



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

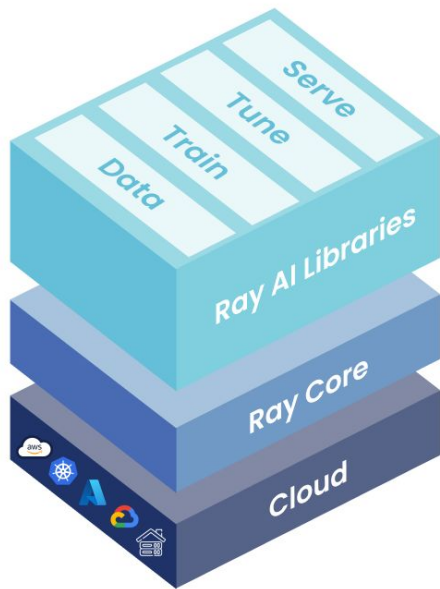
LlamaIndex



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:

```
import ray

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

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

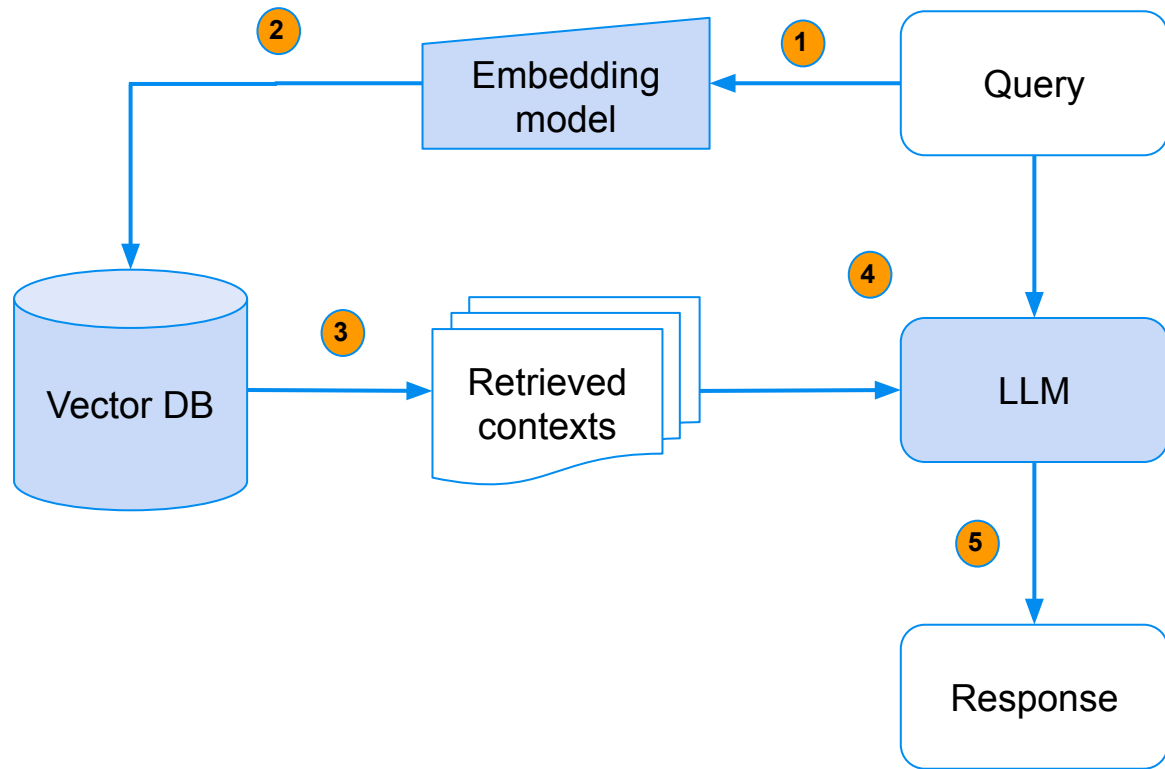
results = ray.get([my_function.remote(x) for x in [1, 2, 3, 4]])
print(results) # [2, 4, 6, 8]
```

This will run the `my_function` function on 4 different CPUs, and return the results in a list.

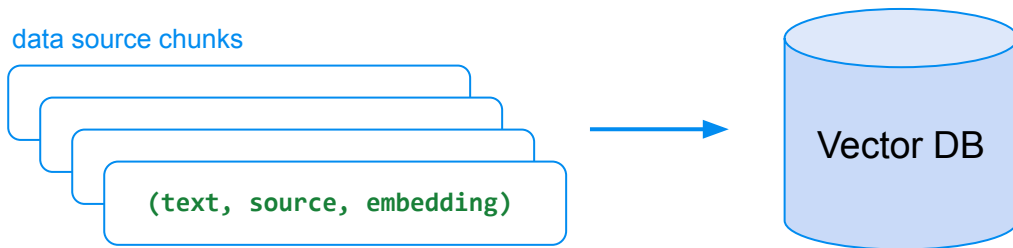
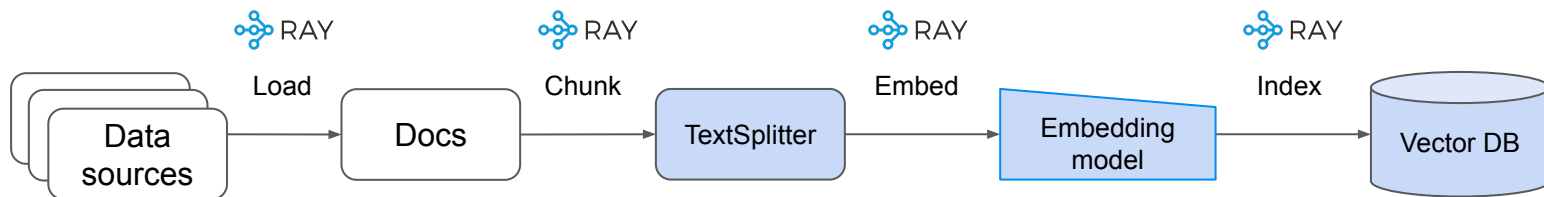
Base LLMs



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

poor retrieval

cost

output parsing
errors

hallucination

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?

Overall Bitext Mining Classification Clustering Pair Classification Reranking Retrieval STS Summarization

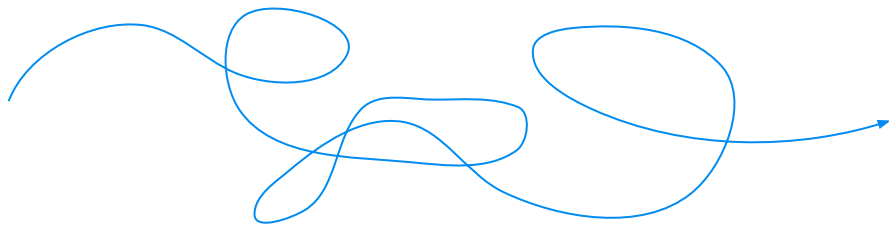
English Chinese Polish

Overall MTEB English leaderboard

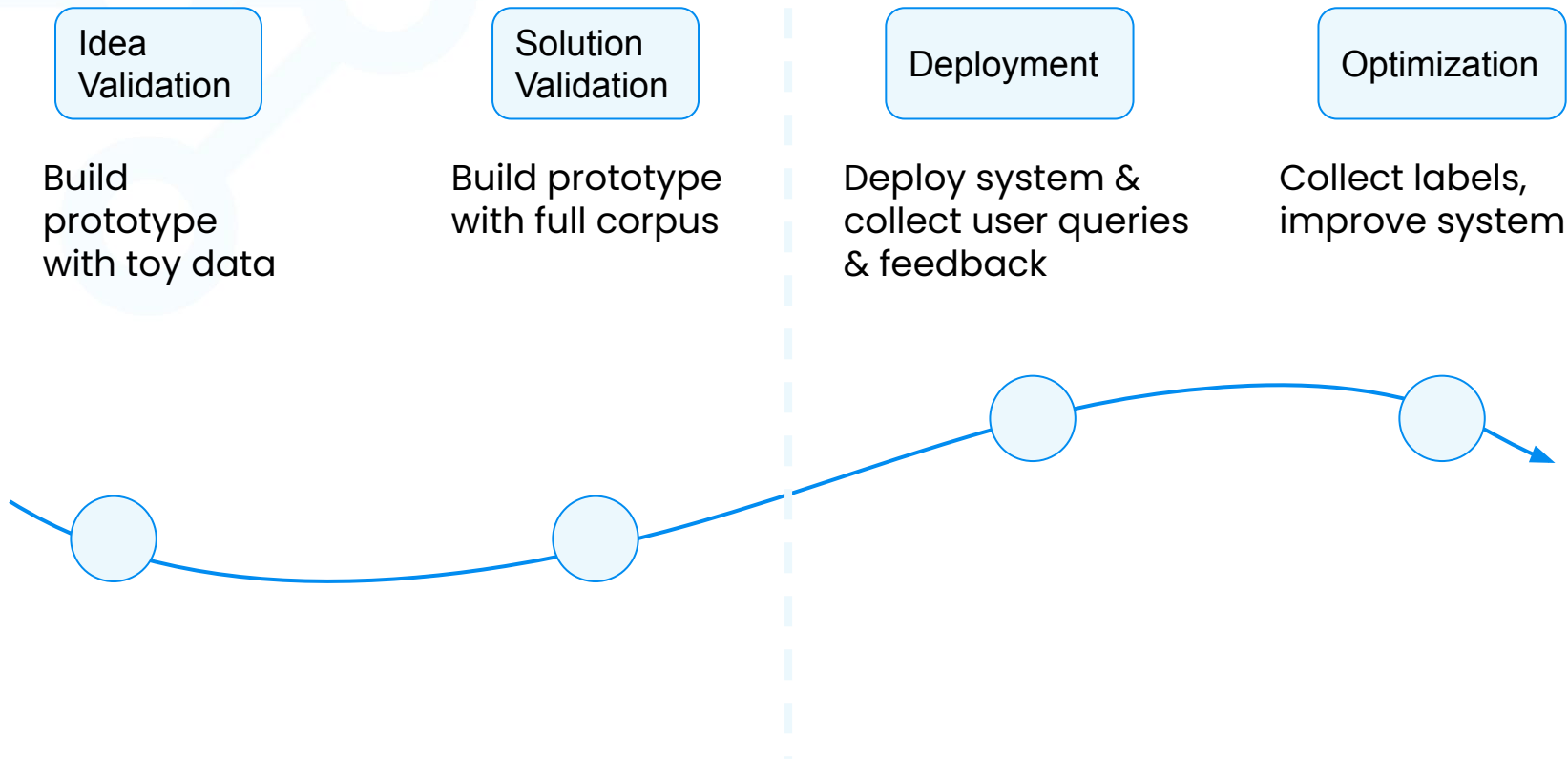
Metric: Various, refer to task tabs

Languages: English

Rank	Model	Model Size (Gb)	Embedding Dimensions	Sequence Length	Average (56 datasets)	Classification Average (12 datasets)	Clustering Average (11 datasets)	Pair Classification Average (3 datasets)	Reranking Average (4 datasets)	Retrieval Average (15 datasets)	STS Average (18 datasets)
1	bee-large-en	1.34	1024	512	63.98	76.21	46.98	85.8	59.48	53.9	81.56
2	bee-base-en	0.44	768	512	63.36	75.27	46.32	85.86	58.7	53	81.84
3	gte-large	0.67	1024	512	63.13	73.33	46.84	85	59.13	52.22	83.35
4	gte-base	0.22	768	512	62.39	73.01	46.2	84.57	58.61	51.14	82.3
5	g3-large-v2	1.34	1024	512	62.25	75.24	44.49	86.03	56.61	50.56	82.05
6	bee-small-en	0.13	384	512	62.11	74.37	44.31	83.78	57.97	51.82	80.72
7



The development process



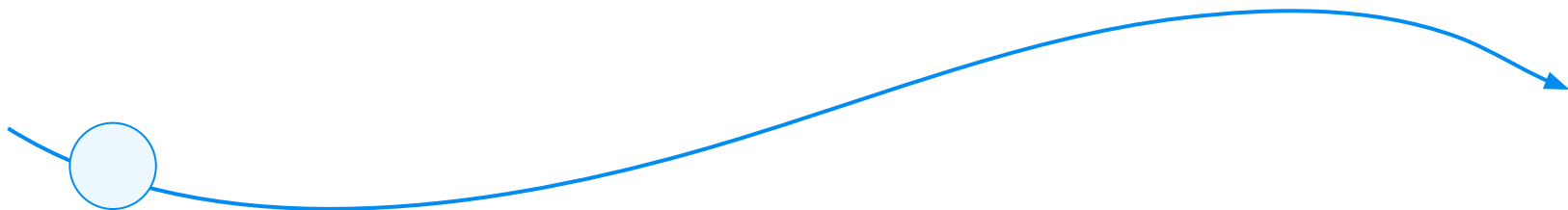
Evaluation in the development process

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

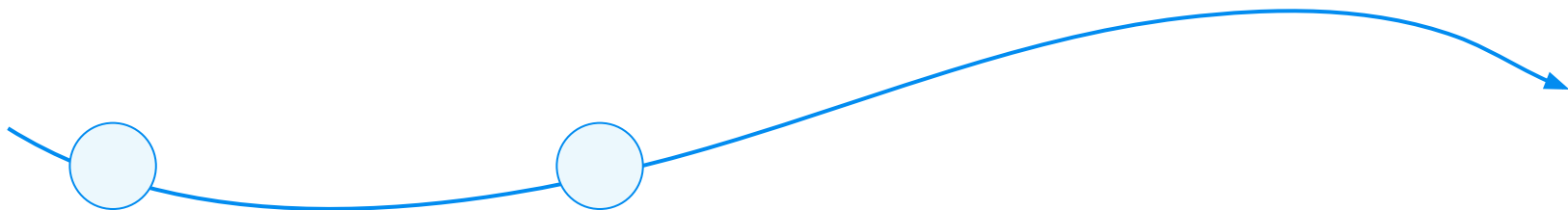
Build
prototype
with toy data

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



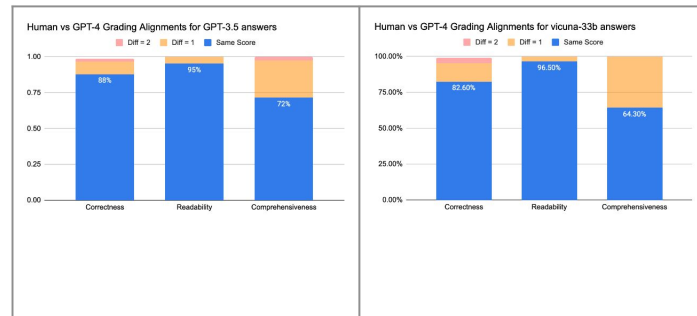
Challenge of Systemic Evaluation

- 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

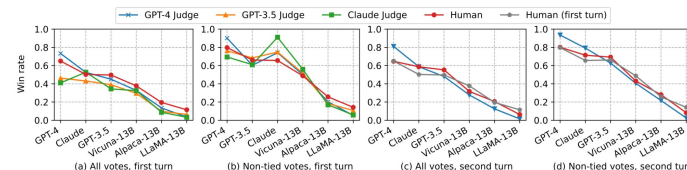
- High agreement with human
- scalability
- Interpretability



<https://www.databricks.com/blog/LLM-as-a-Judge-to-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



Analogous to **integration** tests

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 **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

- Need to deploy system & collect

Relatively easy

User
Feedback

- Need to deploy system & collect

**Require good UX for
good data**

“Golden”
Context

- Need labelers

**Relatively
cheap/easy**

“Golden”
Answer

- Need labelers

**More
costly/tedious**

“Cold Start” Problem

Dilemma

We have the chicken and egg problem of:

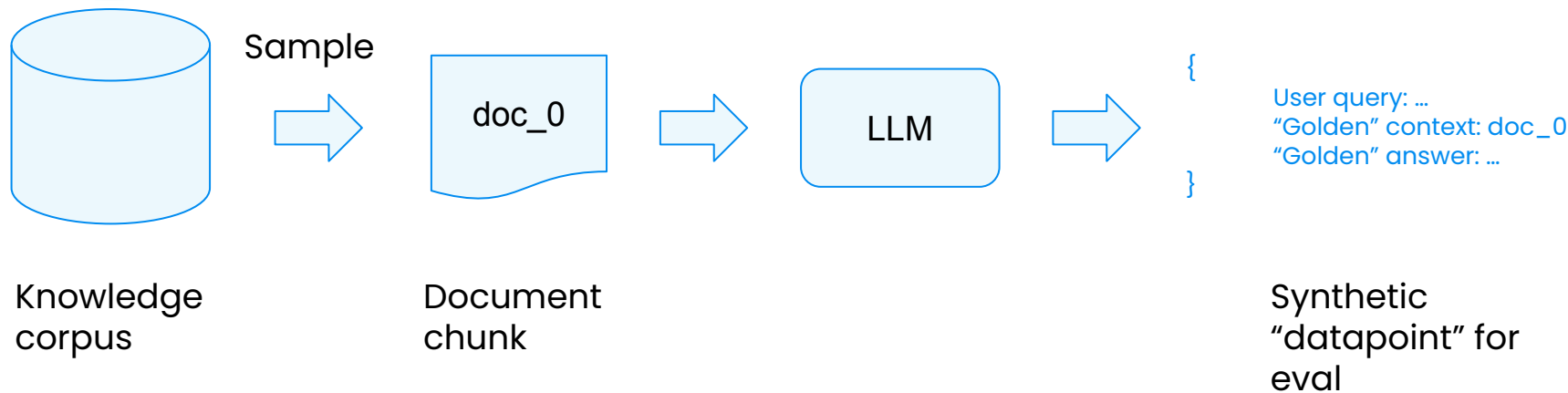
1. We want to evaluate to gain confidence of our RAG system before deployment
2. Without deployment, we can't collect real user queries and label an evaluation dataset

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



The development process

At this stage

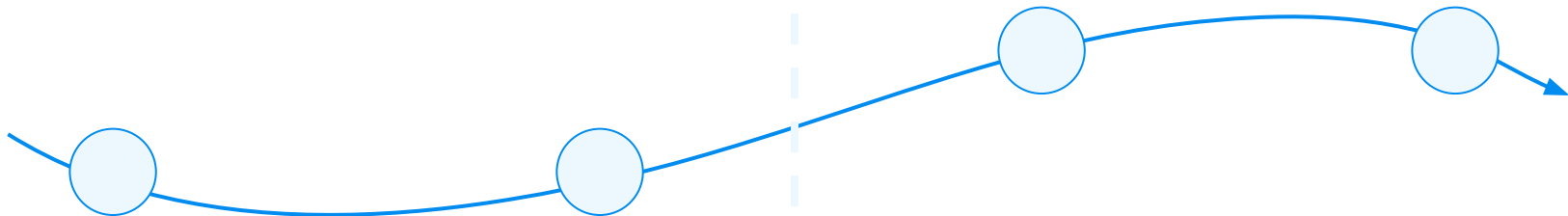
- Have user queries, maybe labels & feedback
- Need repeatable, consistent, automated evaluation
- Need actionable insight or automated tuning

Deployment

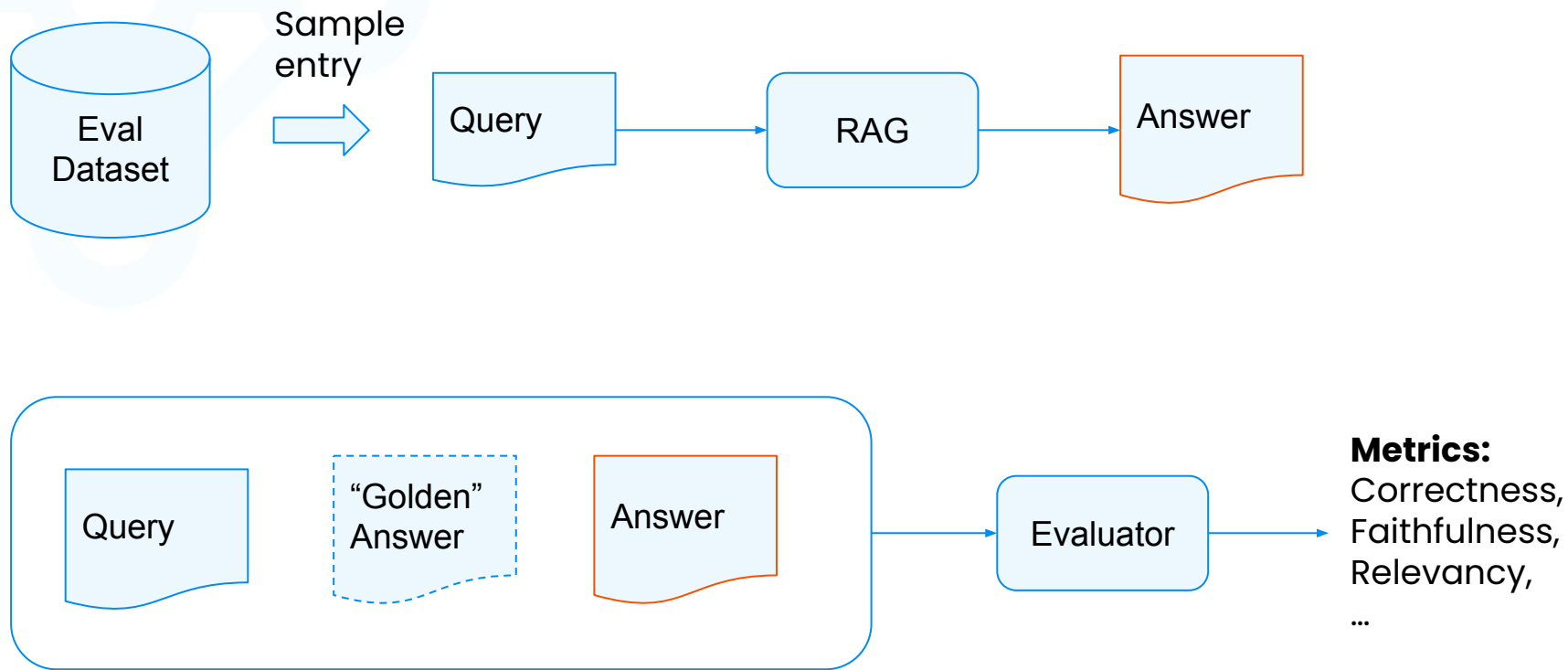
Deploy system & collect user queries & feedback

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.

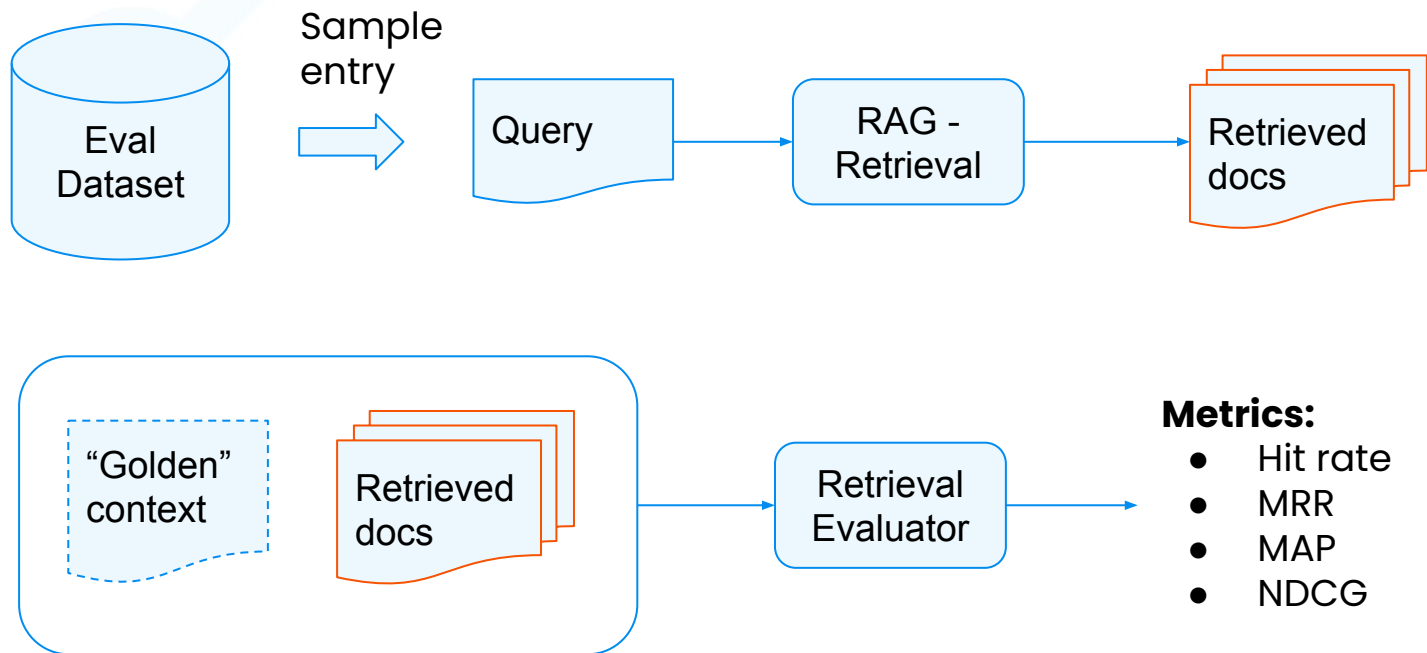
User Query
{query}

Reference Answer
{reference_answer}

Generated Answer
{generated_answer}

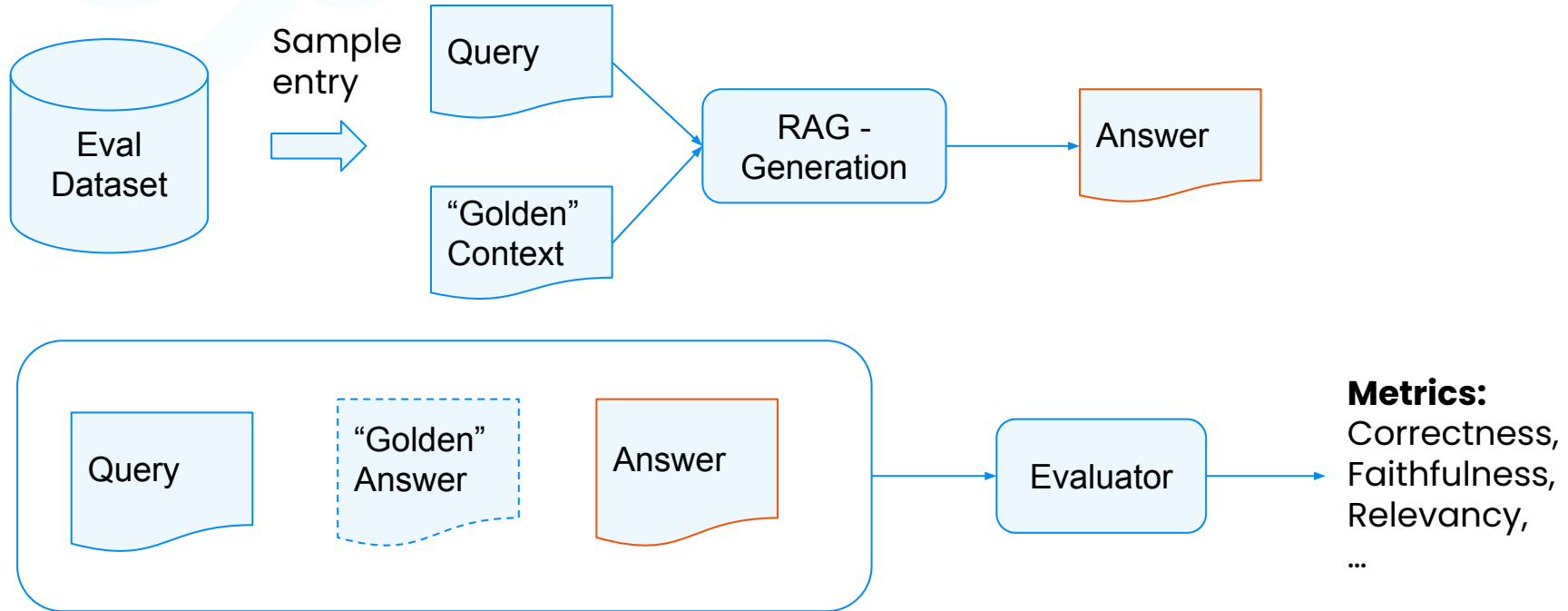
Component-wise evaluation

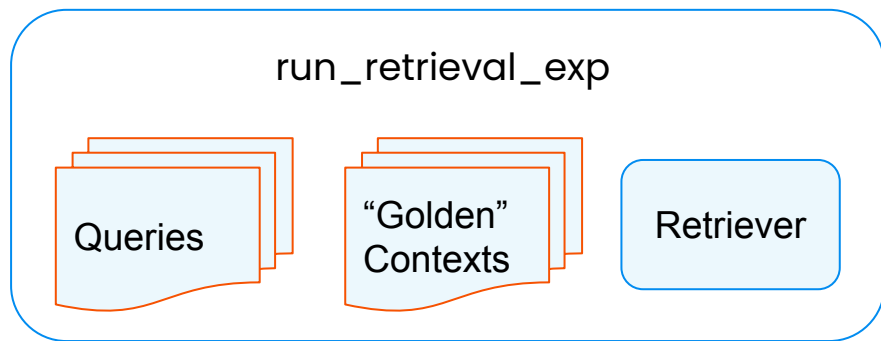
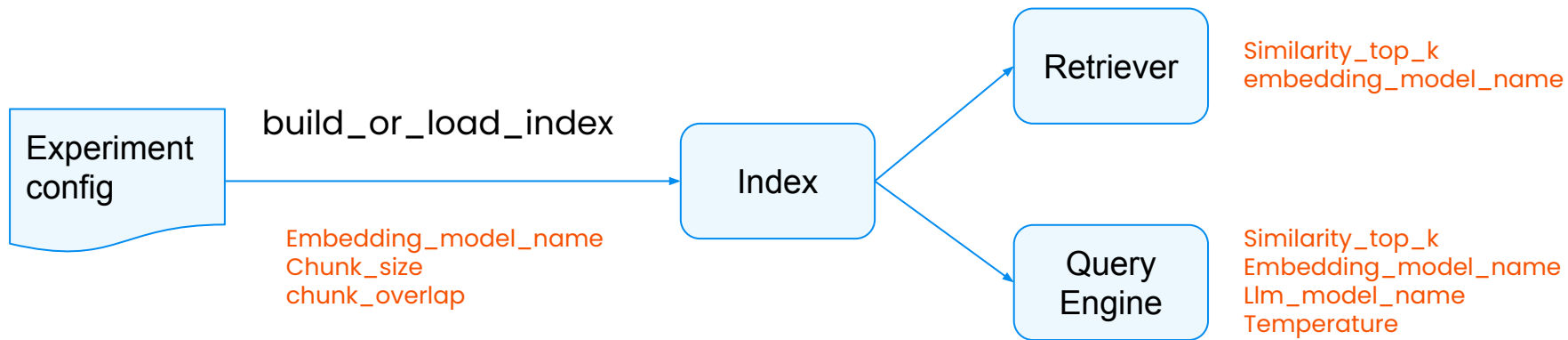
- Evaluate **retrieval** component



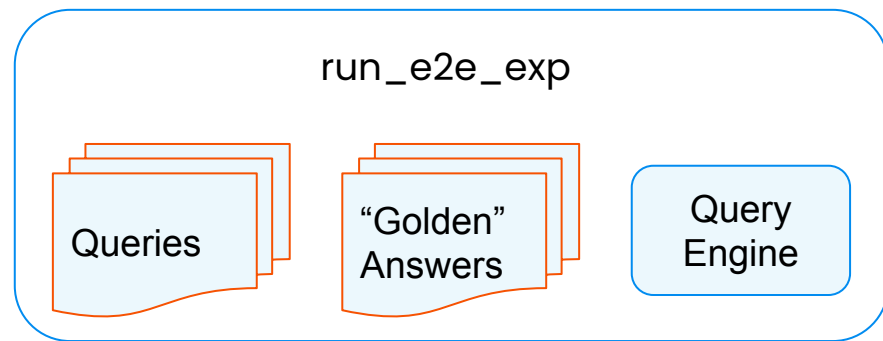
Component-wise evaluation

- Evaluate **generation** component





Hit rate



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

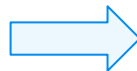
Generation

- LLM
- (prompt)

Grid search

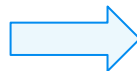
- Run all parameter combinations and evaluate the end performance of system

```
{  
  Chunk_size: 512,  
  Top_k: 5,  
  Llm: gpt-4,  
  ...  
}
```

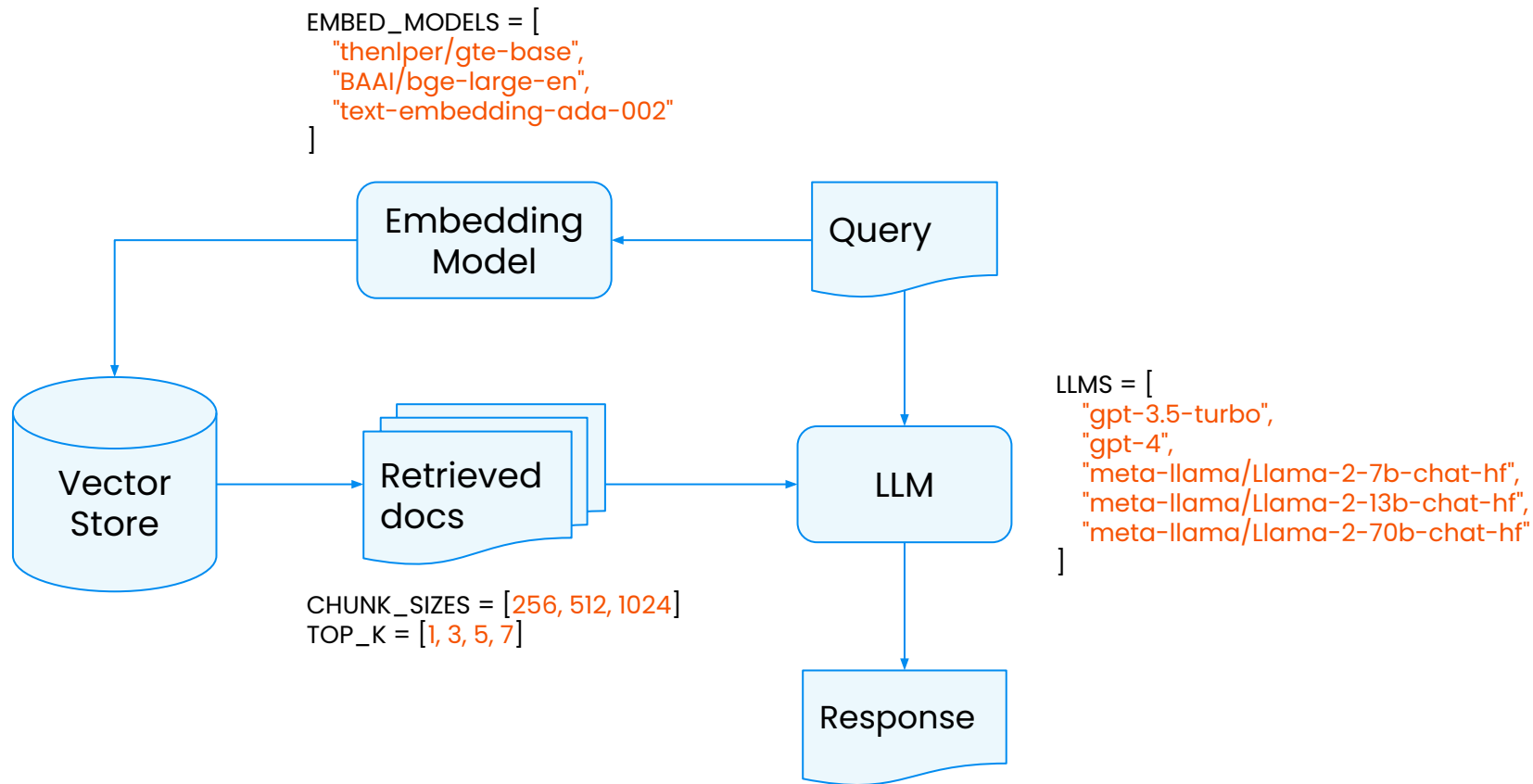


Score: 4.5

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



Score: 3.5



Lab

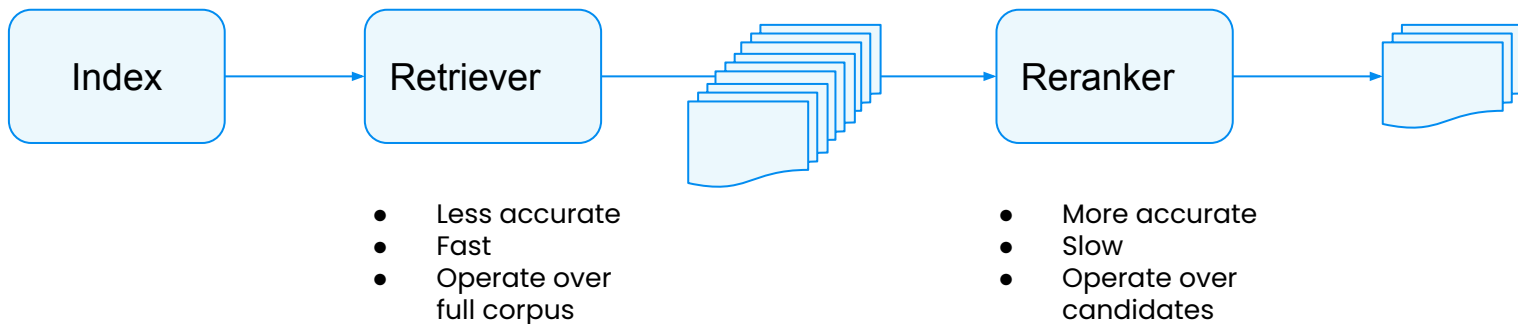
- Run experiments over standard component configurations
- Gain intuition on optimal parameter configurations

Customizing Retrieval & Generation

Strategy 1:

Use two-stage retrieval:

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



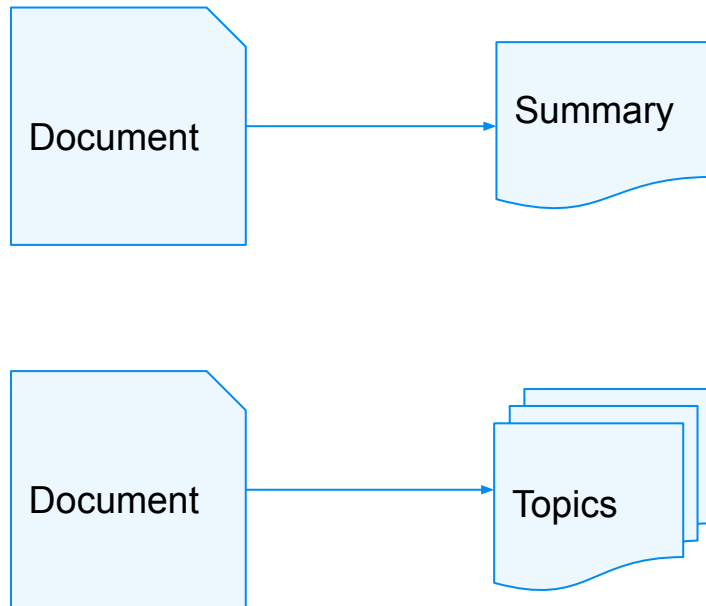
Customizing Retrieval & Generation

Strategy 2a:

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

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

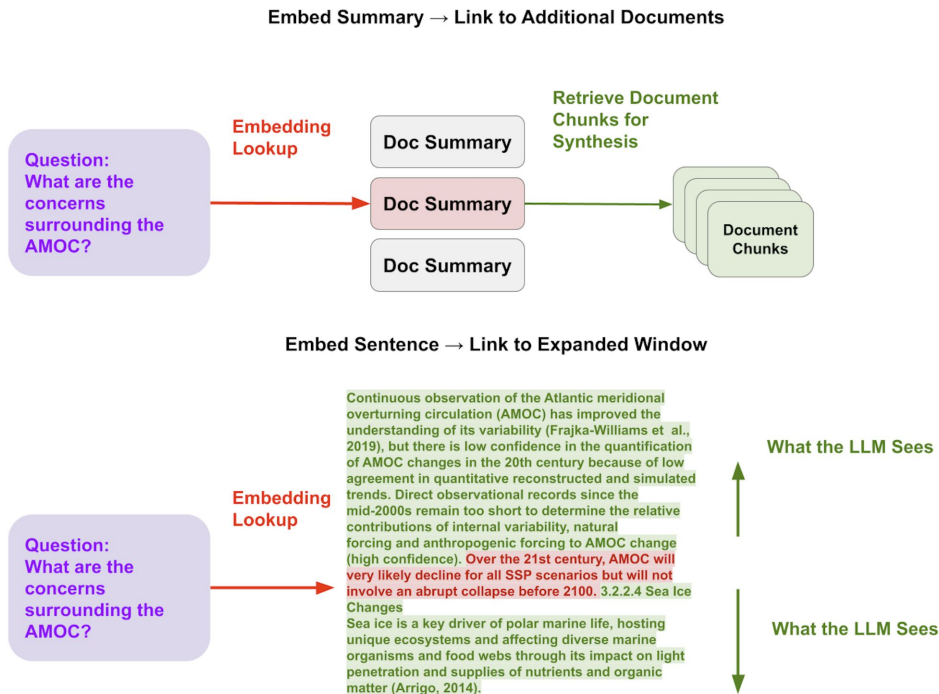
This can improve retrieval over specific questions



Customizing Retrieval & Generation

Strategy 2b:

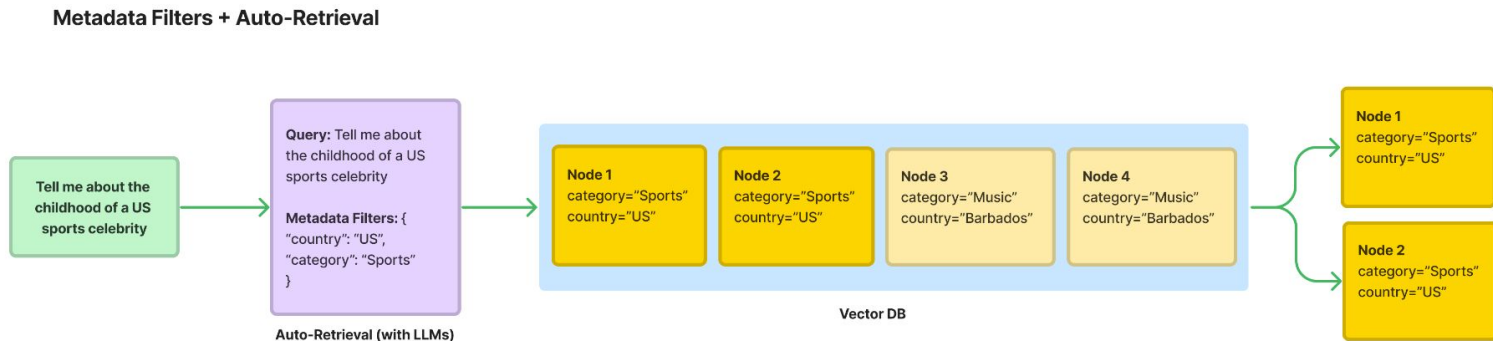
Use different data representation for retrieval & for generation



Customizing Retrieval & Generation

Strategy 3:

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

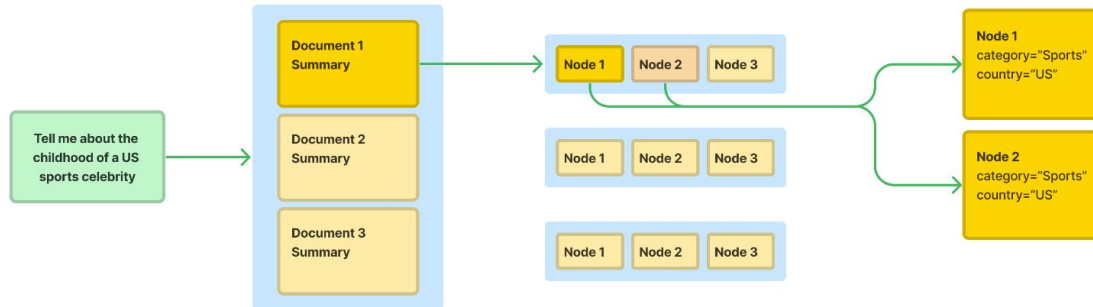


Customizing Retrieval & Generation

Strategy 4:

Recursive retrieval over hierarchical index

Document Hierarchies (Summaries + Raw Chunks) + Recursive Retrieval

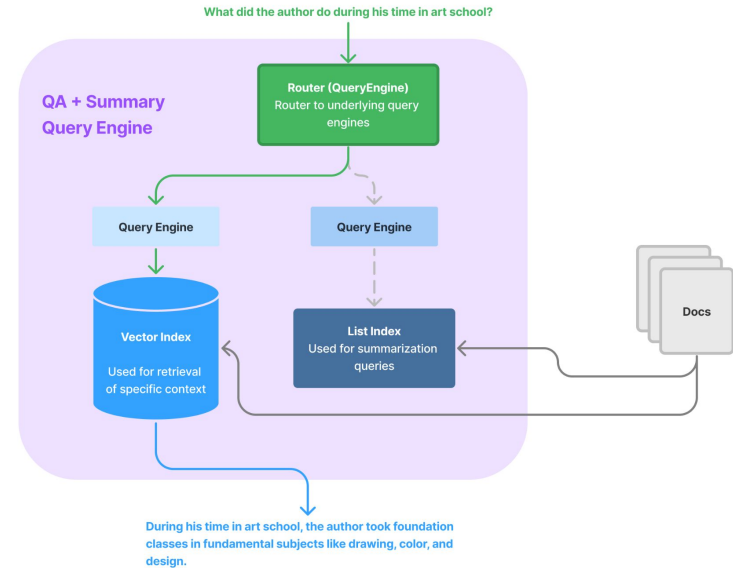


Customizing Retrieval & Generation

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

Not great for injecting new knowledge and fighting hallucination

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

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

Lab

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

Get In Touch



Sign up for office hours

Get our weekly newsletter!

Get a weekly roundup of the latest news and insights on the world of LLMs and word on the newest features of the LlamaIndex libraries.

☐ I accept [terms and conditions](#)

Sign up

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