Building, Evaluating, and Optimizing your RAG App for Production

Simon Suo, co-founder/CTO of LlamaIndex

Setup

1. Clone Github Repo

```
git clone
https://github.com/Disiok/ai-engineer-workshop.git
```

2. Setup python environment

```
python3 -m venv rag
source rag/bin/activate
pip install -r requirements.txt
```

3. Setup postgres

```
docker-compose up -d
```

Launch Notebooks

```
jupyter lab
```

📅 Today's agenda.

- Prototype
- Production Challenges: Diagnosis & Mitigation
- Evaluate
- Experiment & Optimize

Prototype

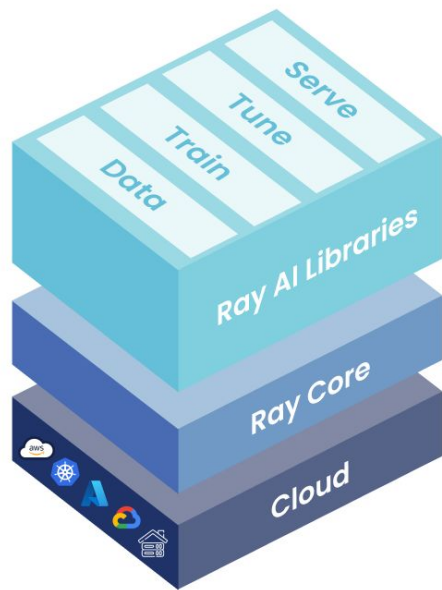## RAG System with LlamaIndex and Ray

LlamaIndex

Data Framework for LLM Applications

- Ingestion, indexing, and querying

- Orchestrates over LLMs, embedding models, vector DBs, graph DBs

- RAG, chat, data agents

Ray

high-level libraries that enable simple scaling of AI workloads

a low-level distributed computing framework with a concise core and Python-first API

Scale the entire ML pipeline

- Scalable embedding generation & indexing with **Ray Data**
- Distributed training & fine-tuning with **Ray Train**
- Scalable application deployments with **Ray Serve**

What are we building?

Ray Assistant

Ray Docs AI - Ask a question   ✕

| How can I parallelize a function with Ray? | Ask AI |

To parallelize a function with Ray, you can use the `ray.remote` decorator to mark the function as remote, and then call it using the `ray.get` method. For example:

```python
import ray

ray.init(num_cpus=4)  # Specify this system has 4 CPUs

@ray.remote
def my_function(x):
    return x * 2

results = ray.get([my_function.remote(x) for x in [1, 2, 3, 4]])
print(results)  # [2, 4, 6, 8]
```
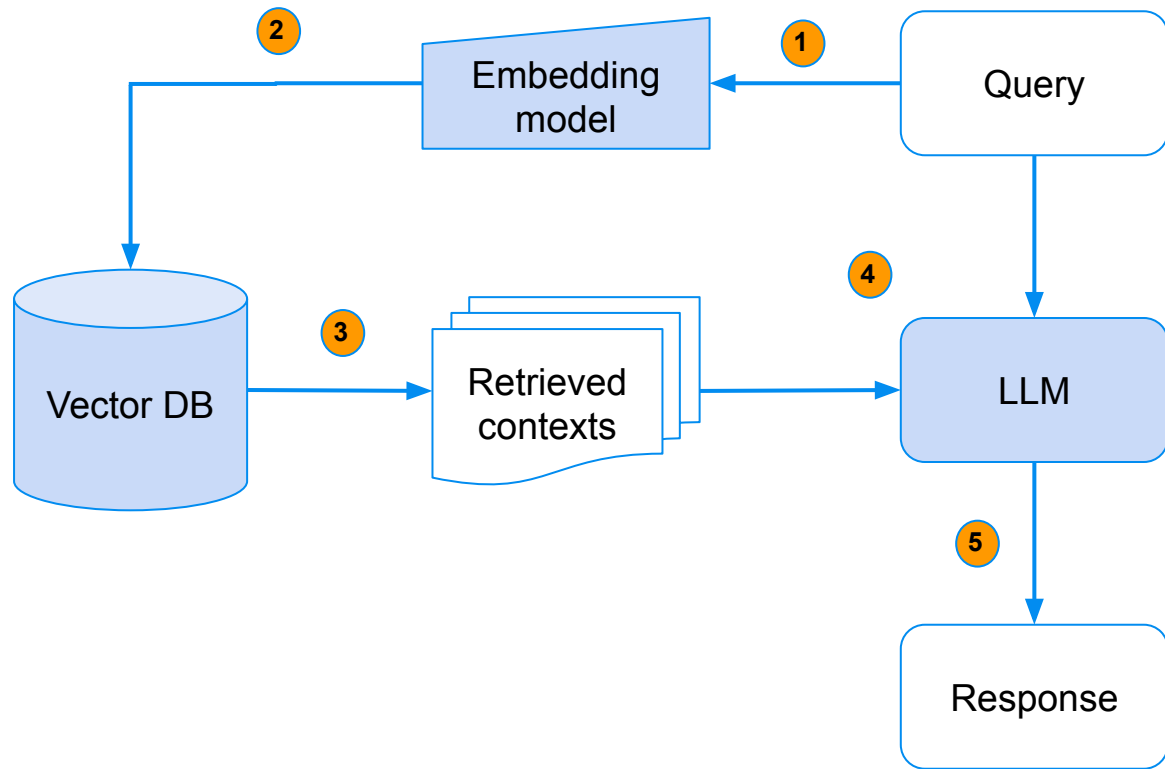
This will run the `my_function` function on 4 different CPUs, and return the results in a list.
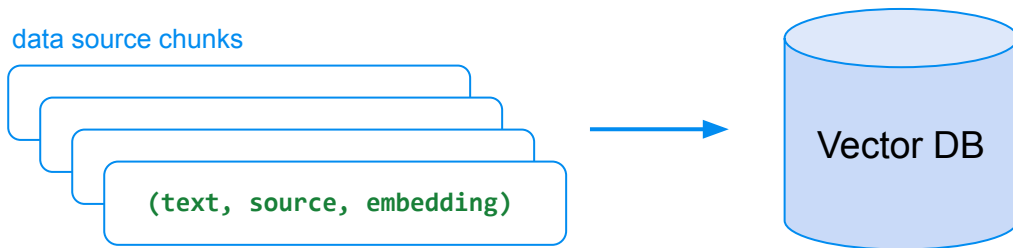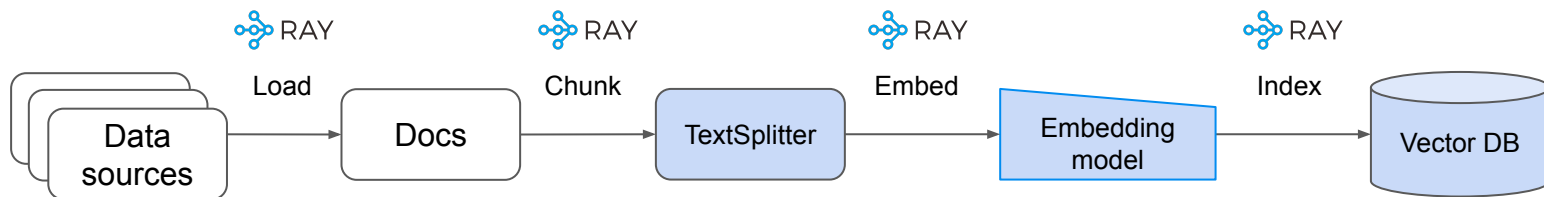
Base LLMs

```
Query  →  LLM  →  Response
```

RAG

Creating our Vector DB

Lab

Let's dive right in!

Production Challenges

Diagnosis & mitigation

From Prototype to Production

Prototype

Productionize

Few lines of code

latency

cost

hallucination

poor retrieval

output parsing errors

harmful answers

Need for evaluation, diagnosis, and optimization!

Challenges - non quality related

Symptoms

- Latency, rate-limits
- Cost
- Service Availability

Diagnosis & Mitigation

- Logging & monitoring
- Isolate issue to retrieval vs. generation components
- Evaluate different LLM service providers
- Use smaller, task-specific models
- Host your own models

Challenges - quality related

Symptoms

- Unknown performance
- Hallucinations
- Incomplete answers
- Poor retrievals

Diagnosis & Mitigation

- **Evaluation**
  - Collect labels
  - Collect user feedback
- **Optimization**
  - Tune configs
  - Customization & fine-tune models

Evaluate

Understanding system performance

Expectation vs. Reality of Evaluation

- "Academic benchmark is all you need"
- Build, evaluate, done
- A single metric number that perfectly tell you the performance

- I don't have user data yet?
- Should I label some data?
- "Vibe-check" only
- I can't run big evaluation on every CI run
- When should I evaluate?

The development process

Idea Validation

Build prototype with toy data

Solution Validation

Build prototype with full corpus

Deployment

Deploy system & collect user queries & feedback

Optimization

Collect labels, improve system

Evaluation in the development process

Idea Validation

Build prototype with toy data

At this stage

- Quick iteration cycle is critical
- "Vibe check" on ad-hoc queries
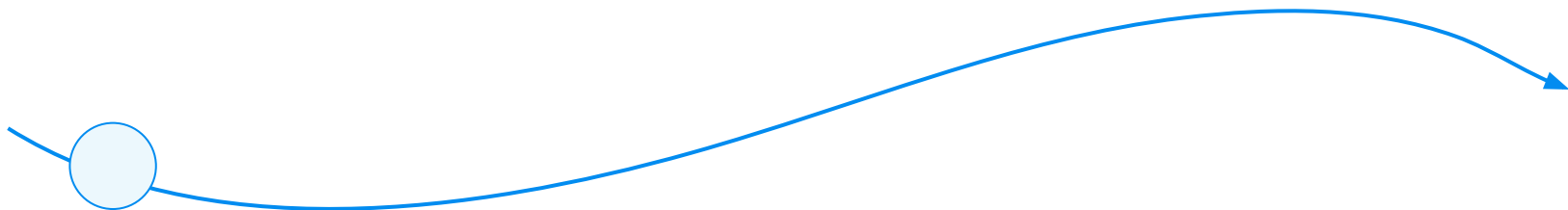- Experiment with different modules
- Ad-hoc configuration changes

The "Vibe Check"

Ad-hoc spot check with random questions.

- Quick way to sanity check and iterate
- Good for initial prototyping and early development
- Could be surprising good indication of performance

Not systematic, but useful!

Before we place a model in front of actual users, we like to test it ourselves and get a sense of the model's "vibes". The HumanEval test results we calculated earlier are useful, but there's nothing like working with a model to get a feel for it, including its latency, consistency of suggestions, and general helpfulness. Placing the model in front of Replit staff is as easy as flipping a switch. Once we're comfortable with it, we flip another switch and roll it out to the rest of our users.

https://blog.replit.com/llm-training

Finally, sometimes the best eval is human eval aka vibe check. (Not to be confused with the poorly named code evaluation benchmark HumanEval.) As mentioned in the Latent Space podcast with MosaicML (34th minute):

> The vibe-based eval cannot be underrated. ... One of our evals was just having a bunch of prompts and watching the answers as the models trained and see if they change. Honestly, I don't really believe that any of these eval metrics capture what we care about. One of our prompts was "suggest games for a 3-year-old and a 7-year-old to play" and that was a lot more valuable to see how the answer changed during the course of training. — Jonathan Frankle

https://eugeneyan.com/writing/llm-patterns/

The development process

Idea Validation

Build prototype with toy data

Solution Validation

Build prototype with full corpus

At this stage

- "Vibe check" on curated/representative set of test queries
- Towards more **systematic evaluation** to gain confidence towards initial deployment
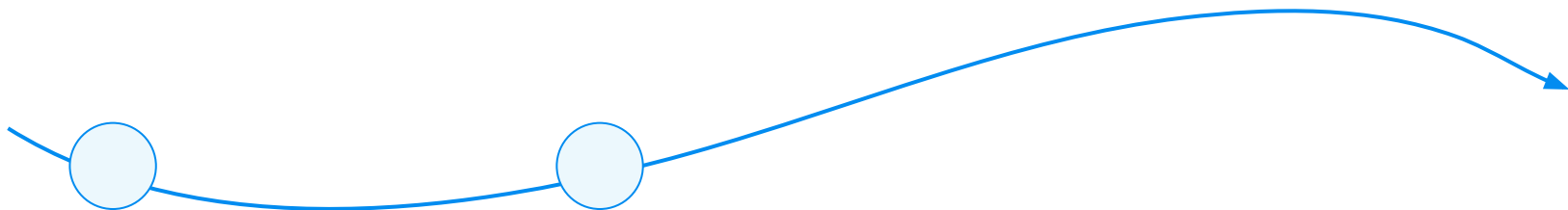
Challenge of Systemic Evaluation

- Metrics
  - Flexibility of natural language, no one right answer
  - Human evaluation is not scalable
- Data availability
  - Labeled data is slow & costly to collect
- Actionable insight
  - Not just "it's bad", but also "how to improve"
  - End-to-end vs. Component-wise

LLM-as-a-Judge

- Strong LLMs (GPT-4, Claude 2) are good evaluators
  - High agreement with human
  - scalability
  - Interpretability

https://www.databricks.com/blog/LLM-auto-eval-best-practices-RAG
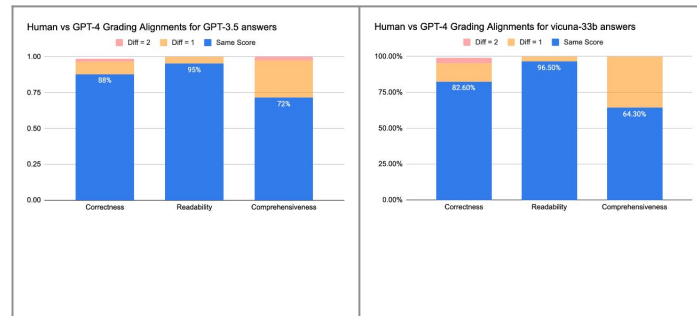
- Approaches
  - Pairwise comparison
  - Single answer grading
  - Reference-guided grading (i.e. "golden" answer)

https://arxiv.org/pdf/2306.05685.pdf
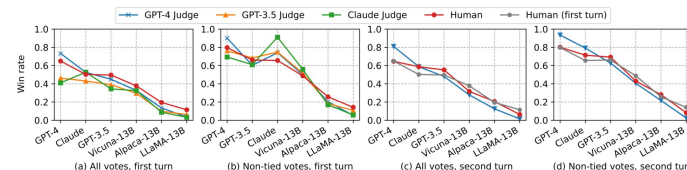
Systematic Evaluation - Overview

End-to-end evaluation

Helps understand how well the **full RAG application** works:

- given user question, how "good" is the answer

Component-wise evaluation

Helps attribute quality issue to specific components:

- Retrieval: are we retrieving the relevant context
- Generation: given context, are we generating an accurate and coherent answer

⬇

Analogous to **integration** tests

⬇

Analogous to **unit** tests

Data for Systematic Evaluation

User Query

- representative set of real user queries

User Feedback

- Feedback from past interaction
- up/down vote, rating
- On retrieval and/or generation

"Golden" Context

- Set of relevant documents from our corpus to best answer a given query

"Golden" Answer

- Best answer given "golden" context
- Can be optional

Data Challenges

User Query	User Feedback	"Golden" Context	"Golden" Answer

- Need to deploy system & collect

- Need to deploy system & collect

- Need labelers

- Need labelers

Relatively easy

Require good UX for good data

Relatively cheap/easy

More costly/tedious

"Cold Start" Problem

Dilemma

We have the chicken and egg problem of:

1. We want to evaluate to gain confidence of our RAG system before deployment
2. Without deployment, we can't collect real user queries and label an evaluation dataset

Strategy

- Use purely LLM-based label-free evaluation
- Leverage LLM to generate synthetic label dataset, including
  - Query
  - "Golden" context
  - "Golden" response

Generating Synthetic Evaluation Data

- Sample document chunks, and leverage LLM to generate Q&A pairs that can be best answer by the given context.
- Not always representative of user queries, but useful for development iterations & diagnostics

Knowledge corpus

Sample

doc_0

Document chunk

LLM

{
User query: …
"Golden" context: doc_0
"Golden" answer: …
}

Synthetic "datapoint" for eval
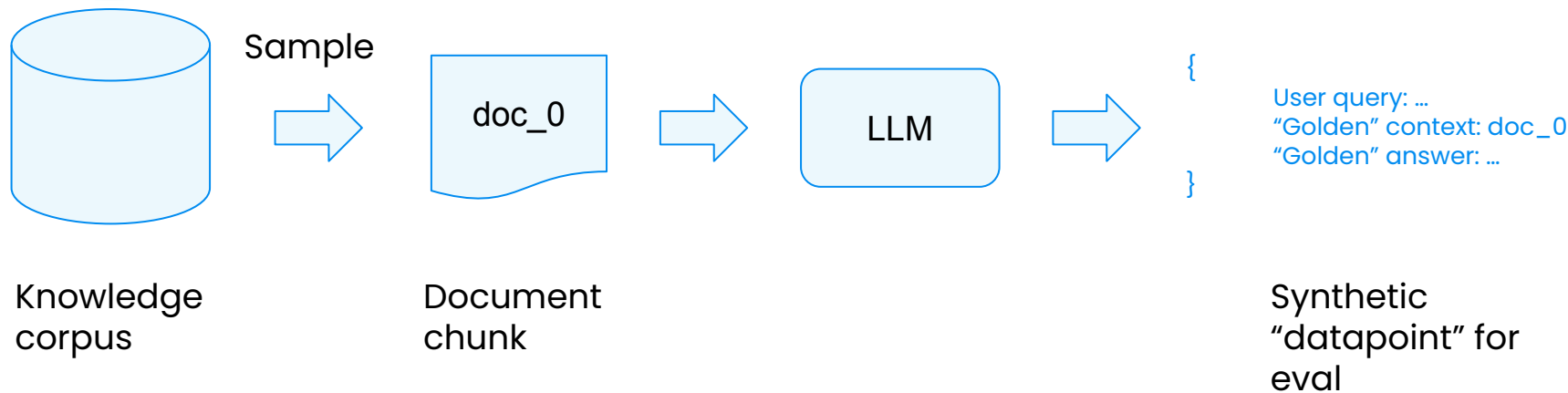
The development process

At this stage

- Have user queries, maybe labels & feedback
- Need repeatable, consistent, automated evaluation
- Need actionable insight or automated tuning

Deployment

Deploy system & collect user queries & feedback

Optimization

Collect labels, improve system

End-to-end evaluation

LLM-as-a-Judge:
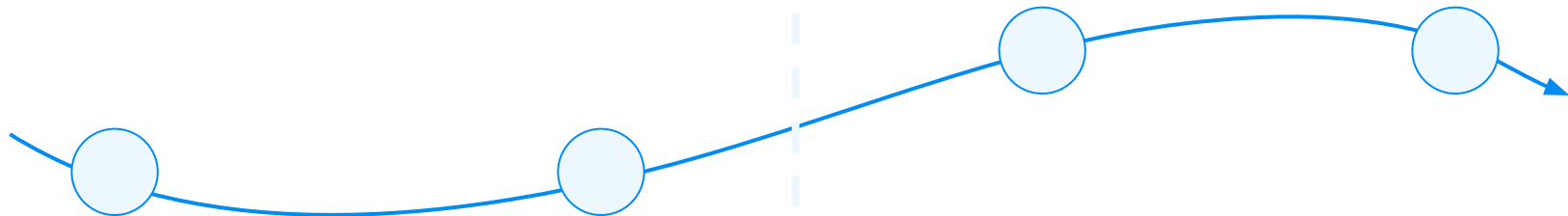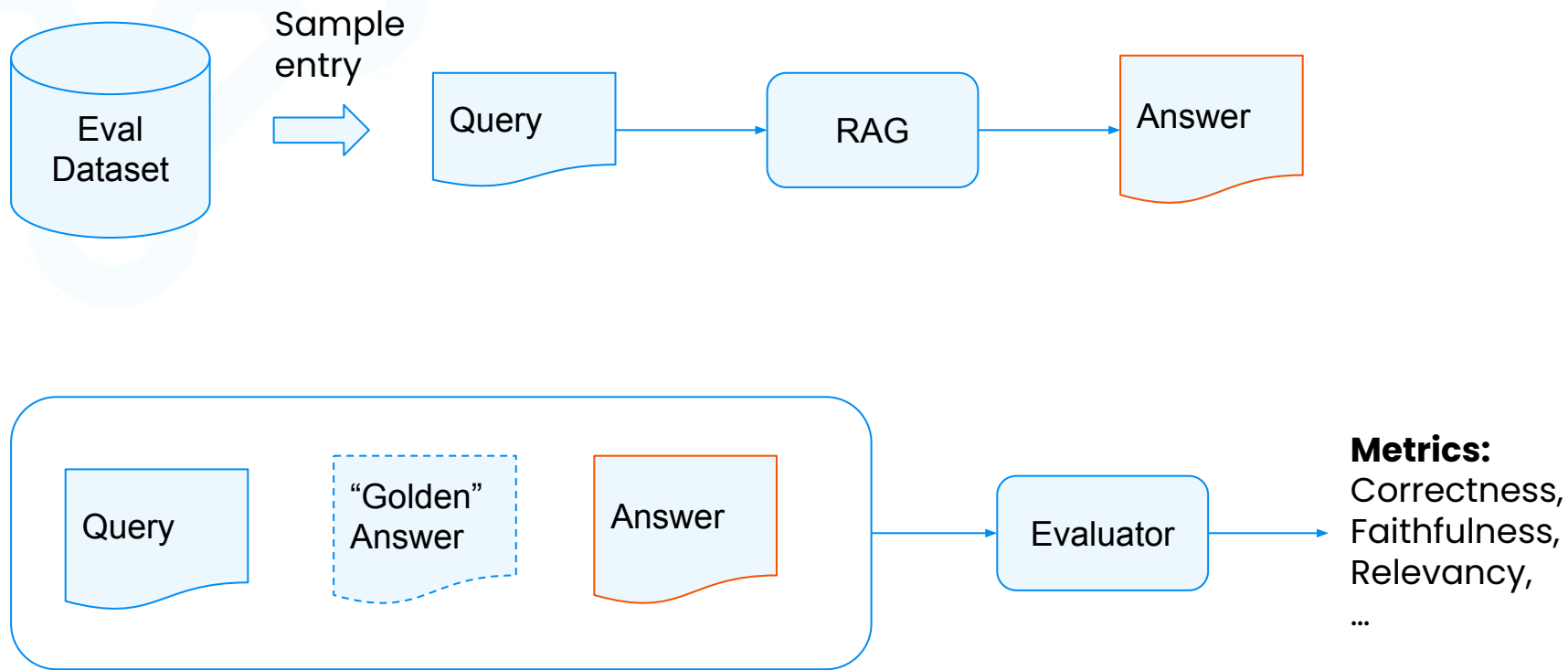# The devil is in the details

- **What model?**
  - Only "strong" LLMs right now
  - GPT-4, Claude 2
- **What grading scale?**
  - Low-precision, e.g. binary, 1-5
  - Easier rubrics, more interpretable
- **Do we need few-shot examples?**
  - Helps to give guidance/example for each score level
- **Holistic judgement vs. individual aspects**
  - Be as concrete as possible
- **Chain of thought reasoning**

```
You are an expert evaluation system for a question answering
chatbot.

You are given the following information:
- a user query,
- a reference answer, and
- a generated answer.

Your job is to judge the relevance and correctness of the
generated answer.
Output a single score that represents a holistic evaluation.
You must return your response in a line with only the score.
Do not return answers in any other format.
On a separate line provide your reasoning for the score as well.

Follow these guidelines for scoring:
- Your score has to be between 1 and 5, where 1 is the worst and 5
is the best.
- If the generated answer is not relevant to the user query, \
you should give a score of 1.
- If the generated answer is relevant but contains mistakes, \
you should give a score between 2 and 3.
- If the generated answer is relevant and fully correct, \
you should give a score between 4 and 5.

## User Query
{query}

## Reference Answer
{reference_answer}

## Generated Answer
{generated_answer}
```

LLM-as-a-Judge:
Limitations

- **Position bias**
  - First, last, etc

- **Verbosity bias**
  - Longer the better?

- **Self-enhancement bias**
  - GPT <3 GPT

```
You are an expert evaluation system for a question answering
chatbot.

You are given the following information:
- a user query,
- a reference answer, and
- a generated answer.

Your job is to judge the relevance and correctness of the
generated answer.
Output a single score that represents a holistic evaluation.
You must return your response in a line with only the score.
Do not return answers in any other format.
On a separate line provide your reasoning for the score as well.

Follow these guidelines for scoring:
- Your score has to be between 1 and 5, where 1 is the worst and 5
is the best.
- If the generated answer is not relevant to the user query, \
you should give a score of 1.
- If the generated answer is relevant but contains mistakes, \
you should give a score between 2 and 3.
- If the generated answer is relevant and fully correct, \
you should give a score between 4 and 5.

## User Query
{query}

## Reference Answer
{reference_answer}

## Generated Answer
{generated_answer}
```

Component-wise evaluation

- Evaluate **retrieval** component

Eval Dataset → Sample entry → Query → RAG - Retrieval → Retrieved docs

"Golden" context, Retrieved docs → Retrieval Evaluator → **Metrics:**
- Hit rate
- MRR
- MAP
- NDCG
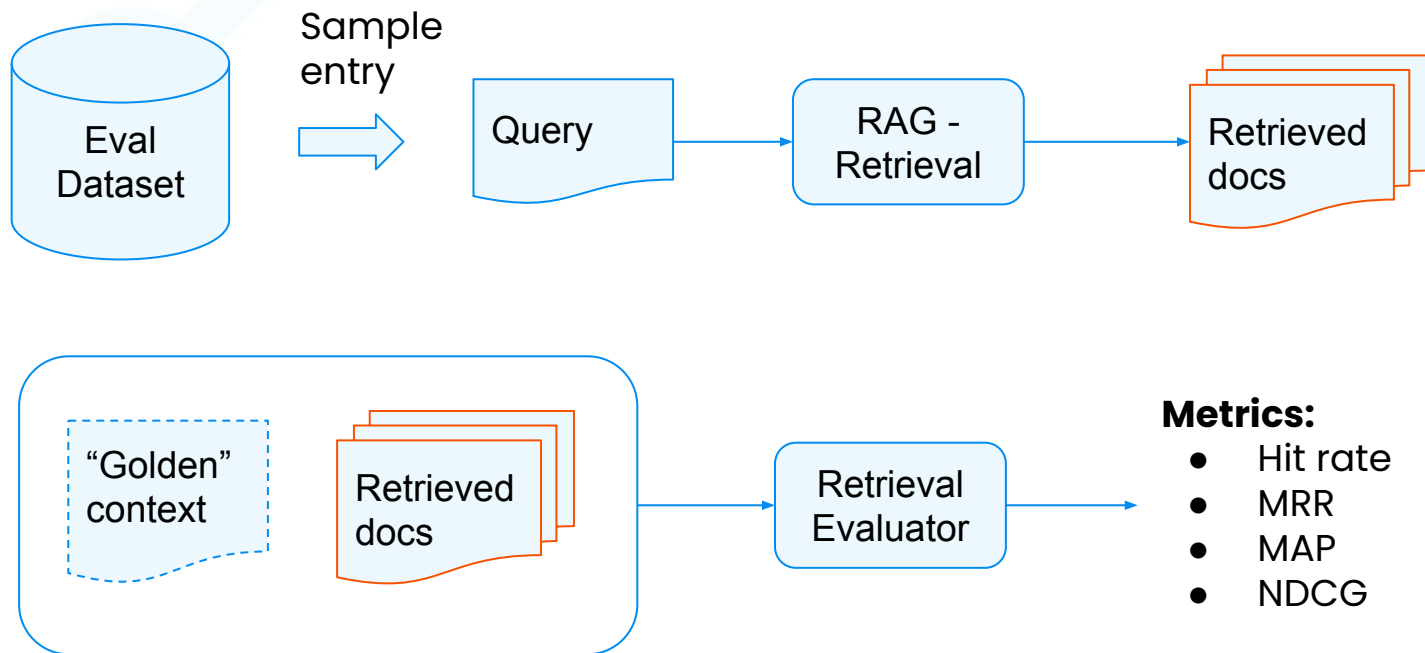
Component-wise evaluation

- Evaluate **generation** component

Metrics:
Correctness,
Faithfulness,
Relevancy,
…

Experiment config

build_or_load_index

Embedding_model_name
Chunk_size
chunk_overlap

Index

Retriever

Similarity_top_k
embedding_model_name

Query Engine

Similarity_top_k
Embedding_model_name
Llm_model_name
Temperature

run_retrieval_exp

Queries

"Golden" Contexts

Retriever

Hit rate

run_e2e_exp

Queries

"Golden" Answers

Query Engine
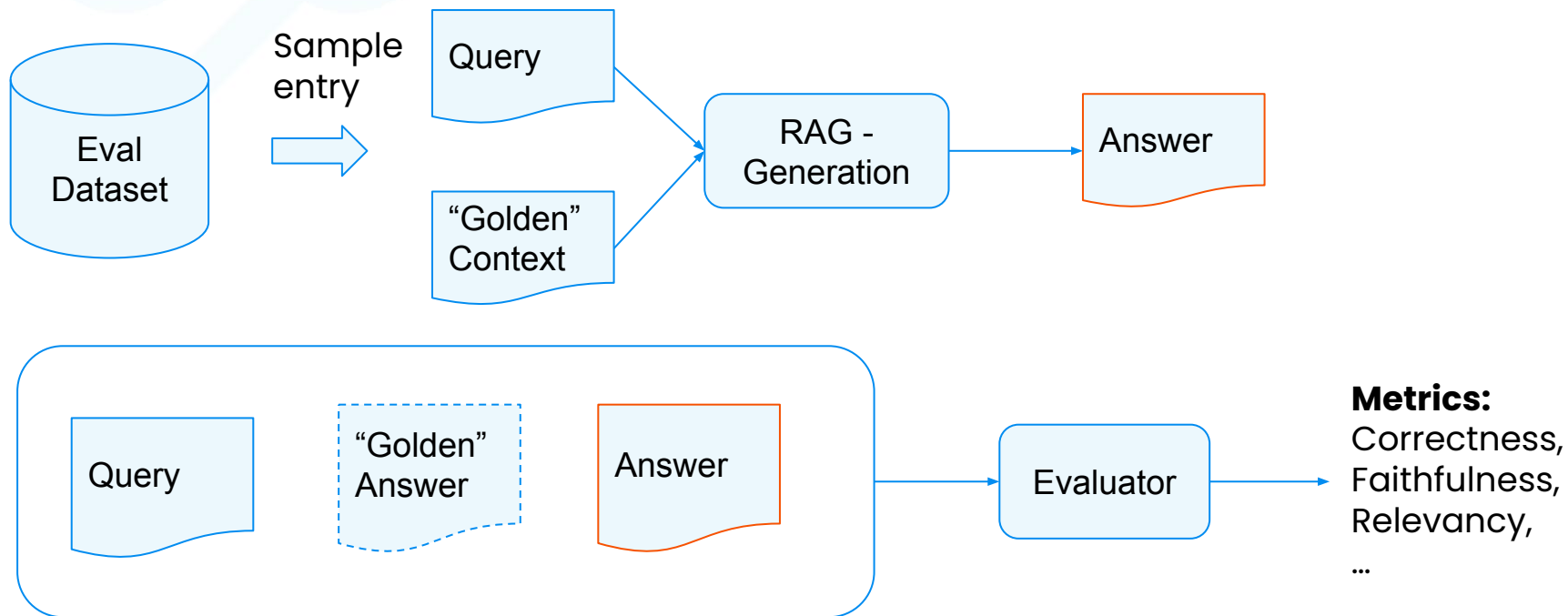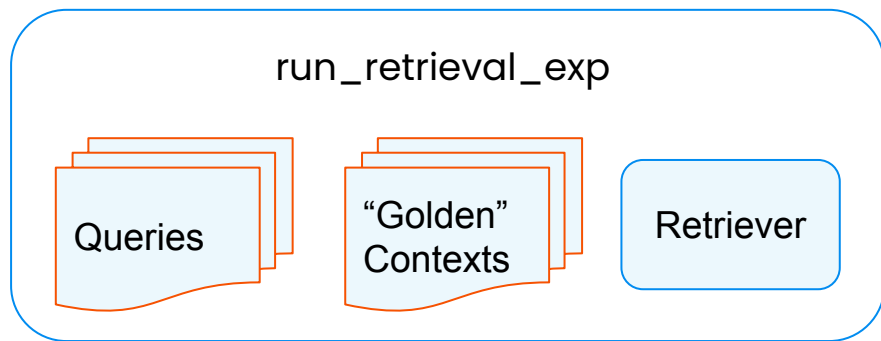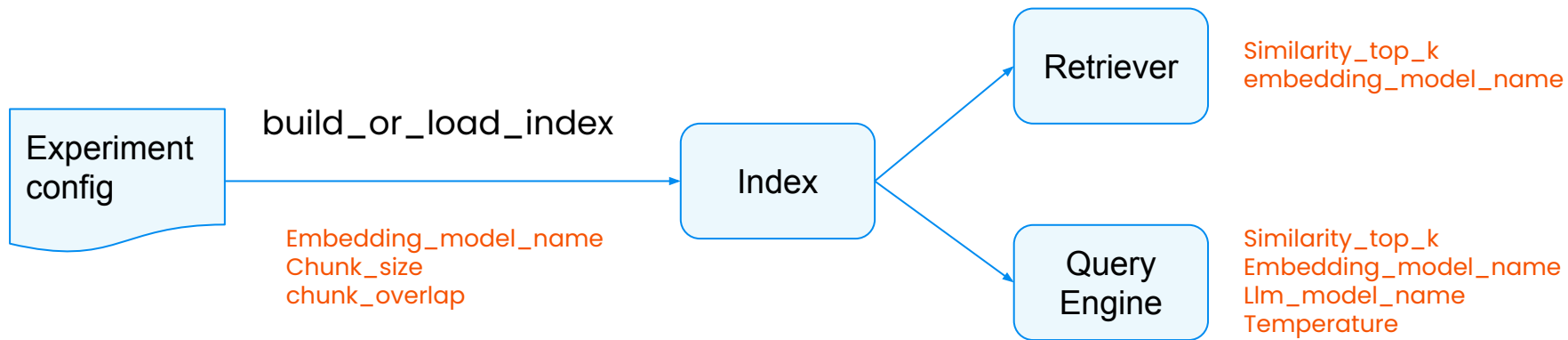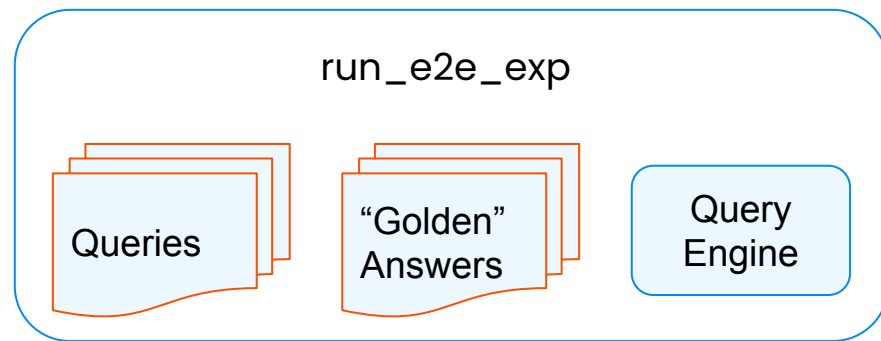
Mean correctness score

Lab

- Cold start problem: generating synthetic eval dataset from documents
- Component wise evaluation
  - Retrieval
  - Generation
- End-to-end evaluation

Experiment & Optimize

Improve system performance

Ways to optimize the RAG system

Configure standard components

- Use standard, off the shelf components (e.g. LLM, embedding model, standard semantic search)

- Select good components, tune parameters for end-performance

Build customized pipeline

- Deeply understand your data & query pattern
- define indexing & retrieval strategies optimized for your specific use-case

Model Fine-tuning

- Specialize embedding model and LLM to your domain

Configure standard components

Retrieval

- Chunk size
- Embedding model
- Number of retrieved chunks

Generation

- LLM
- (prompt)

Grid search

- Run all parameter combinations and evaluate the end performance of system

```
{
    Chunk_size: 512,
    Top_k: 5,
    Llm: gpt-4,
    …
}
```
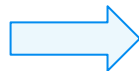
➡️ Score: 4.5

```
{
    Chunk_size: 512,
    Top_k: 10,
    Llm: gpt-4,
    …
}
```

➡️ Score: 3.5

EMBED_MODELS = [
    "thenlper/gte-base",
    "BAAI/bge-large-en",
    "text-embedding-ada-002"
]

Embedding Model

Query

LLMS = [
    "gpt-3.5-turbo",
    "gpt-4",
    "meta-llama/Llama-2-7b-chat-hf",
    "meta-llama/Llama-2-13b-chat-hf",
    "meta-llama/Llama-2-70b-chat-hf"
]

Vector Store

Retrieved docs

LLM

CHUNK_SIZES = [256, 512, 1024]
TOP_K = [1, 3, 5, 7]

Response
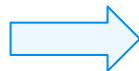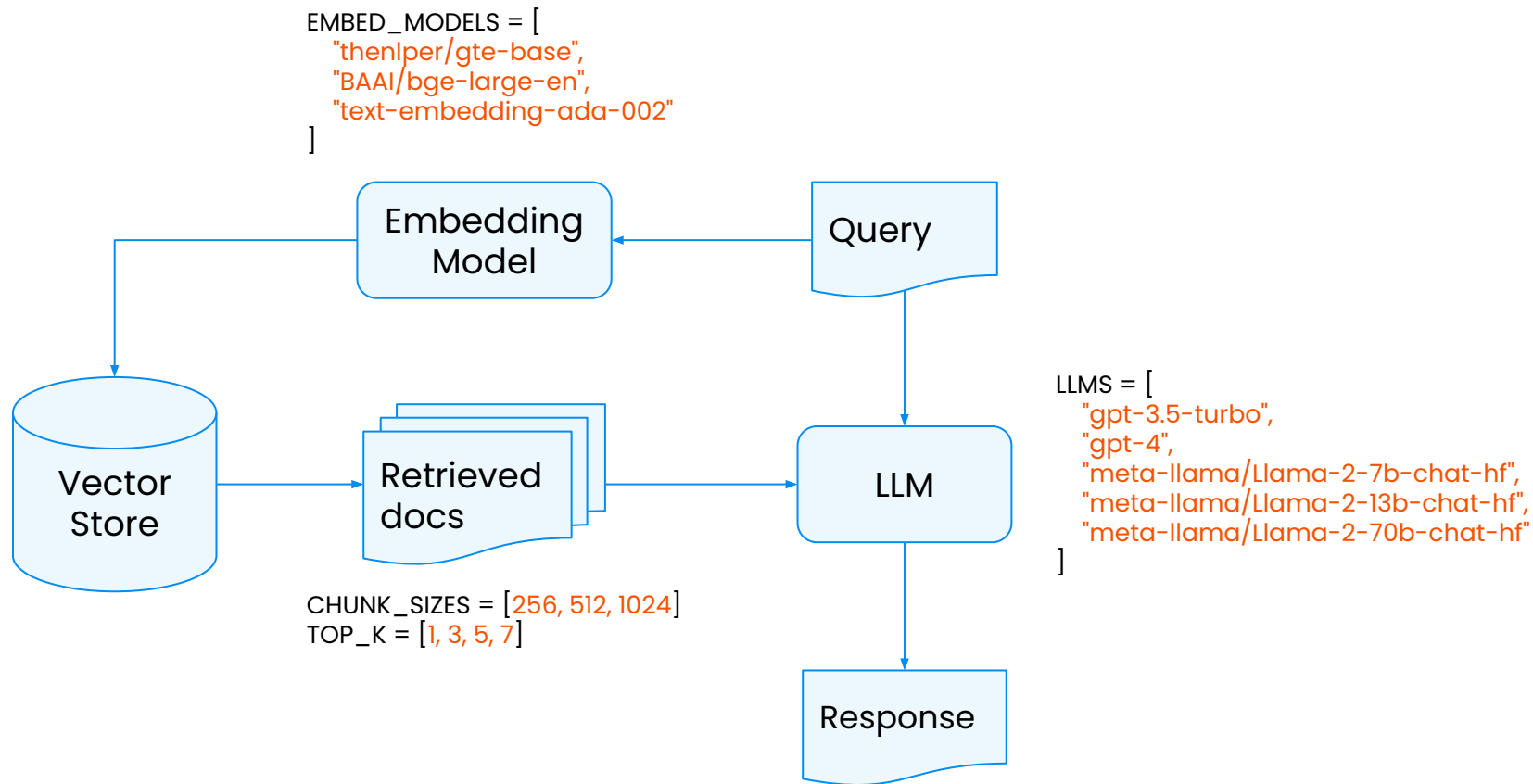
Lab

- Run experiments over standard component configurations
- Gain intuition on optimal parameter configurations

Customizing Retrieval & Generation

Strategy 1:

Use two-stage retrieval:

- First retrieve a lot of potentially relevant contexts
- Then rerank/filter to a smaller subset

Index → Retriever → Reranker →

- Less accurate
- Fast
- Operate over full corpus

- More accurate
- Slow
- Operate over candidates

Index → Retriever

Query

Top 20
- doc_0
- doc_1
- doc_2
- ⋮
- doc_19

Reranker

Query

Top 5
- doc_2
- doc_19
- ⋮

LLM

Query
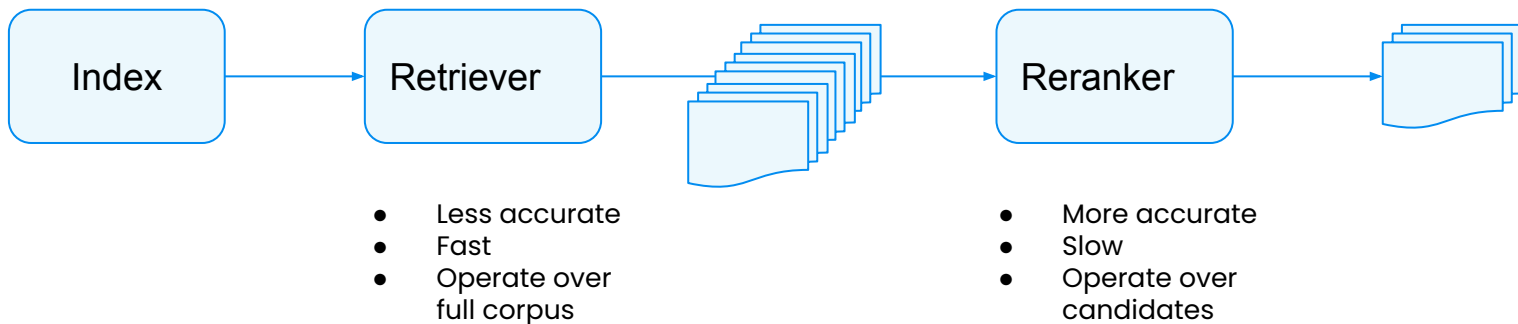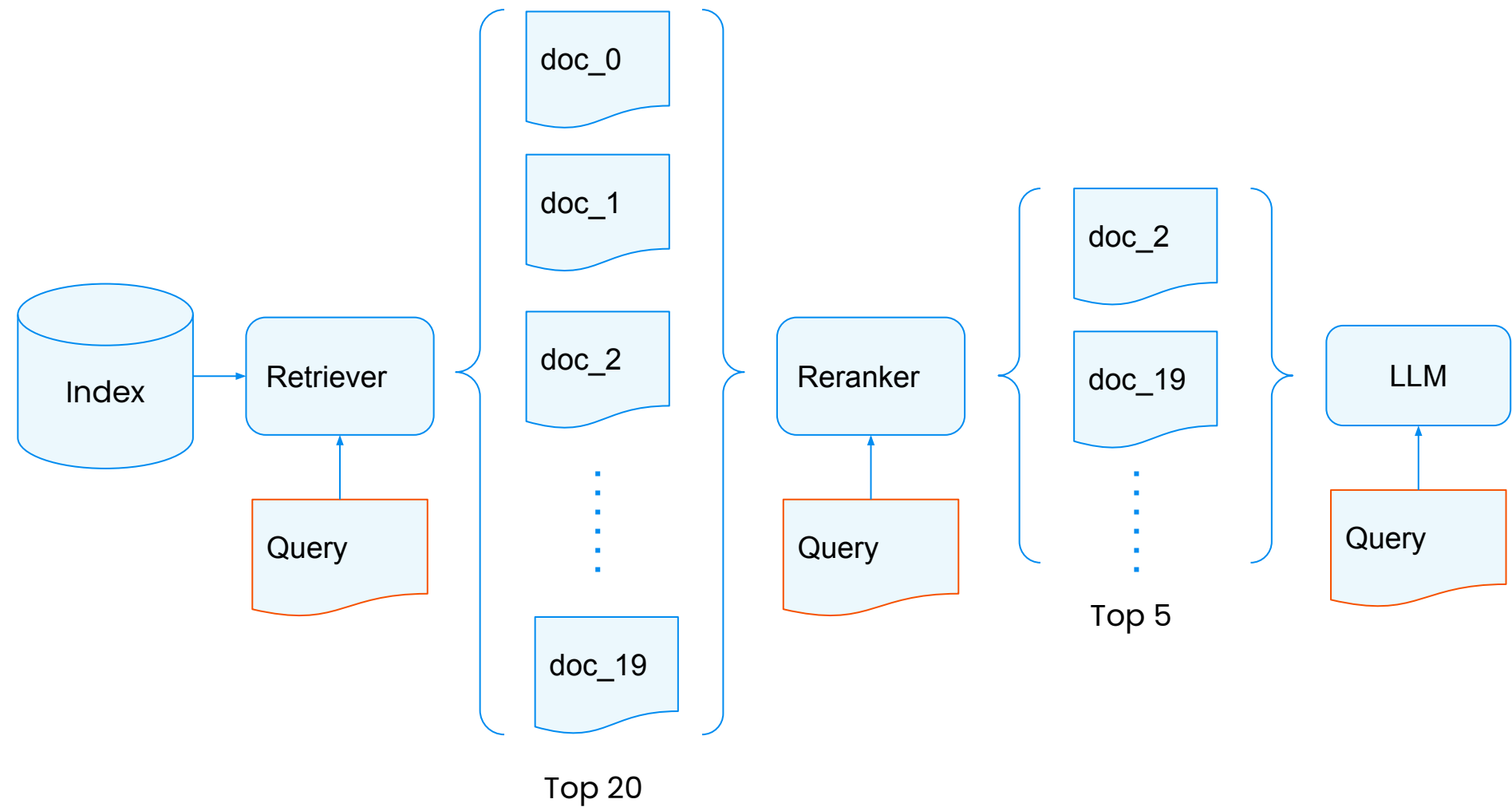
Customizing Retrieval & Generation

Strategy 2a:

Embed different representations of the same data: e.g.

- Summarize document and embed summary
- Extract distinct topics and embed topic extractions separately

This can improve retrieval over specific questions
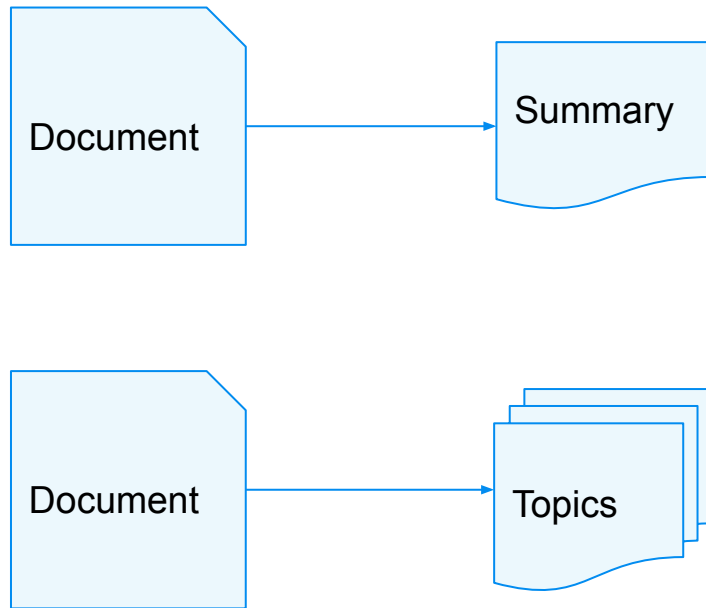
Customizing Retrieval & Generation

Strategy 2b:

Use different data representation for retrieval & for generation

Embed Summary → Link to Additional Documents

Question: What are the concerns surrounding the AMOC?

Embedding Lookup

Doc Summary

Doc Summary

Doc Summary

Retrieve Document Chunks for Synthesis

Document Chunks

Embed Sentence → Link to Expanded Window

Question: What are the concerns surrounding the AMOC?

Embedding Lookup

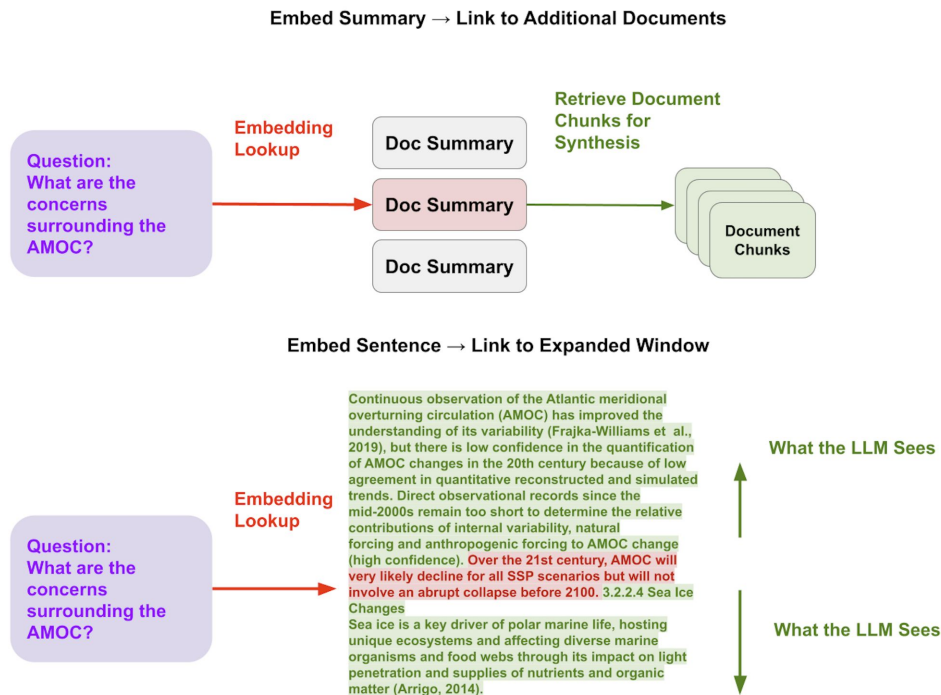Continuous observation of the Atlantic meridional overturning circulation (AMOC) has improved the understanding of its variability (Frajka-Williams et al., 2019), but there is low confidence in the quantification of AMOC changes in the 20th century because of low agreement in quantitative reconstructed and simulated trends. Direct observational records since the mid-2000s remain too short to determine the relative contributions of internal variability, natural forcing and anthropogenic forcing to AMOC change (high confidence). Over the 21st century, AMOC will very likely decline for all SSP scenarios but will not involve an abrupt collapse before 2100. 3.2.2.4 Sea Ice Changes
Sea ice is a key driver of polar marine life, hosting unique ecosystems and affecting diverse marine organisms and food webs through its impact on light penetration and supplies of nutrients and organic matter (Arrigo, 2014).
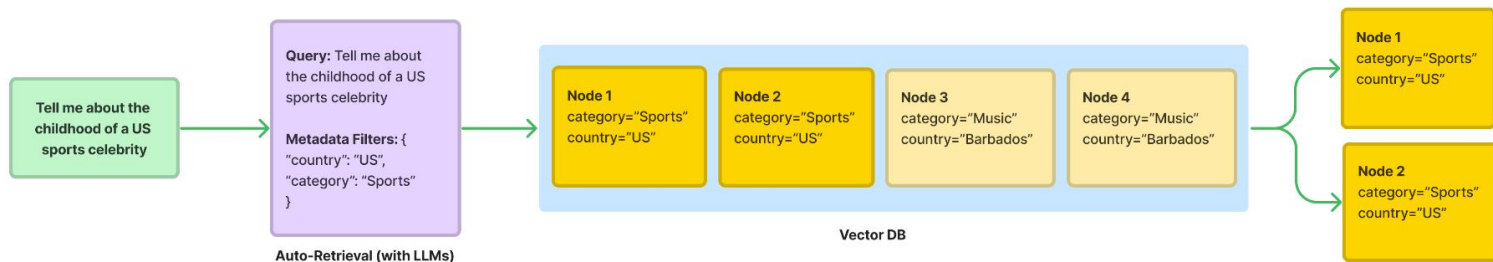
What the LLM Sees

What the LLM Sees

Customizing Retrieval & Generation

Strategy 3:

Leverage LLM to infer structured query for retrieval (e.g. metadata filters, top-k, score threshold)

Metadata Filters + Auto-Retrieval

Tell me about the childhood of a US sports celebrity

Query: Tell me about the childhood of a US sports celebrity

Metadata Filters: {
"country": "US",
"category": "Sports"
}

Auto-Retrieval (with LLMs)

Node 1
category="Sports"
country="US"

Node 2
category="Sports"
country="US"

Node 3
category="Music"
country="Barbados"

Node 4
category="Music"
country="Barbados"

Vector DB

Node 1
category="Sports"
country="US"
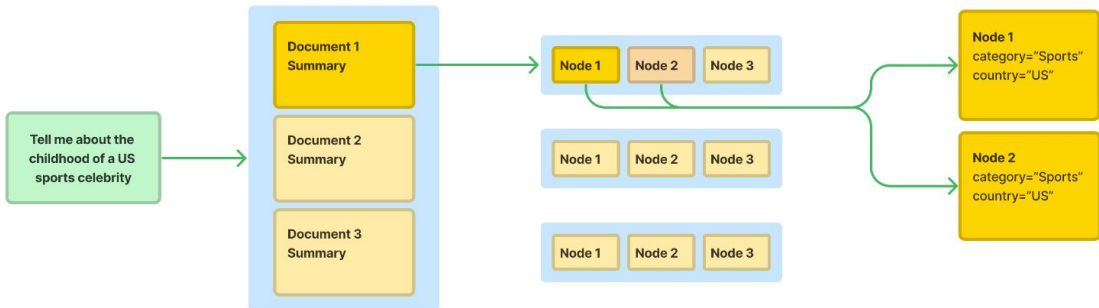
Node 2
category="Sports"
country="US"

Customizing Retrieval & Generation

Strategy 4:

Recursive retrieval over hierarchical index

Document Hierarchies (Summaries + Raw Chunks) + Recursive Retrieval
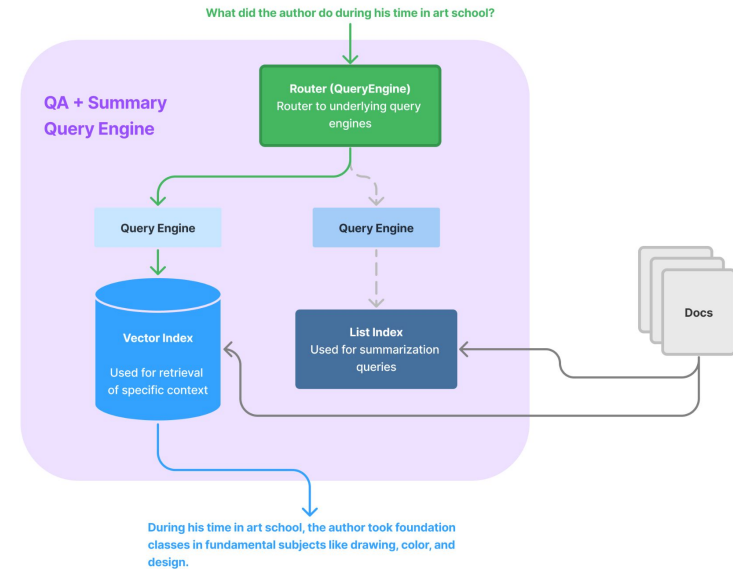
Customizing Retrieval & Generation

Strategy 5:

Routing to different retrieval methods depending on the query

May need to rewrite query as well, depending on the interface of the retriever.

Model Fine-tuning

LLM Fine-tuning

- When want to adjust the "style" of generation: e.g. professional legal assistant
- When want to enforce output structure: e.g. JSON

Embedding Model Fine-tuning

- Improve retrieval performance, especially documents that are
  - Domain specific terminology
  - Malformed (e.g. extraneous spacing due to parsing, etc)
- Two approaches
  - Fine-tune the full embedding model
  - Fine-tune an adapter layer on top of a frozen embedding model

Not great for injecting new knowledge and fighting hallucination

Great to improving retrieval (and thus end-to-end RAG performance)

Fine-tuning embeddings for RAG with synthetic data
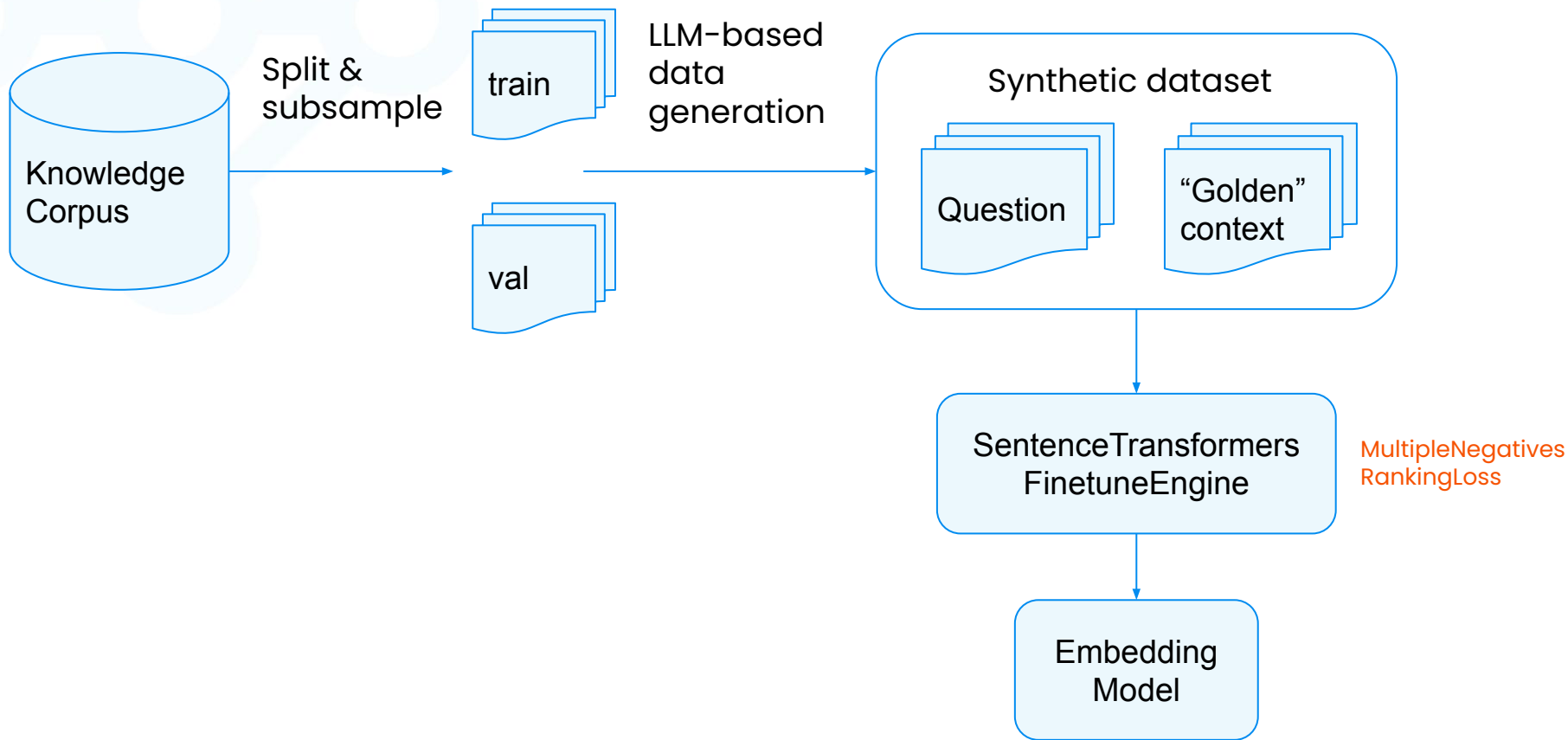
Lab

- Second-stage re-ranking
- Sentence window strategy
- Fine-tuning embeddings for RAG with synthetic data

Get In Touch

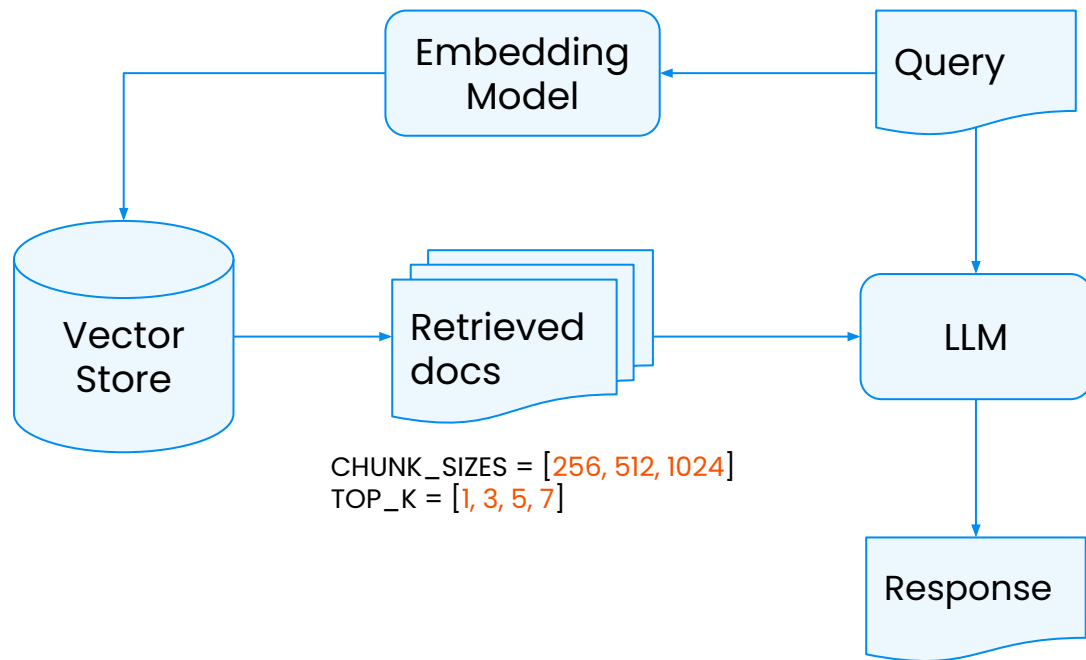Sign up for office hours

Sign up for our newsletter:
https://www.llamaindex.ai/

EMBED_MODELS = [
    "thenlper/gte-base",
    "BAAI/bge-large-en",
    "text-embedding-ada-002"
]

Embedding Model

Query

Vector Store

Retrieved docs

LLM

LLMS = [
    "gpt-3.5-turbo",
    "gpt-4",
    "meta-llama/Llama-2-7b-chat-hf",
    "meta-llama/Llama-2-13b-chat-hf",
    "meta-llama/Llama-2-70b-chat-hf"
]

CHUNK_SIZES = [256, 512, 1024]
TOP_K = [1, 3, 5, 7]

Response

Experiment config

build_or_load_index

Embedding_model_name
Chunk_size
chunk_overlap

Index

Retriever

Similarity_top_k
embedding_model_name

Query Engine

Similarity_top_k
Embedding_model_name
Llm_model_name
Temperature

run_retrieval_exp

Queries

"Golden" Contexts

Retriever

Hit rate

run_e2e_exp

Queries

"Golden" Answers

Query Engine

Mean correctness score

Index → Retriever ← Query

Top 20: doc_0, doc_1, doc_2, ⋮, doc_19

Reranker ← Query

Top 5: doc_2, doc_19, ⋮

LLM ← Query

Knowledge Corpus → Split & subsample → train, val → LLM-based data generation → Synthetic dataset (Question, "Golden" context) → SentenceTransformers FinetuneEngine (MultipleNegatives RankingLoss) → Embedding Model