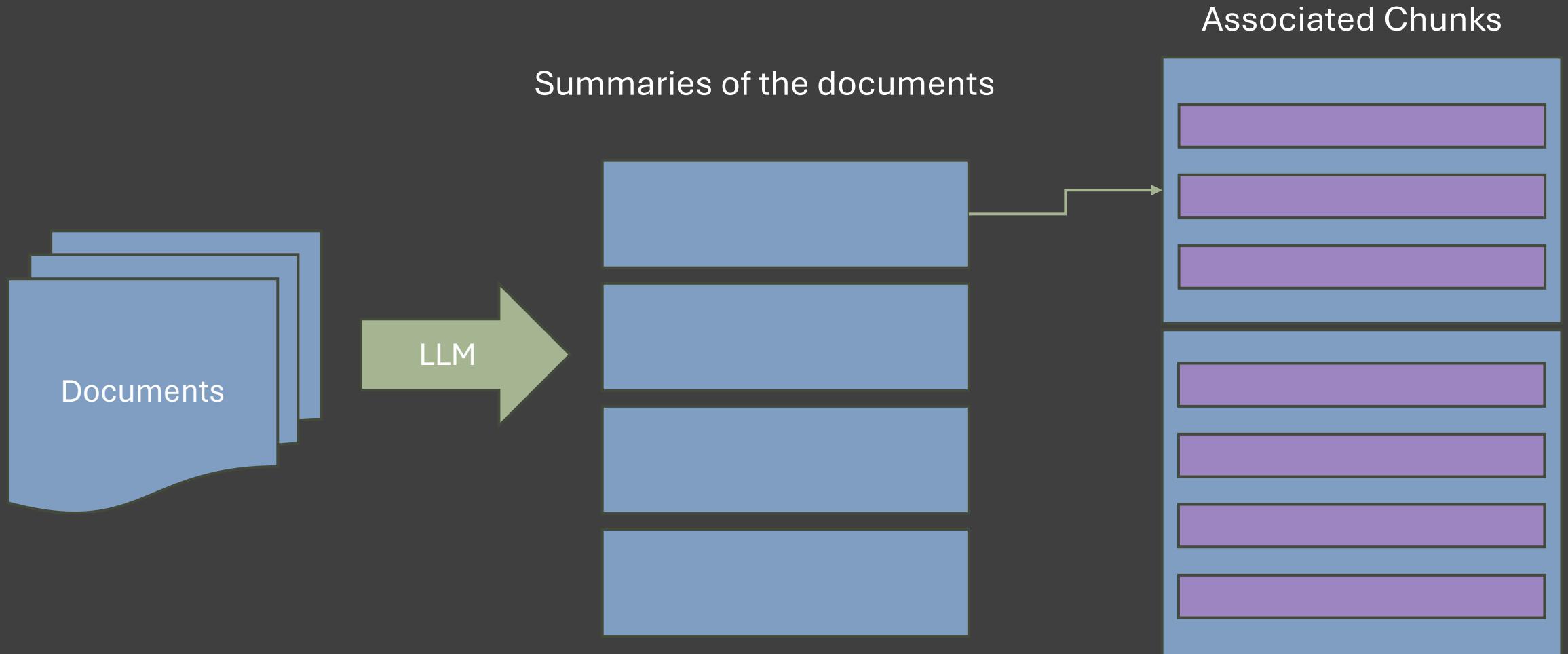
- LLM frameworks like LangChain implement several chunking approaches
- LangChain TextSplitters ([link](#))

Name	Splits On	Adds Metadata	Description
Recursive	A list of user defined characters		Recursively splits text. Splitting text recursively serves the purpose of trying to keep related pieces of text next to each other. This is the recommended way to start splitting text.
HTML	HTML specific characters	<input checked="" type="checkbox"/>	Splits text based on HTML-specific characters. Notably, this adds in relevant information about where that chunk came from (based on the HTML)
Markdown	Markdown specific characters	<input checked="" type="checkbox"/>	Splits text based on Markdown-specific characters. Notably, this adds in relevant information about where that chunk came from (based on the Markdown)
Code	Code (Python, JS) specific characters		Splits text based on characters specific to coding languages. 15 different languages are available to choose from.
Token	Tokens		Splits text on tokens. There exist a few different ways to measure tokens.
Character	A user defined character		Splits text based on a user defined character. One of the simpler methods.
[Experimental] Semantic Chunker	Sentences		First splits on sentences. Then combines ones next to each other if they are semantically similar enough. Taken from Greg Kamradt



Further
Preprocessing
before
Embedding

Store LLM Summaries of documents

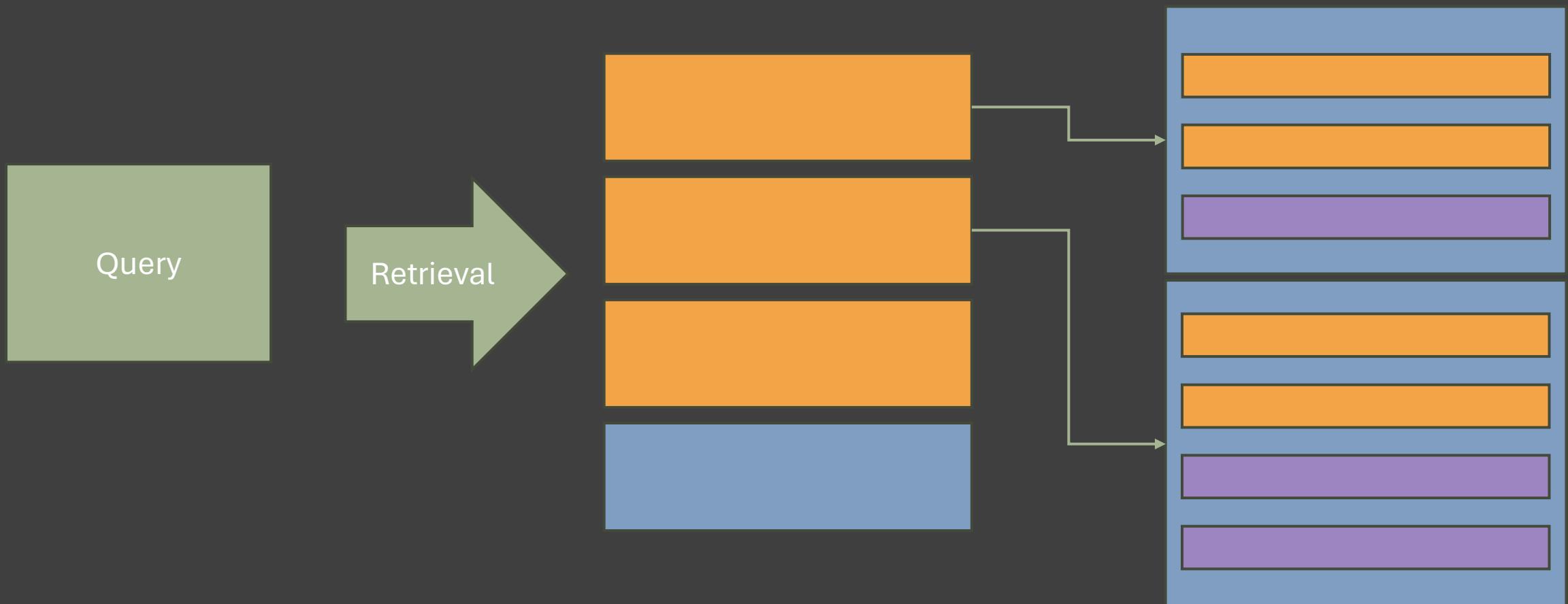


Store LLM Summaries of documents

Two stage retrieval process

Top N relevant documents
based on summary

Top K relevant chunks inside
the relevant documents

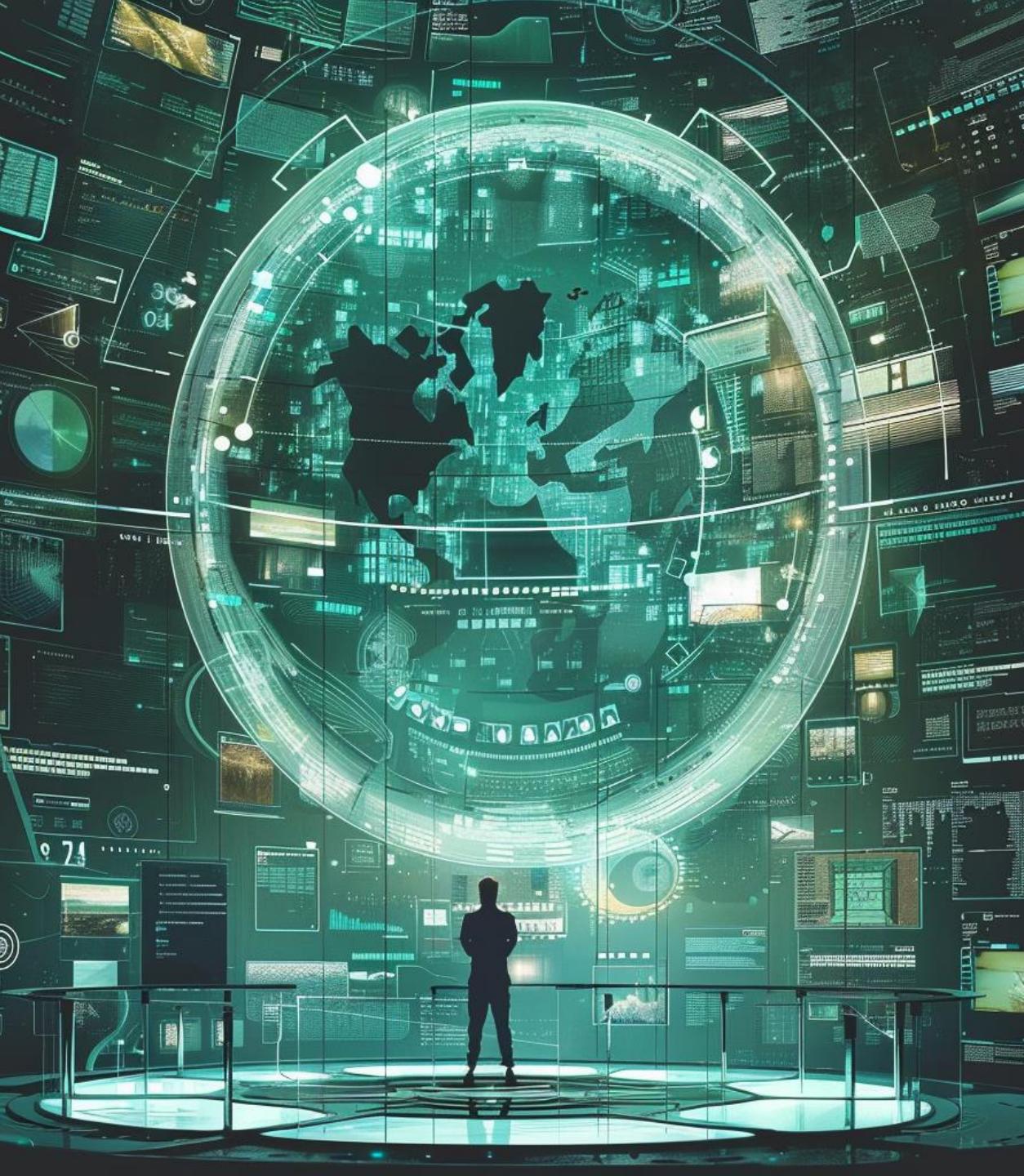


Other Types of LLM Preprocessing

Some more ideas:

- Generate Q&A for each chunk and store them
- Rewrite it to certain format/style
- Unstructured data -> Structured data
- Tables to text
- (Later: We can also rewrite the prompt/query!)

...



Retrieval: Data Structures and Algorithms

The Modern Classic: Vector Search

1. Compute vector embeddings for all your documents using a suitable model
2. Store those embeddings
3. At runtime, embed your query and find the closest k documents (chunks) w.r.t their embeddings
4. Use the retrieved documents to generate the answer to the query (or whatever output is desired – there are a lot of applications beyond Q&A)

<https://huggingface.co/blog/getting-started-with-embeddings>

<https://huggingface.co/blog/mteb>

Embeddings

Embeddings: One-vector semantic summary of a document produced by the LLM

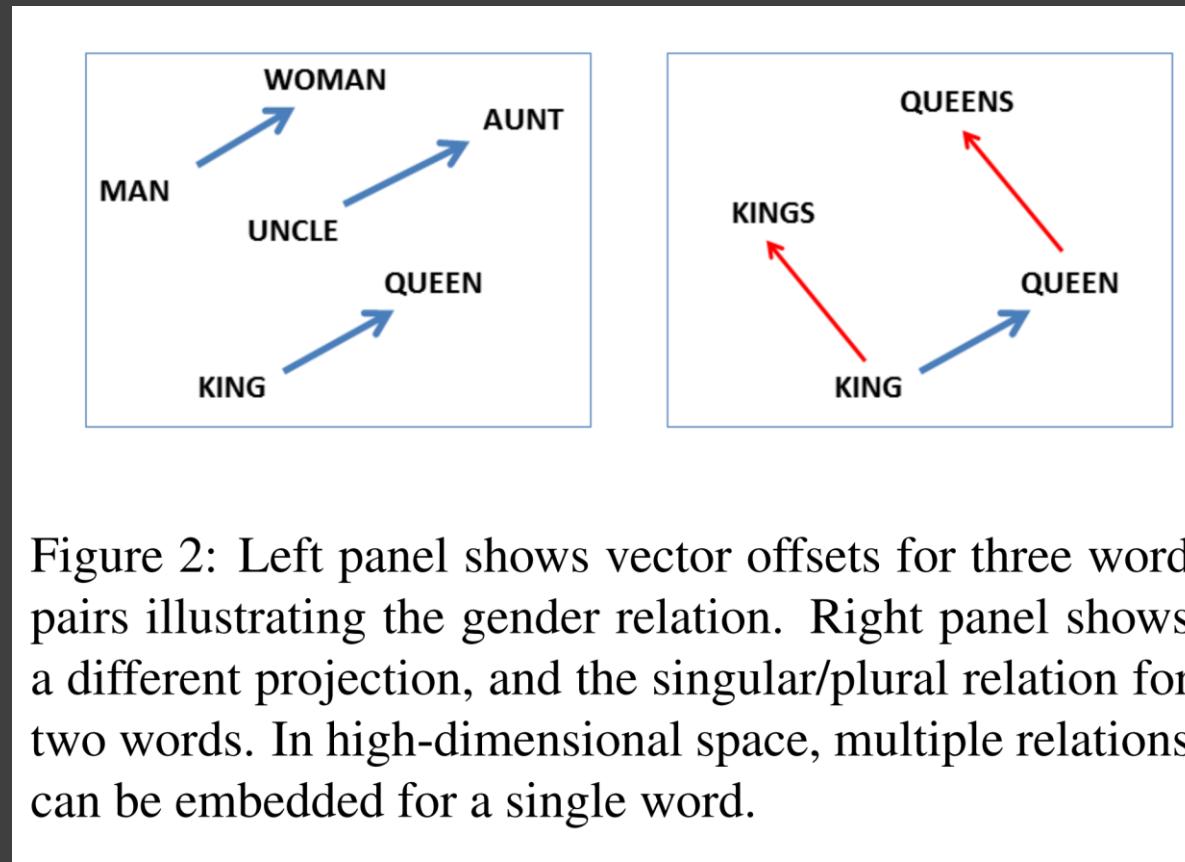


Figure 2: Left panel shows vector offsets for three word pairs illustrating the gender relation. Right panel shows a different projection, and the singular/plural relation for two words. In high-dimensional space, multiple relations can be embedded for a single word.

Image from Mikolov et al. 2013: <https://aclanthology.org/N13-1090/>
<https://huggingface.co/blog/getting-started-with-embeddings>

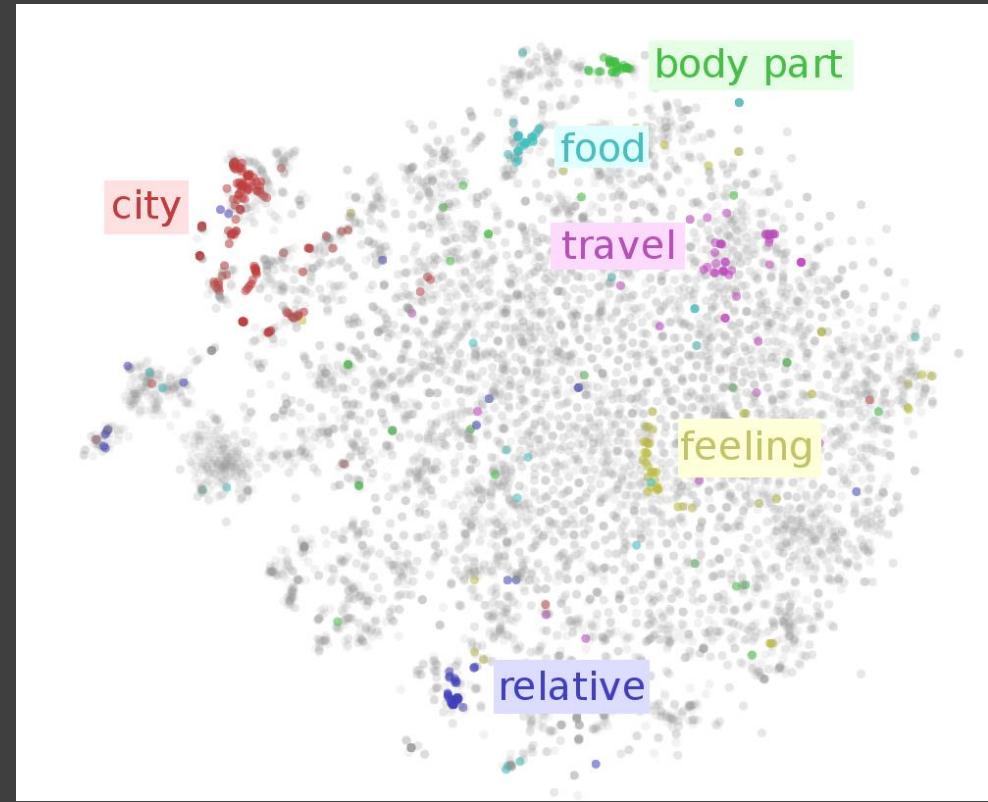
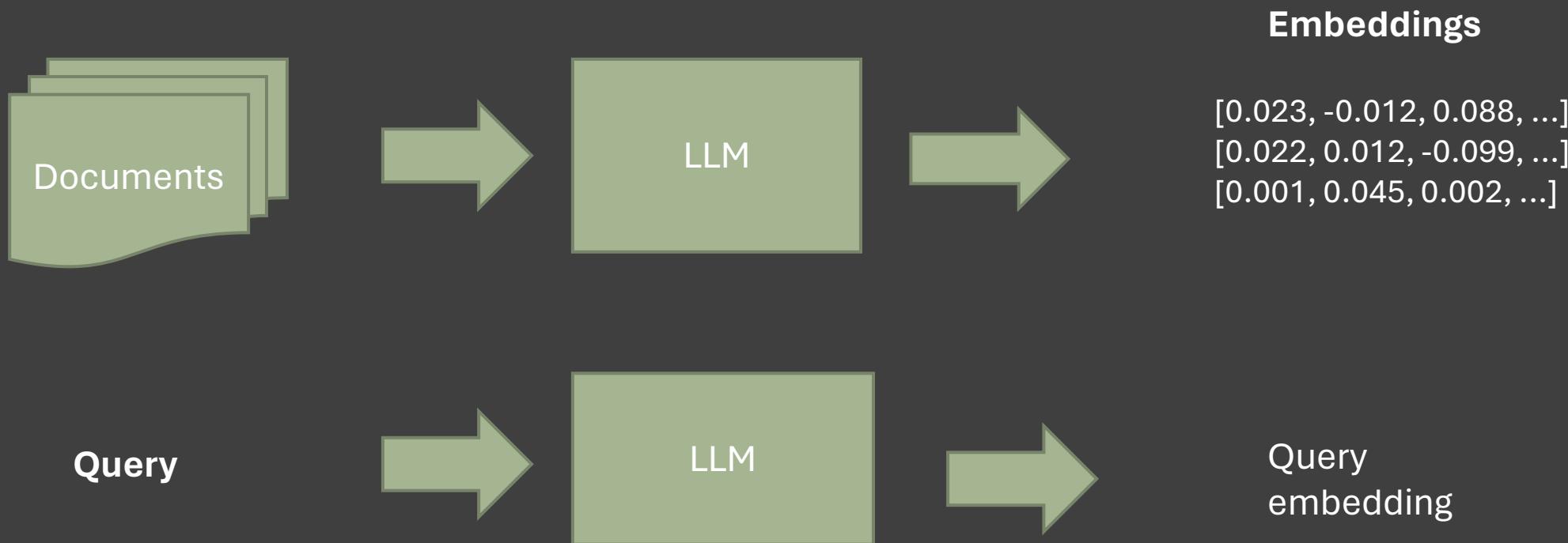


Image from: <https://www.ruder.io/word-embeddings-1/>

Retrieval Using LLMs



Rank documents based on distance to query embedding (dot product, cosine distance)

The Embedding Model

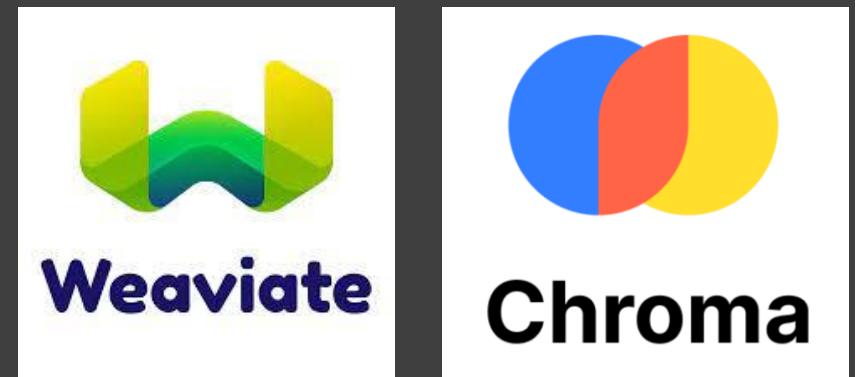
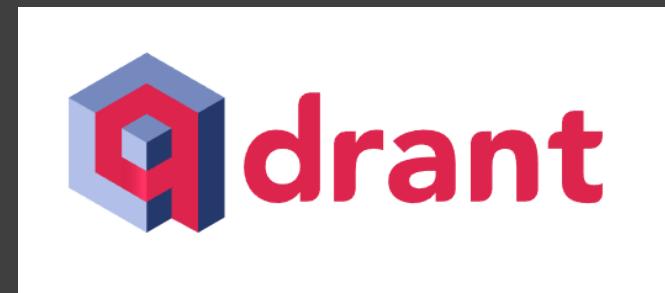
- There are a lot of good open source models and proprietary models out there
- Benchmark MTEB
- A lot of tutorials use OpenAI's ada-002, but there are many better models (currently ranked 38th on MTEB)
- Consider also embedding dimension and context window size when choosing a model
- Later you can even fine tune an embedding model for your specific use case
- Probably not the most important thing to optimize early on

Overall MTEB English leaderboard 🧠					
Rank	Model	Model Size (GB)	Embedding Dimensions	Max Tokens	Average (56 datasets)
1	SFR-Embedding-Mistral	14.22	4096	32768	67.56
2	voyage-lite-02-instruct		1024	4000	67.13
3	GritLM-7B	14.48	4096	32768	66.76
4	e5-mistral-7b-instruct	14.22	4096	32768	66.63
5	GritLM-8x7B	93.41	4096	32768	65.66
6	UAE-Large-V1	1.34	1024	512	64.64
7	text-embedding-3-large		3072	8191	64.59
8	voyage-lite-01-instruct		1024	4000	64.49
9	Cohere-embed-english-v3.0		1024	512	64.47
10	multilingual-e5-large-instruct	1.12	1024	514	64.41
11	GIST-large-Embedding-v0	1.34	1024	512	64.34

Massive Text Embedding Benchmark (MTEB) Leaderboard
<https://huggingface.co/spaces/mteb/leaderboard>

Vector Databases

- A lot of vector databases have become popular:
Open/closed source, managed/self hosted
- Implement storage and similarity search
(approximate NN)
- Support for other retrieval methods, metadata
- Note: Do you actually need a vector database for your prototype?



Other Search Indices / Databases

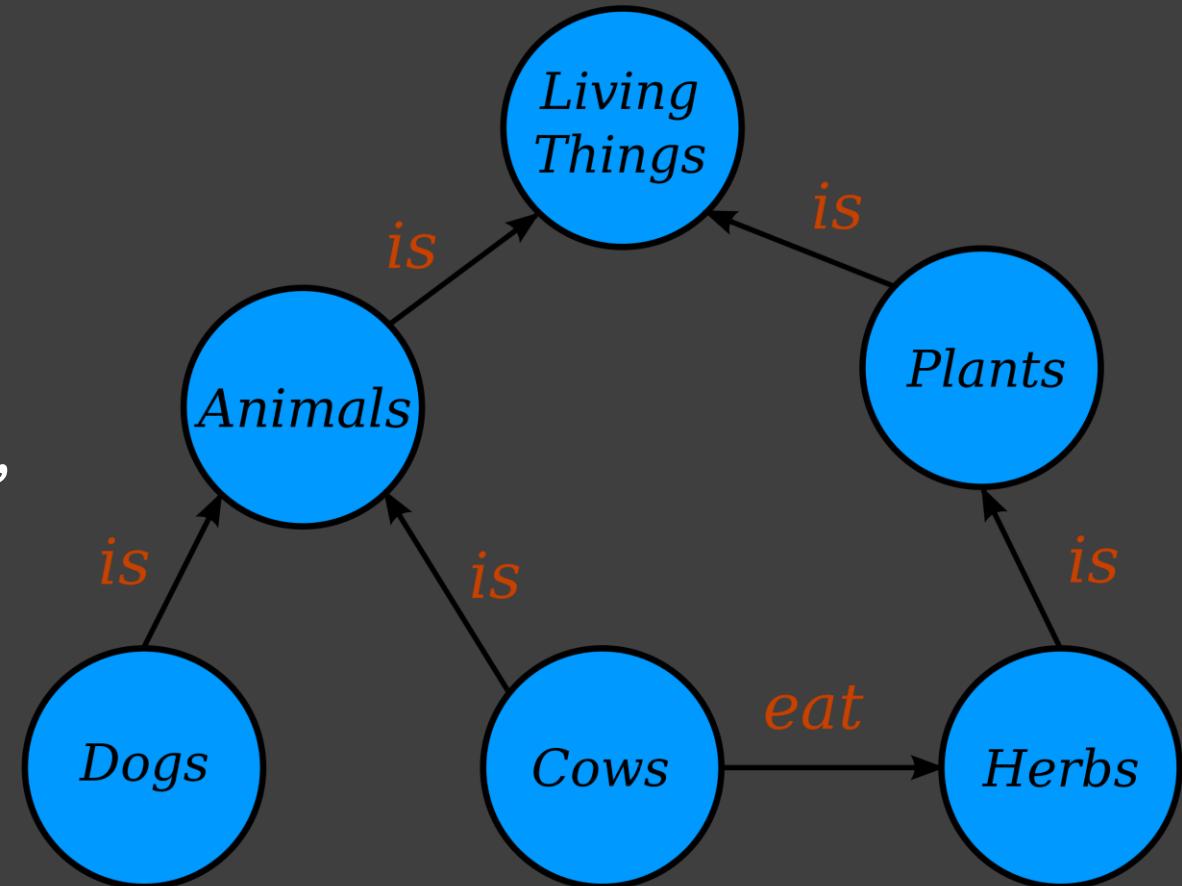
- Graph Databases / Knowledge Graph
- SQL/NOSQL Databases
- APIs
- Web Search

Your choice depends on your use case and existing data infrastructure

Knowledge Graph Index

Different use cases:

- Existing knowledge graph
- Construct graph using LLM
- Construct graph from metadata, contextual information



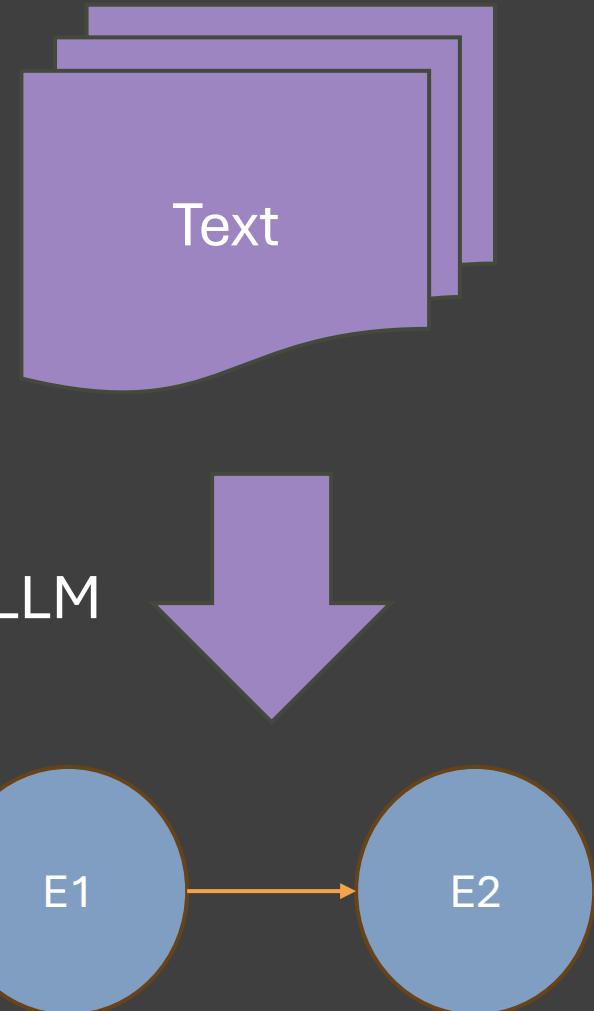
Construct graph using LLM

Knowledge Graph:

- Nodes: Entities of the domain modeled
- Edges: Relationships of the domain

Can also be described as <subject> <predicate> <object>

Extract entities and relationships from text



GraphIndex in Llama-Index

Supported Graph Stores:

- Nebula
- Neo4J
- Kuzu
- FalkorDB

```
from llama_index.core import KnowledgeGraphIndex

kg_index = KnowledgeGraphIndex.from_documents(
    documents,
    storage_context=storage_context,
    max_triplets_per_chunk=10,
    space_name=space_name,
    edge_types=edge_types,
    rel_prop_names=rel_prop_names,
    tags=tags,
    include_embeddings=True,
)
```

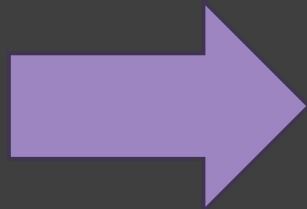
https://docs.llamaindex.ai/en/stable/community/integrations/graph_stores.html

https://docs.llamaindex.ai/en/stable/examples/query_engine/knowledge_graph_query_engine.html

LLM Writes Cypher Query Language

Raw input

“Tell me about
Gandalf the White!”

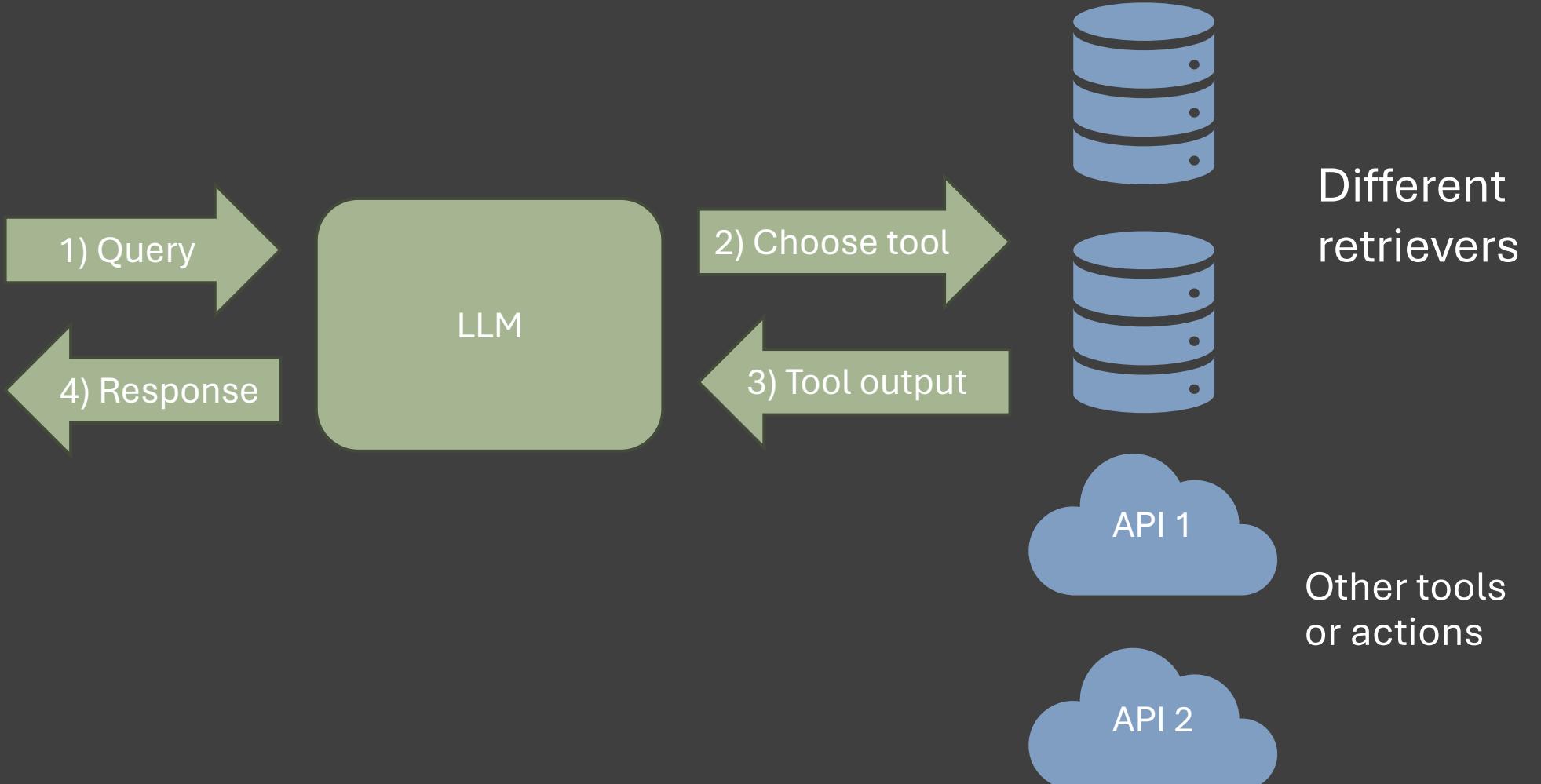


Query Language (Cypher)

```
MATCH (p:`entity`)-[:relationship]->(m:`entity`)  
WHERE p.`entity`. `name` == 'Gandalf the White'  
RETURN p.`entity`. `name`;
```

- This also works for SQL or other query languages (e.g., Jira query language)
- For query languages with low support in training data, you might need to fine-tune or use few-shot learning

Choosing the right tool for the job



Give LLMs Access To Tools / APIs

- OpenAI Function Calling
- OpenAI Assistant Tools
- LangChain tools
(built-in, custom)

[Link to Assistant](#)

```
assistant = client.beta.assistants.create(  
    instructions="You are a weather bot. Use the provided functions to answer questions.",  
    model="gpt-4-turbo-preview",  
    tools=[  
        {  
            "type": "function",  
            "function": {  
                "name": "getCurrentWeather",  
                "description": "Get the weather in location",  
                "parameters": {  
                    "type": "object",  
                    "properties": {  
                        "location": {"type": "string", "description": "The city and state e.g. San Francisco, CA"},  
                        "unit": {"type": "string", "enum": ["c", "f"]}  
                    },  
                    "required": ["location"]  
                }  
            }  
        }, {  
            "type": "function",  
            "function": {  
                "name": "getNickname",  
                "description": "Get the nickname of a city",  
                "parameters": {  
                    "type": "object",  
                    "properties": {  
                        "location": {"type": "string", "description": "The city and state e.g. San Francisco, CA"},  
                    },  
                    "required": ["location"]  
                }  
            }  
        }  
    ]  
)
```

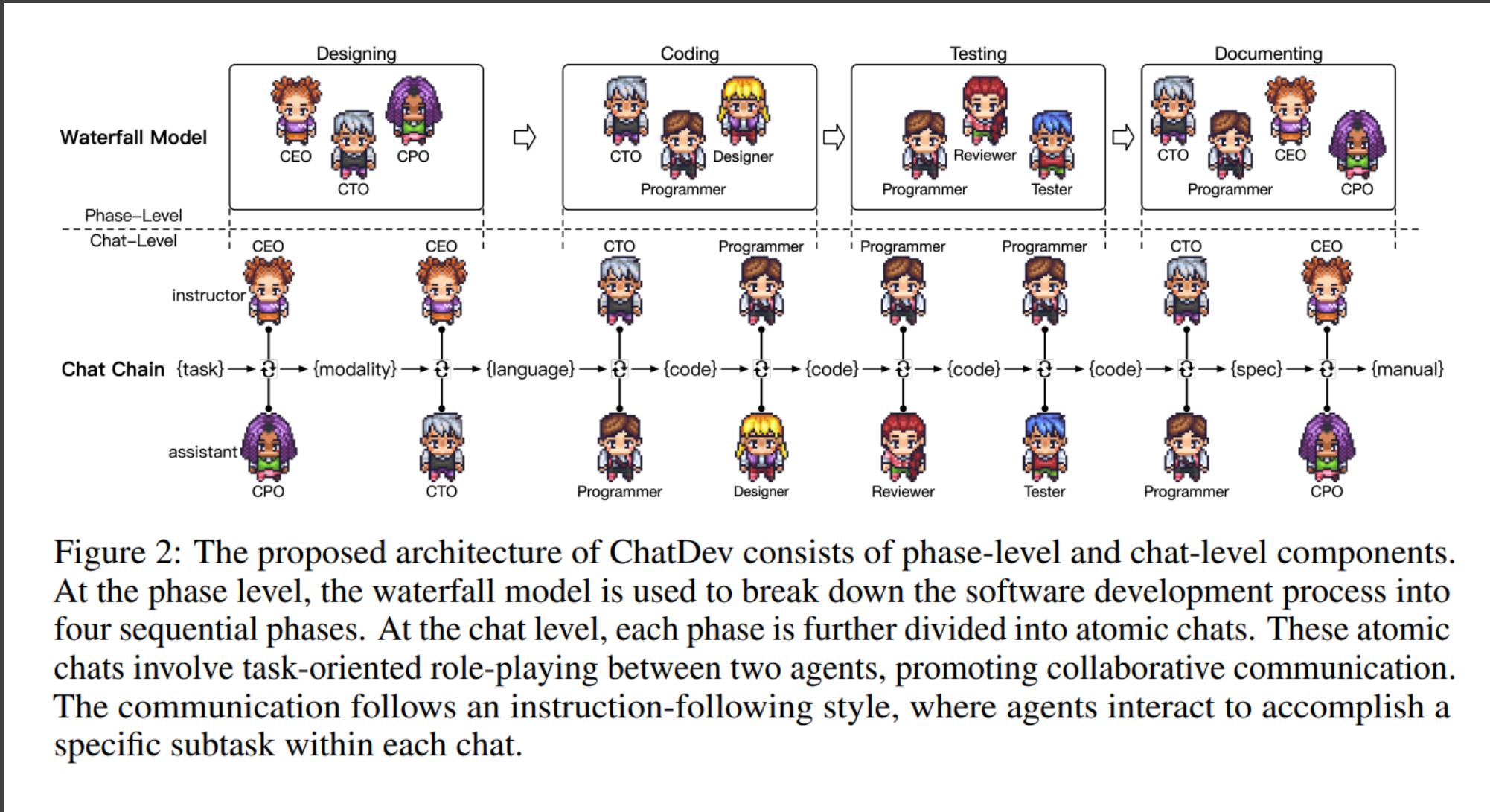
Using an API to retrieve context

```
{  
  "name": "search_jira_issues",  
  "description": "Retrieves issues from a jira client by parsing a JQL query",  
  "parameters": {  
    "type": "object",  
    "properties": {  
      "jql_str": {  
        "type": "string",  
        "description": "JQL string that is forwarded to the jira api to search for issues"  
      }  
    },  
    "required": ["jql_str"]  
  }  
}
```

Towards Agents

- A “tool” or function can of course use again powered by an LLM.
- Allow LLMs to use tools like APIs and talk to each other

ChatDev: Agents for Software Development



ChatDev

- Different specialized agents solving problems and writing code inside a chat
- Qian et al. 2023:
<https://arxiv.org/abs/2307.07924>

Coding: Code Generation

 CTO (Chief Technology Officer)  Programmer

 We have decided to complete the task through a executable software with multiple files implemented via Python. You should write one or multiple files and make sure that every detail of the architecture is implemented as code. The software should be equiped with graphical user interface (GUI); so you must choose a suitable GUI framework. Think step by step and reason yourself to the right decisions to make sure we get it right. You will first lay out the names of the core classes, functions, methods that will be necessary, as well as a quick comment on their purpose.

 To design a basic Gomoku game in Python with a GUI, we will need the following core classes, functions, and methods:

- Board: This class represents the game board and handles the logic for placing and checking moves.
- Game: This class manages the game flow and user interactions.
- GUI: This class handles the graphical user interface using a GUI framework (e.g., tkinter).

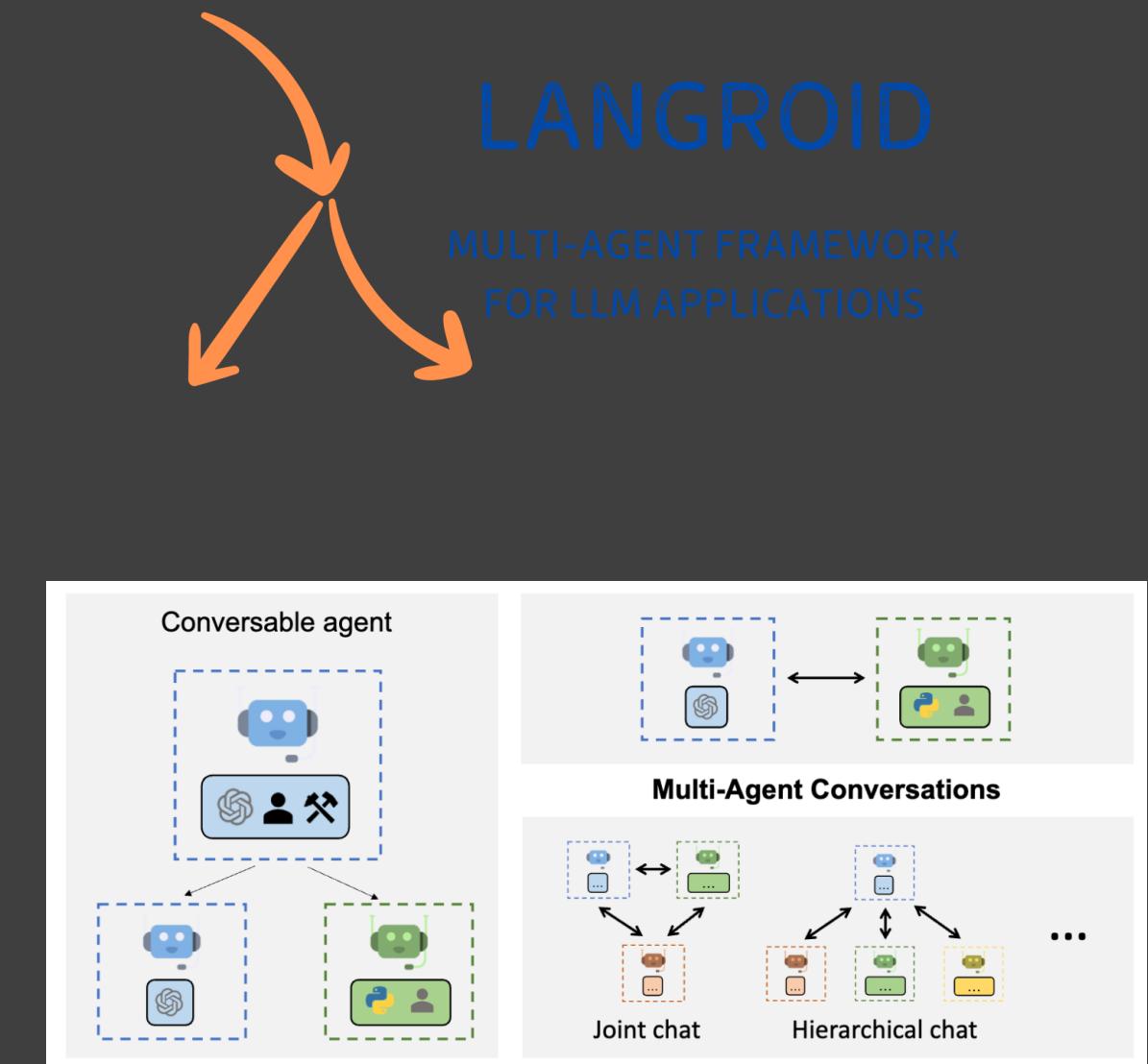
Now, let's proceed with the implementation of each file.

game.py

```
class Game:  
    def __init__(self):  
        self.board = [[0] * 15 for _ in range(15)]  
        self.current_player = 1  
        self.game_over = False  
    def make_move(self, row, col):  
        if self.board[row][col] != 0 or self.game_over:  
            return  
        self.board[row][col] = self.current_player  
        if self.check_win(row, col):  
            self.game_over = True  
        elif self.is_board_full():  
            self.game_over = True  
        self.current_player = 3 - self.current_player  
    ...
```

Implementing Agents

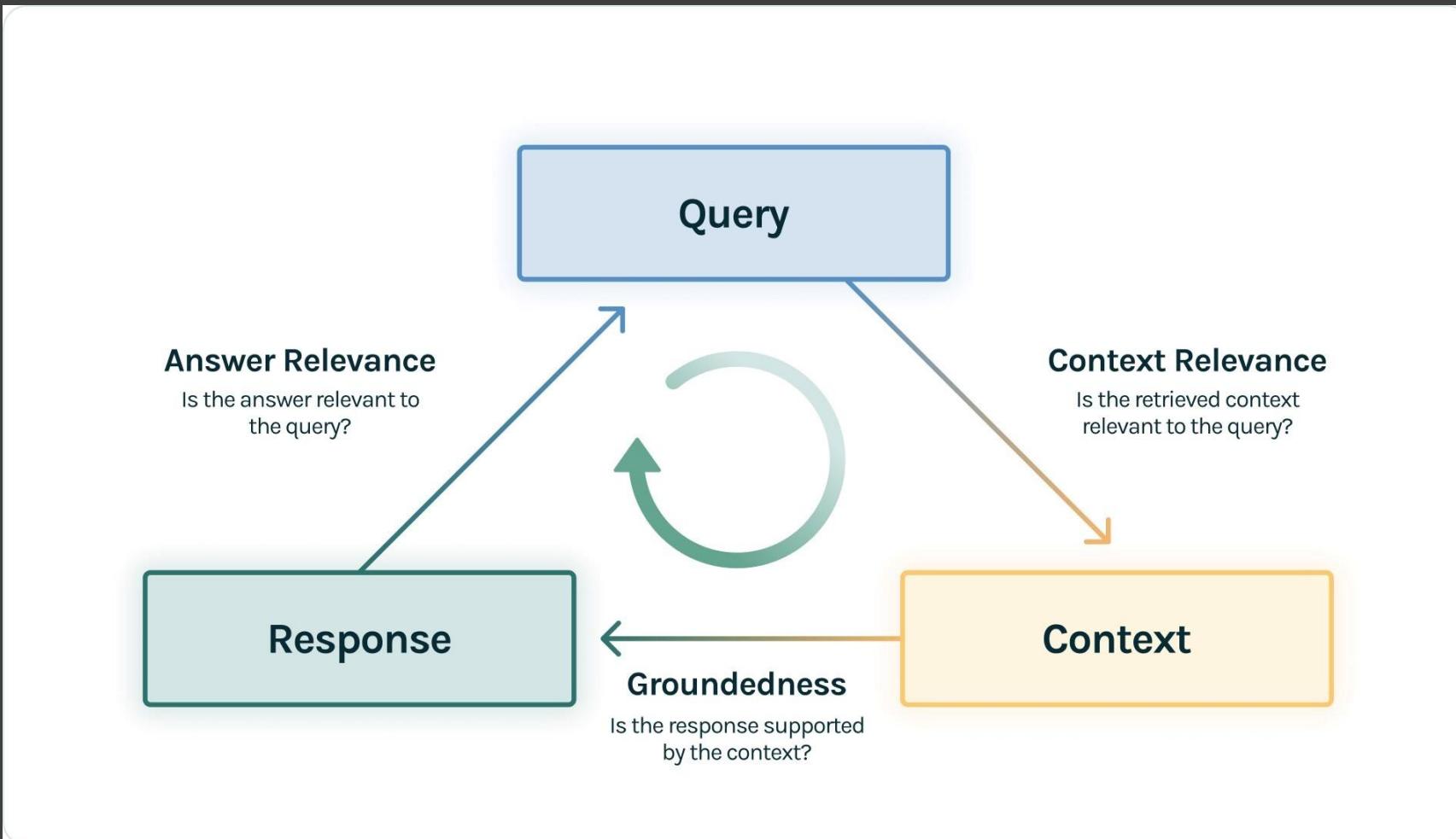
- OpenAI assistants can be a good starting point, as it handles thread management
- AutoGen
- Langroid



Evaluation of RAG Systems

- Treat this as seriously as evaluation of a classical ML / data science problem
- Evaluating the retrieval algorithm
- Evaluating the overall output of the RAG system
- Often done: Human eval, but that doesn't scale!
- Use LLMs to evaluate the output
- Evolving field, many new tools and companies
- Ragas: <https://docs.ragas.io/en/stable/>
- Trulens: <https://www.trulens.org/>

RAG Triad (Trulens)



https://www.trulens.org/trulens_eval/core_concepts_rag_triad/

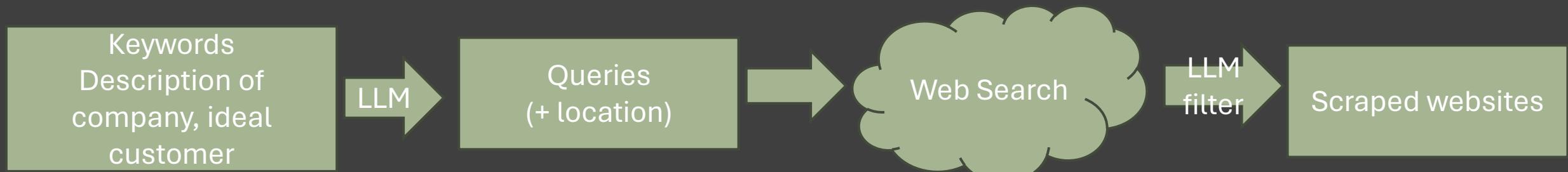
https://www.trulens.org/trulens_eval/quickstart/#set-up-feedback-functions

Case Study: Lead Generation

- “Can we get leads automatically with AI?”
- “But only the ones working with certain materials and processes!”
- “They should be grouped geographically”



Case Study: Lead Generation



UI for human rating



Applications of RAG

- Semantic Search, Chatbot
- Lead Generation
- Customized Outreach
- Create Marketing material
- RAG as a general tool for LLM agents
 - Programming, Software Dev
 - Research and Summarization
 - Active Knowledge Management
- Games?
- Interactive Fiction?



RAG in the Future – Another Bitter Lesson?

- There are a lot of exciting opportunities for RAG now and in the foreseeable future, but...
- Cost per token dropping; LLMs scaling
- Can we solve the “lost in the middle” problem?
- We might be in for another bitter lesson.
- RAG might at least get a lot simpler in the near future
- Nevertheless, it will teach us principles how to use LLMs as reasoning engines within larger systems and ensembles of AI agents



Links and Resources

- Lewis et al 2013 (original RAG paper) <https://arxiv.org/abs/2005.11401>
- RAG From Scratch Tutorial: <https://github.com/luspr/rag-tutorial>
- Gao et al. (RAG Survey late 2023) <https://arxiv.org/abs/2312.10997>
- OpenAI Cookbook: <https://cookbook.openai.com/>
- OpenAI prompt engineering guide: <https://platform.openai.com/docs/guides/prompt-engineering>
- Prompt examples and templates: <https://smith.langchain.com/hub>
- RAG Triad Evaluation: https://www.trulens.org/trulens_eval/core_concepts_rag_triad/
- Chunking and vectorization on LLama index:
https://docs.llamaindex.ai/en/latest/module_guides/loading/ingestion_pipeline/root.html#
- Llama-index Graph: Knowledge Graph Query Engine:
https://docs.llamaindex.ai/en/stable/examples/query_engine/knowledge_graph_query_engine.html
- Communicative Agents for Software Development: <https://arxiv.org/pdf/2307.07924.pdf>
- Lost in the Middle: <https://arxiv.org/abs/2307.03172>

Links and Resources

- Greg Kamradt: 5 Levels Of Text Splitting, https://github.com/FullStackRetrieval-com/RetrievalTutorials/blob/main/5_Levels_Of_Text_Splitting.ipynb
- https://docs.llamaindex.ai/en/stable/community/integrations/graph_stores.html
- https://docs.llamaindex.ai/en/stable/examples/query_engine/knowledge_graph_query_engine.html
- <https://www.pinecone.io/learn/retrieval-augmented-generation/>
- <https://weaviate.io/developers/weaviate/search/hybrid>
- <https://weaviate.io/blog/hybrid-search-explained>
- https://www.trulens.org/trulens_eval/core_concepts_rag_triad/
- <https://github.com/microsoft/autogen>
- <https://github.com/langroid/langroid>

Going Deeper on Chunking

One can go very deep on Chunking

Don't hesitate to write your own text splitter for your specific use case, it's often worth it!

Greg Kamradt: 5 Levels Of Text Splitting ([Notebook Link](#))

1. Character Splitting - Simple static character chunks of data
2. Recursive Character Text Splitting - Recursive chunking based on a list of separators
3. Document Specific Splitting - Various chunking methods for different document types (PDF, Python, Markdown)
4. Semantic Splitting - Embedding walk based chunking
5. Agentic Splitting - Experimental method of splitting text with an agent-like system. Good for if you believe that token cost will trend to \$0.00

Context matters!

- Naïve RAG misses contextual information about the document
- We often cannot answer more complex queries
- Use metadata and contextual information when Loading data
- Implications for your data structures and search index

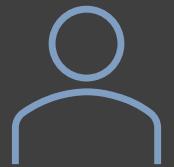
Project X



Team A



Team B



Document for our
RAG system



Cloud Storage

Processing the query

Straightforward approach:

- Retrieve documents based on the query
- Insert context in prompt template with instructions and query
- There are more fancy things you can try to improve performance

```
prompt = """<Instructions for use case>
{query}

# Context
{context}
"""

query = input()
retrieved_docs = retriever.retrieve_docs(query)

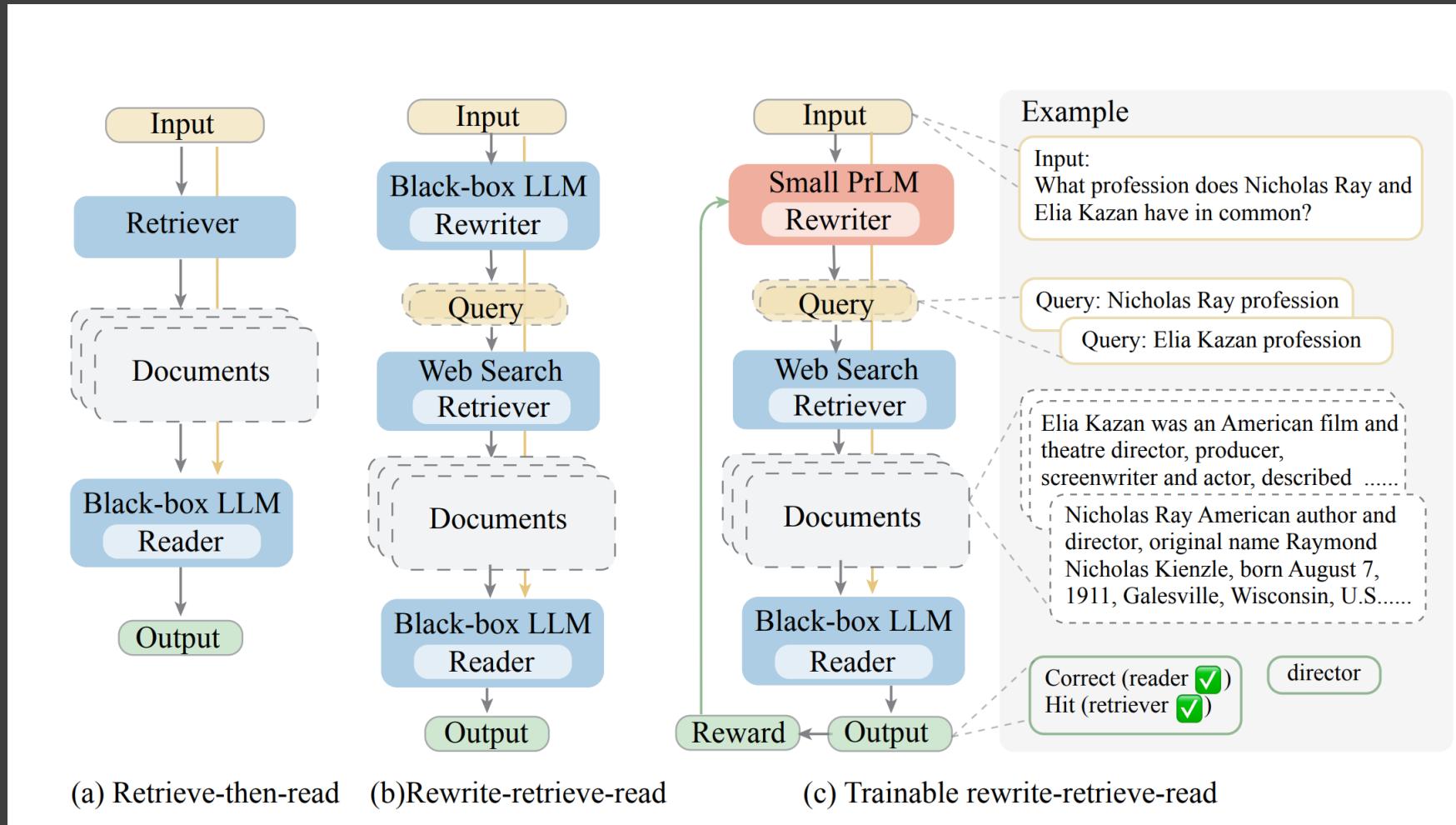
context = '\n\n'.join([d for d in retrieved_docs])
prompt.format(query=query, context=context)
```

Rewriting the query using LLM

Use LLMs to reason about and rewrite the query

1. Write different versions of one query and run RAG for all of them
2. Extract subqueries and search for those individually
3. Write query in a particular format / style

Trainable query rewriting (Ma et al. 2023)



Hybrid Retrieval 1: Hybrid search

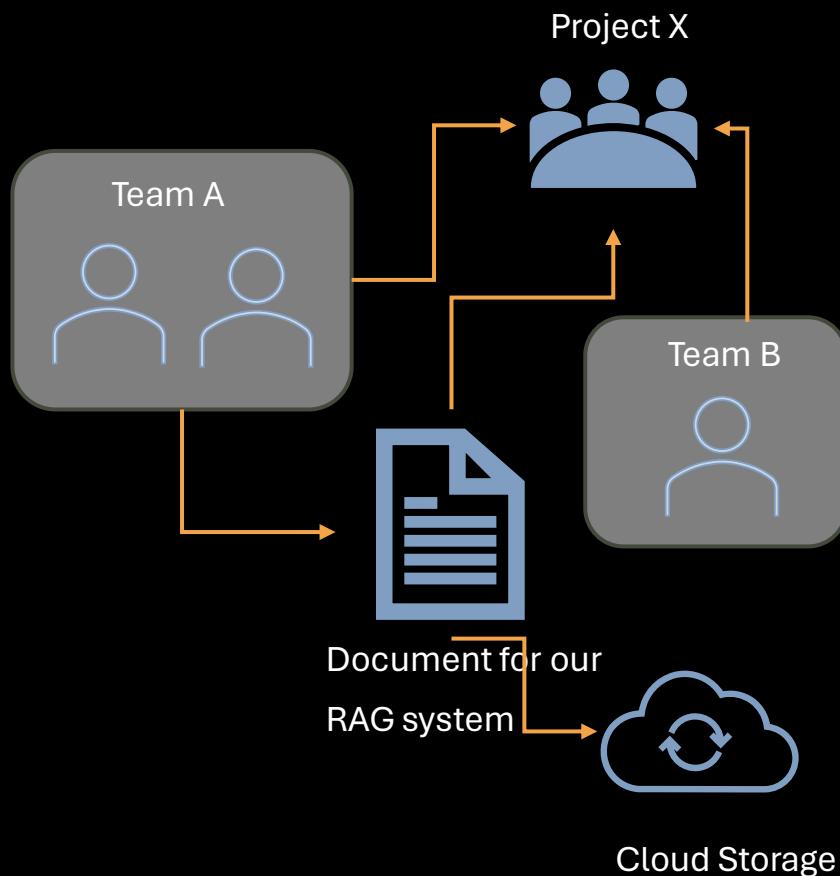
Combine Keyword search (like BM25) with vector search and combine results

Supported by many vector databases (here weaviate)

```
jeopardy = client.collections.get("JeopardyQuestion")
response = jeopardy.query.hybrid(
    ... query="food",
    ... alpha=0.5,
    ... limit=3
)

for o in response.objects:
    ... print(o.properties)
    ... print(o.metadata.score, o.metadata.explain_score)
```

Hybrid: Use Meta-data and different views on the data



Don't forget about classical retrieval techniques

- Sometimes we reinvent things, because we forget that they already exist
- Examples:
 - FFT (Gauss 1805 (number theory), Cooley and Tukey 1965)
 - Backpropagation
- Information retrieval is a well-established discipline within computer science,