Stat 359
Modern Deep Learning

Lecture 9
RAG

1/28/25

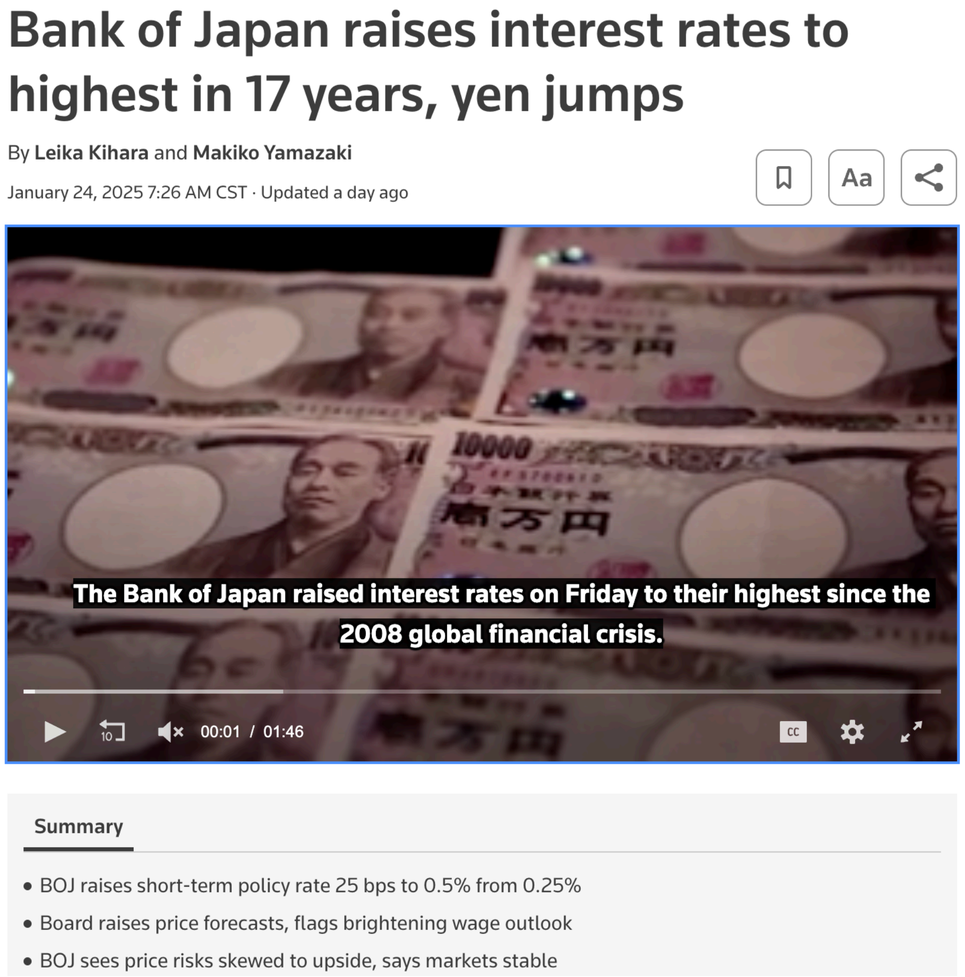
Northwestern

Report Generation

- Scenario: we want LLMs to generate a report on some topic.
- For example:
  - Volatile sectors in 2025 across the globe.
  - Tech sector undervalued companies over the next decade.
  - Advances in robotics in 2024
  - Bank of Japan’s stance on interest rate changes in 2025.

Report Generation

- Bank of Japan’s stance on interest rate changes in 2025.
- How would a human generate such a report?
- The first step is usually to conduct research on the topic. A lot of research, if you’re a good analyst.
  - News, market consensus and forecasts, macro indicators.



Report Generation

- Research leads to some answers, and many more questions.
- Eventually, you synthesize all the information you gather into a coherent viewpoint, and publish it as a report.
- **Question**: How can we mimic this research and synthesis process in LLMs?

Report Generation

- Most naive idea.
- Have an LLM read all of the day's news.
- Ask the LLM to summarize the news, finding and distilling relevant information.
- Then take these summaries, and use them to synthesize a coherent viewpoint.

Report Generation

- Problem
- There is a lot of news.
- Around 78,000 news articles are published per day across reputable media outlets.
- We might also have our own internal documents.
- You can fit about 500 two-page articles into the largest possible LLM context window — Claude Opus 200k context.
- Around 300 LLM calls per day to summarize that day’s news.

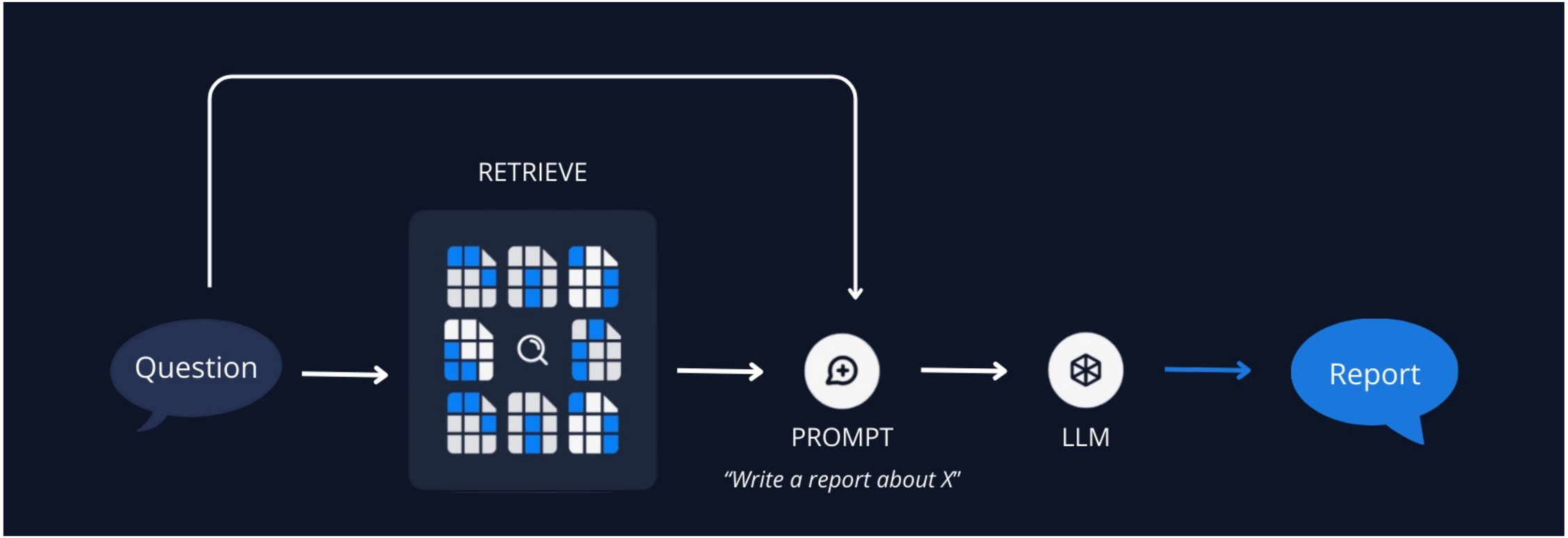
Report Generation

- This strategy of reading all news is not viable.
- In practice, LLM’s get confused when the amount of irrelevant information in the context window increases.
- You’ll have a hard time knowing which of the 500 articles in each LLM call are most important. LLMs won’t cite sources.
- Also, the LLM won’t be able to synthesize information across batches. Each batch of articles are processed separately.

Report Generation

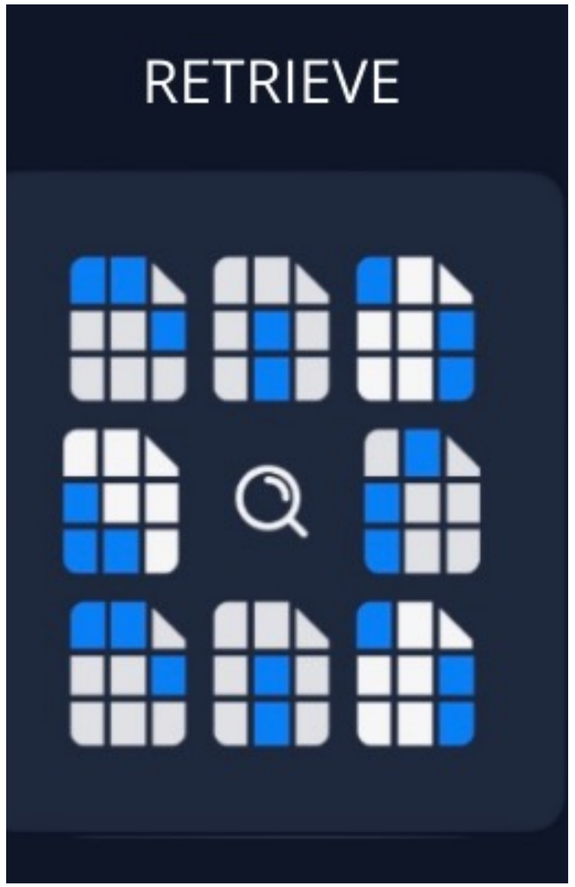
- Solution.
- We don’t want to give the LLM access to all the news that is published.
- Instead, we want to selectively pre-filter the news we give to the LLM.
  - We should only give the LLM news that is relevant for generating the desired report.

Report Generation



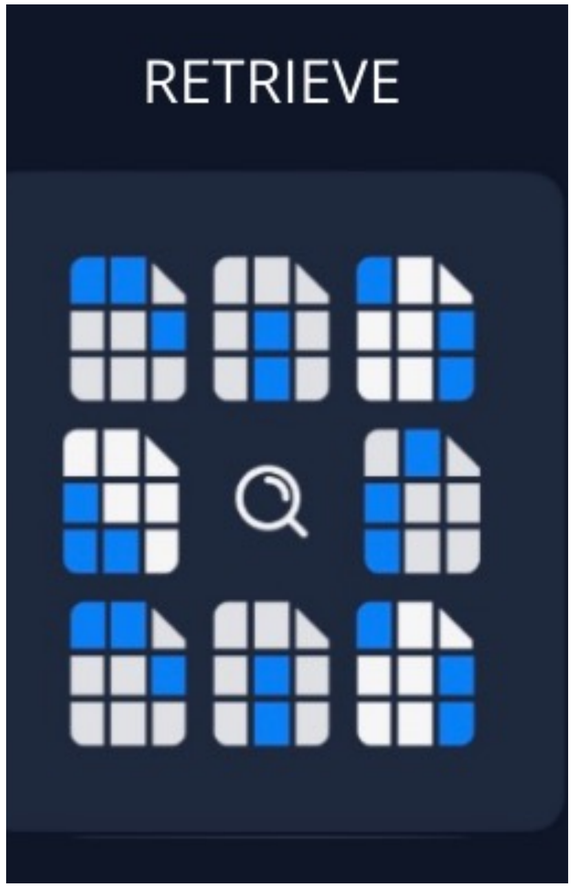
Report Generation

- Retrieval is an example of an LLM tool.
- Tools are external helpers LLMs can call upon to help them solve a problem.
- What makes a good firm?
  - Data. Infrastructure. And people with good skills.



Report Generation

- I content that an LLM is only as good as the tools you provide it.
- Without good tools, there is little differentiation between your LLM and everyone else's LLM.
- Aaron will talk more about some tools we use in our pipelines.
- For now, let's focus on a retrieval tool.



Report Generation

- How can we find the news that is most relevant for generating our report?

Report Generation

- How can we find the news that is most relevant for generating our report?
- Classical techique: Lexicographical Matching

TF matching

TF matching

- Suppose we start with some large corpus of documents.
- Given a simplified query
  - Japan Changing Bond Rates
  - Break the query into a list of words: Bank, of, Japan, Changing,

    Bond, Rates

TF matching

- Break the query into a list of words: Bank, of, Japan, Changing, Bond,

  Rates

- For each document, compute the frequency over all words in the query.

  - Document one contains 500 total words. 10 of those words are **bank**.

  - Thus, the total TF for bank is 10/500.

  - 20 of the words are **Japan**. Thus, the TF of Japan is 20/500.

TF matching

- Break the query into a list of words: Bank, of, Japan, Changing,

  Bond, Rates

- Compute the average TF across all words in the query.

$$TF_1 = \frac{\overbrace{20/500}^{\text{Bank}} + \overbrace{17/500}^{\text{of}} + \overbrace{20/500}^{\text{Japan}} + \overbrace{0/500}^{\text{Changing}} + \overbrace{5/500}^{\text{Bond}} + \overbrace{3/500}^{\text{Rates}}}{6} = 1.41$$

TF matching

- We can do this same calculation over all documents.

$$TF_1 = \frac{\overset{\text{Bank}}{20/500} + \overset{\text{of}}{17/500} + \overset{\text{Japan}}{20/500} + \overset{\text{Changing}}{0/500} + \overset{\text{Bond}}{5/500} + \overset{\text{Rates}}{3/500}}{6} = 1.41$$

$$TF_2 = \frac{0/500 + 13/500 + 5/500 + 0/500 + 3/500 + 3/500}{6} = 1.02$$

- Once we have all TF scores, return the documents with the highest TF for our query.

TF matching — Drawbacks

TF matching — Drawbacks

- Now, is TF matching used in practice?
- Not really.
- The problems are obvious, right?

TF matching — Drawbacks

- The problems are obvious, right?
- Bank, of, Japan, Changing, Bond, Rates
- We’re giving equal weighting to words such as “of”
- We’re looking only at individual words. But often combinations of words are important. e.g. “bond rates”
- Most importantly, we’re completely ignoring the idea of semantic similarity.

TF matching — Drawbacks

- Semantic similarity
- When I give you a query, I want to find all documents that are talking about that topic, even if those documents don’t match the exact wording of my query.

CHICAGO BULLS

TF matching — Drawbacks

- Example
  - I search for “potential bull markets 2025”
  - The articles “Which industries will rally in 2025?” and “"Rising sectors poised for 2025 growth” are both good matches.
  - If you use TF matching, you will likely get articles about the basketball team named “The Bulls” or the animal.

CHICAGO BULLS

Semantic Similarity Matching

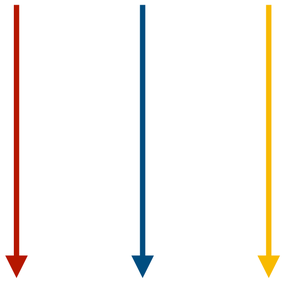
- How can we build a query tool that respects semantic similarity?
- Core idea
  - Find some embedding function that maps a sentence to a list of numbers.
  - $f_\theta(\text{Markets are booming}) \rightarrow [0.1, 0.9, 0.2]$
  - Sentences that are similar to one another should go to lists of numbers that are also close.
  - $f_\theta(\text{Markets continue to increase}) \rightarrow [0.2, 0.8, 0.2]$

Semantic Similarity Matching

- How do we know when two lists of numbers are close to one another?
- Just subtract them, then sum the absolute value.
- [0.1,0.9,0.2]
- [0.3,0.5,0.8]

Semantic Similarity Matching

- How do we know when two lists of numbers are close to one another?
- Just subtract them, then sum the absolute value.
- [0.1,0.9,0.2]



- [0.3,0.5,0.8]

$$0.1 - 0.3 = -0.2$$

$$0.9 - 0.5 = 0.4$$

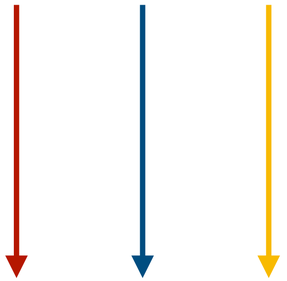
$$0.2 - 0.8 = -0.6$$

Semantic Similarity Matching

- How do we know when two lists of numbers are close to one another?

- Just subtract them, then sum the absolute value.

- [0.1,0.9,0.2]



- [0.3,0.5,0.8]

$$0.1 - 0.3 = -0.2$$

$$0.9 - 0.5 = 0.4$$

$$0.2 - 0.8 = -0.6$$

difference $= 0.2 + 0.4 + 0.6$

Semantic Similarity Matching

- Sum and then absolute value gives us a way to tell how close sentences are:
  - $f_\theta(\text{Markets are booming}) \rightarrow [0.1, 0.9, 0.2]$
  - $f_\theta(\text{Booming fireworks tomorrow}) \rightarrow [0.9, 1.5, 2.3]$
  - $f_\theta(\text{Markets continue to increase}) \rightarrow [0.2, 0.8, 0.2]$

Semantic Similarity Matching

- f_θ(Markets are booming) $\rightarrow$ [0.1,0.9,0.2]
- f_θ(Booming fireworks tomorrow) $\rightarrow$ [0.9,1.5,2.3]
- f_θ(Markets continue to increase) $\rightarrow$ [0.2,0.8,0.2]
- Δ([0.1,0.9,0.2] - [0.9,1.5,2.3]) = 3.5
- Δ([0.1,0.9,0.2] - [0.2,0.8,0.2]) = 0.2

Semantic Similarity Matching

- f_θ(Markets are booming) $\rightarrow$ [0.1,0.9,0.2]
- f_θ(Booming fireworks tomorrow) $\rightarrow$ [0.9,1.5,2.3]
- f_θ(Markets continue to increase) $\rightarrow$ [0.2,0.8,0.2]
- Δ([0.1,0.9,0.2] - [0.9,1.5,2.3]) = 3.5
- Δ([0.1,0.9,0.2] - [0.2,0.8,0.2]) = 0.2

Markets are booming

and

Markets continue to increase

are close

Semantic Similarity Matching

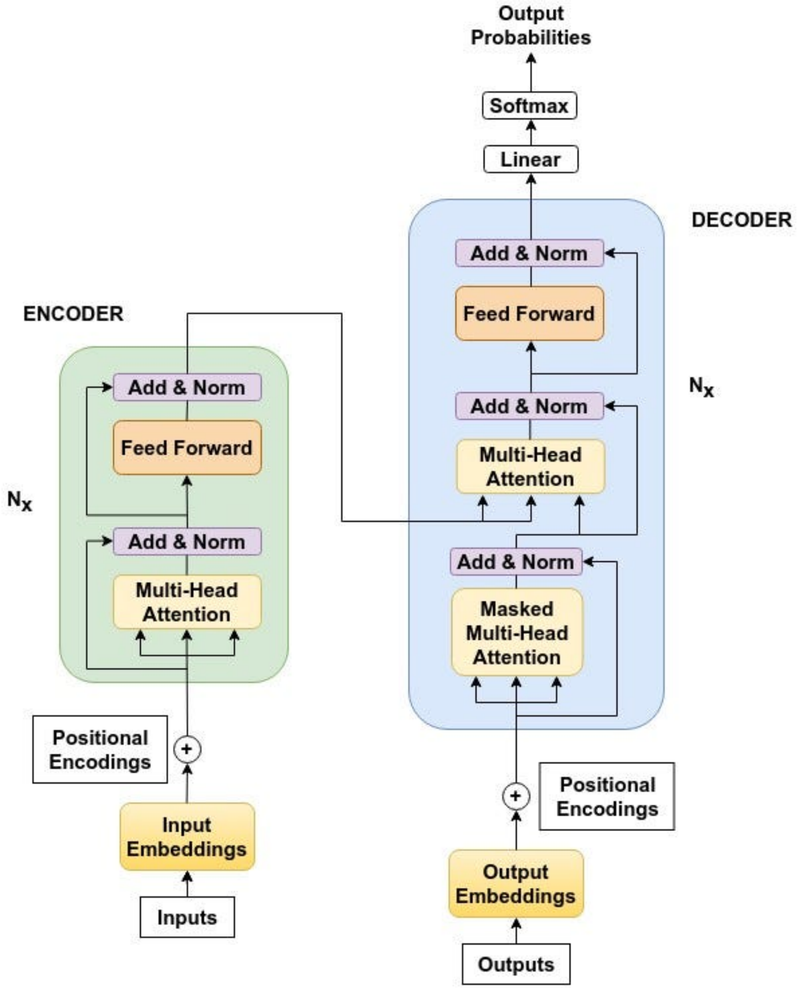
- Major problem.
- We just assumed we had some magic function f_θ that maps sentences to a list of numbers that respect semantic similarity.
- Do you know such a thing exists?
- Yes, I do. And you do as well.
- You use it every day.
- LLMs provide us with such a magic function.

Semantic Similarity Matching

- How LLMs work.
- Take some text.
- Use some function

  $f_\theta(\text{your text}) \rightarrow [0.1, 0.4, 0.9]$

- This vector is used to predict what the next word is in the sentence.
- To learn this skill effectively, f_θ is learns to map sentences to vectors that respect semantic similarity.



Semantic Similarity Matching

- Is there any more to it, or do we literally just steal f_θ from LLMs?
- There is some subtlety.
- Typically, we take an LLM f_θ and send it to the gym.
  - f_θ is like a world class athlete that needs to do some specialized exercises to prepare for the olympics.
  - We need f_θ to get really good at telling documents apart. We refer to this process as training or tuning.
- How do we train f_θ from an LLM?
- Well, we play a few games.

Game 1 — Triplet Loss Function

- Collect by triplets of the form

  (anchor, positive, negative)

- Anchor

  Bull market expected in tech sector 2025

- Positive

  Technology stocks showing strong upward

  potential next year

- Negative

  Bear market continues in real estate

Game 1 — Triplet Loss Function

- Collect by triplets of the form

  (anchor, positive, negative)

- Anchor

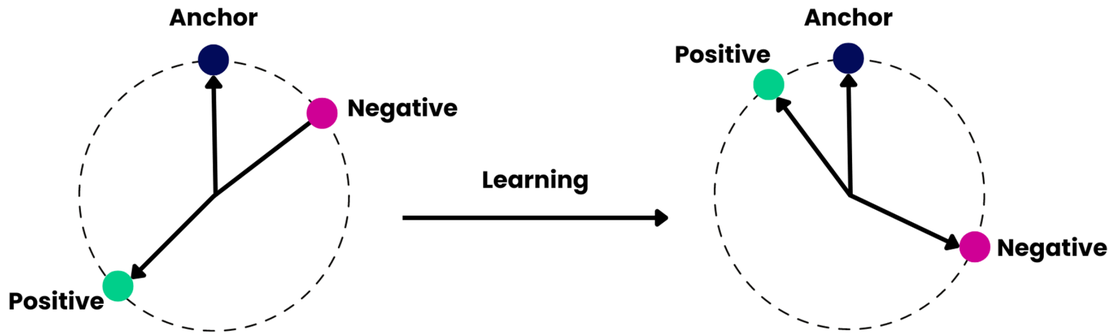
  Bull market expected in tech sector 2025

- Positive

  Technology stocks showing strong upward potential next year

- Negative

  Bear market continues in real estate



- Train f_θ so that the anchor and positive are closer than the anchor and negative.

Semantic Similarity Matching

```
def triplet_loss_finance():
    # Distance between similar market sentiments should be small
    d_pos = distance("Bull market expected in tech sector 2025",
                     "Technology stocks showing strong upward potential next year")

    # Distance between opposite market sentiments should be large
    d_neg = distance("Bull market expected in tech sector 2025",
                     "Bear market continues in real estate")

    loss = max(0, d_pos - d_neg + margin)
```

Multiple Negative Ranking Loss

Multiple Negative Ranking Loss

- Collect data of the form

- Anchor

  Fed likely to raise rates in Q3

- Positive

  Central bank signals hawkish policy stance

- Negatives:

  [“Crypto markets reach all-time high",

  "Oil prices decline on supply news",

  "Tech startup announces IPO plans"]

Multiple Negative Ranking Loss

- Collect data of the form
- Anchor

  Fed likely to raise rates in Q3
- Positive

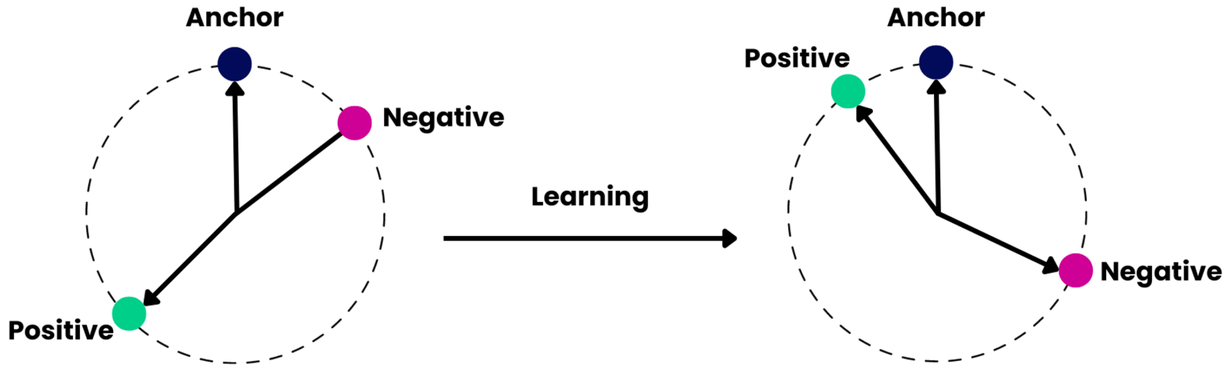
  Central bank signals hawkish policy stance
- Negatives:

  [“Crypto markets reach all-time high",

  "Oil prices decline on supply news",

  "Tech startup announces IPO plans"]

- Ask f_θ the question: Which of these examples is mostly likely to pair with the anchor.
- Train f_θ until it tells you the positive example is most likely.



Multiple Negative Ranking Loss

```python
def mnrl_finance():
    anchor = "Fed likely to raise rates in Q3"
    positive = "Central bank signals hawkish policy stance"
    negatives = ["Crypto markets reach all-time high",
                 "Oil prices decline on supply news",
                 "Tech startup announces IPO plans"]

    # Model should recognize monetary policy similarities
    # while distinguishing from unrelated market news
    scores = [similarity(anchor, sent) for sent in [positive] + negatives]
    probs = softmax(scores)
    loss = -log(probs[0])  # Should give highest prob to positive pair
```

What now?

- We sent this f_θ to the gym.
- It's now good at taking two sentences and telling you if they are semantically similar.
- But how do we use f_θ to find good documents for report generation?

LLM Report Generation

- Take a collection of documents and news.
- For every single document, compute an embedding

  $f_\theta(\text{doc}_i) = v_i$

  where v_i is some set of numbers, for example $[0.1, 0.5, 0.8]$
- Now. Take the question you want to use to generate a report.

  $q =$ Report on Bank of Japan Interest Rate Stance
- Compute that embedding as well, which is also just a list of numbers.

  $f(q) = v_q$

LLM Report Generation

- Compute the distance between the query embedding and every document embedding
- $D_1 = \Delta(v_1 - v_q)$

  $D_n = \Delta(v_n - v_q)$
- Find the documents that have the smallest distance to the query.
- This is the news that you will give to your LLM for report generation.

doc_j

Bank of Japan

BOJ chief surprises with dovish remarks, trimming odds of January hike

Japanese central bank stands pat in bow to sluggish economy
19 December 2024

doc_k

Release of 2024 Cherry Blossom Forecast (14th forecast)

April 25, 2024

Share :

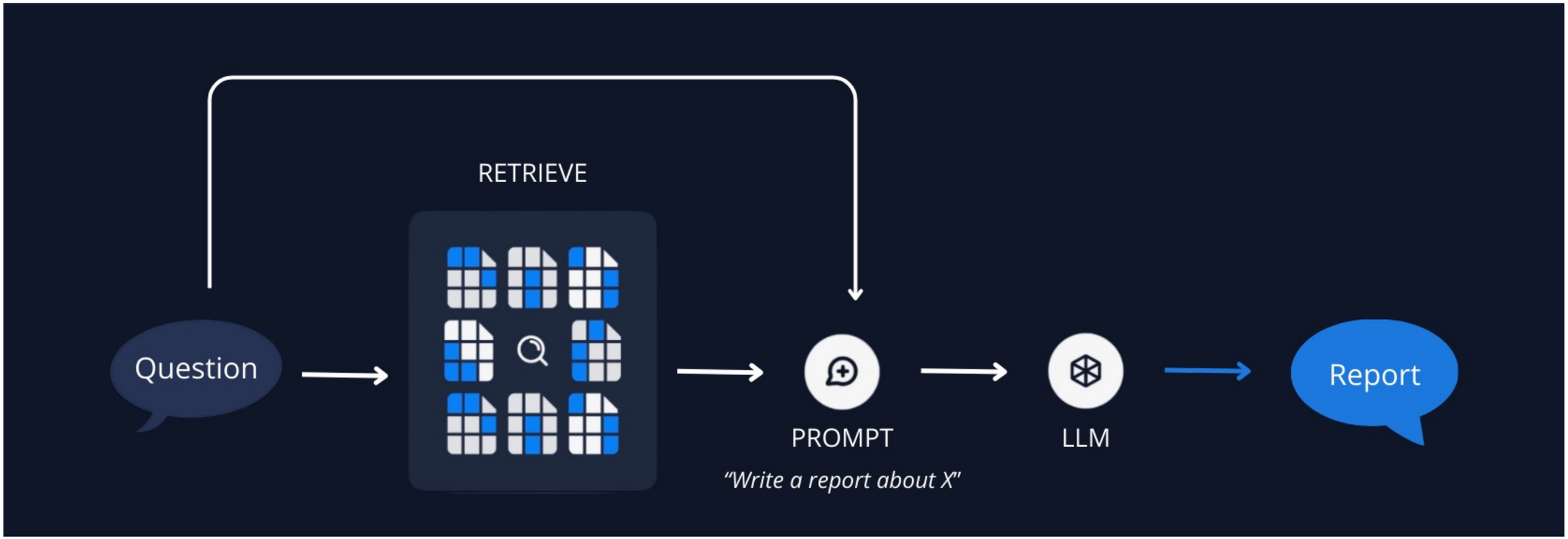
日本語 English 简体字 繁體字

On April 25, 2024, JMC released its 14th forecast of the dates when cherry blossoms will start to flower (kaika) and reach full bloom (mankai). JMC has estimated the flowering and full bloom dates for Somei Yoshino (Yoshino Cherry) trees in approximately 1,000 cherry blossom viewing locations in cities from Hokkaido to Kagoshima. (This will be the last cherry blossom forecast for this season)

App "Sakura Navi - Forecast in 2024" (App Store / Google Play)

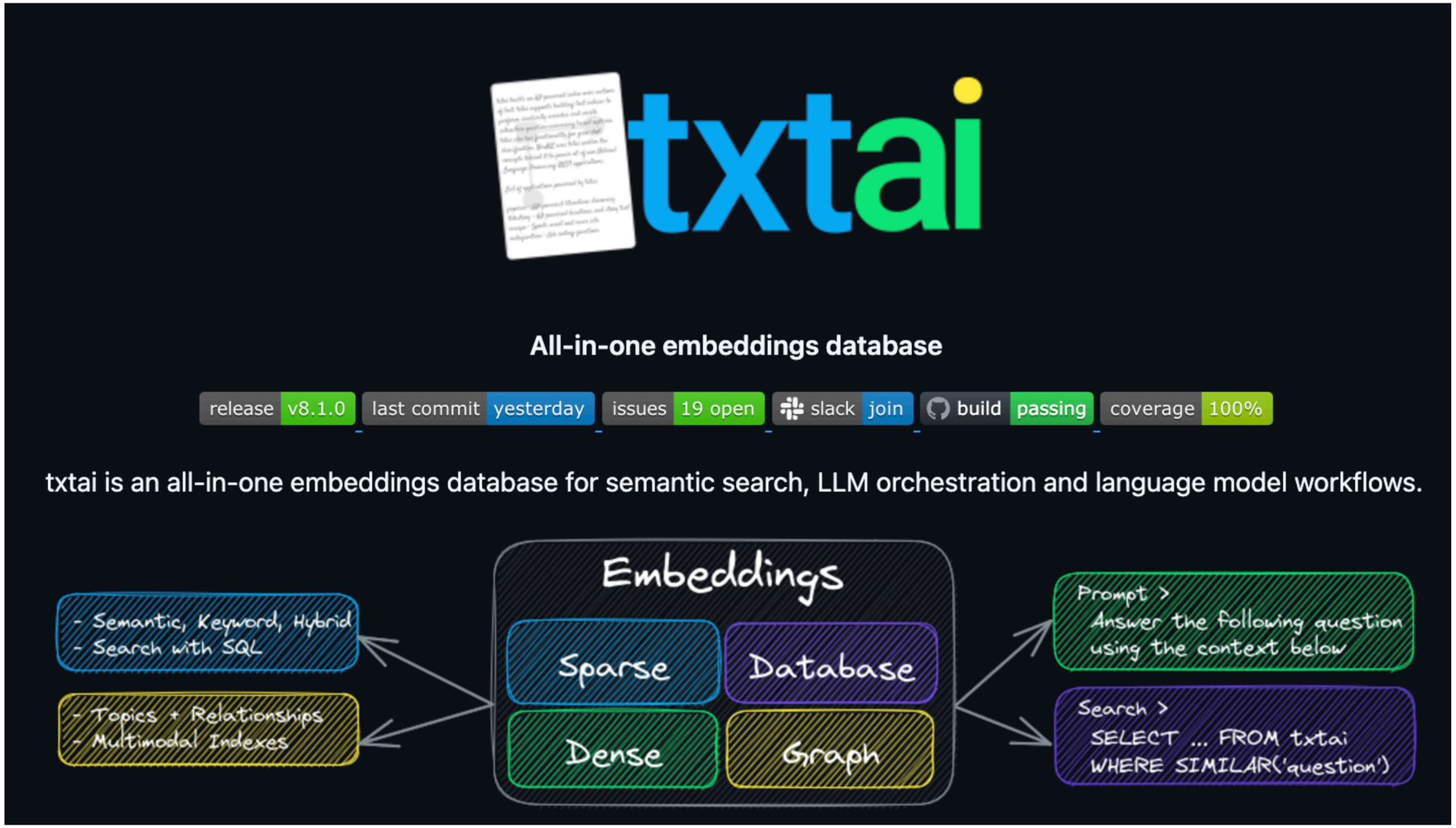
"Sakura Navi" won the first place in the AppStore travel category (paid) in Hong Kong in 2019.

Report Generation



Off the Shelf Tools for Doing This

Off the Shelf Tools for Doing This



Off the Shelf Tools for Doing This

🗂️ LlamaIndex 🦙

downloads 3M/month | contributors 478 | chat 2k online | Phorm Ask AI

LlamaIndex (GPT Index) is a data framework for your LLM application. Building with LlamaIndex typically involves working with LlamaIndex core and a chosen set of integrations (or plugins). There are two ways to start building with LlamaIndex in Python:

1. **Starter**: `llama-index`. A starter Python package that includes core LlamaIndex as well as a selection of integrations.

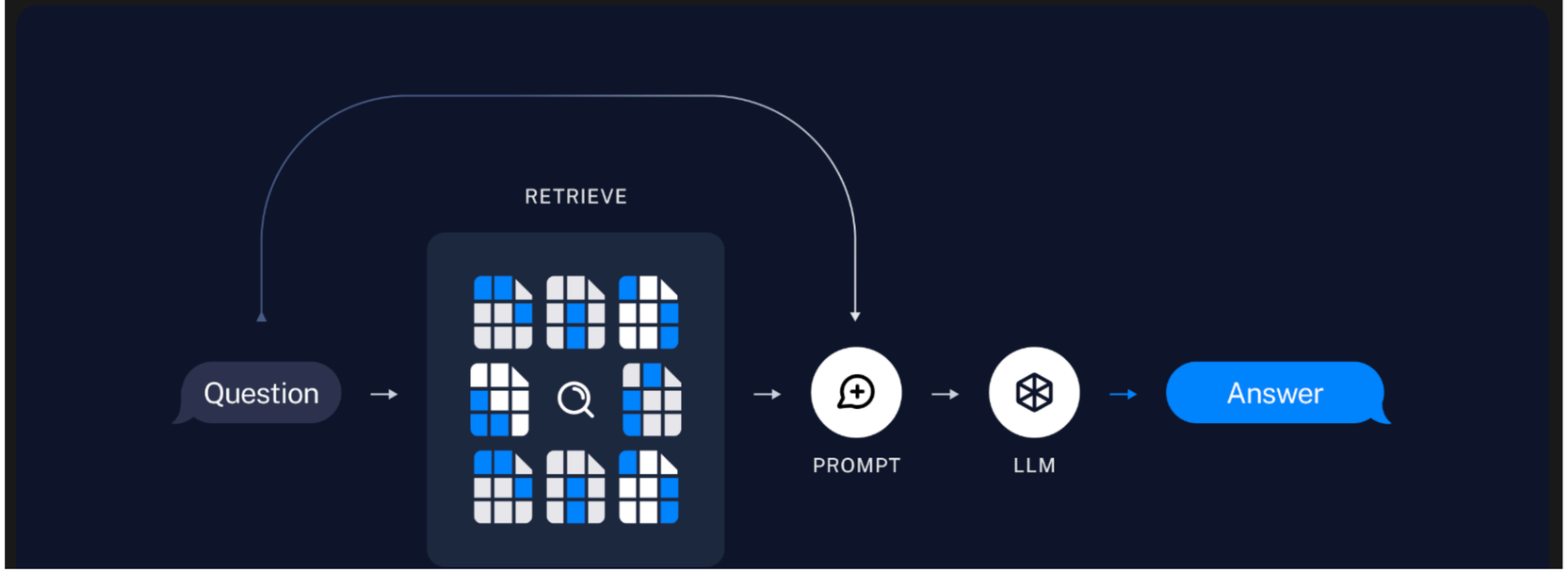
2. **Customized**: `llama-index-core`. Install core LlamaIndex and add your chosen LlamaIndex integration packages on LlamaHub that are required for your application. There are over 300 LlamaIndex integration packages that work seamlessly with core, allowing you to build with your preferred LLM, embedding, and vector store providers.

Off the Shelf Tools for Doing This

LangChain

Retrieval and generation

4. **Retrieve**: Given a user input, relevant splits are retrieved from storage using a Retriever.
5. **Generate**: A ChatModel / LLM produces an answer using a prompt that includes both the question with the retrieved data



Off the Shelf Tools for Doing This

PostgreSQL as a Vector Database: A Pgvector Tutorial

OpenAI Research Products Safety Company

January 25, 2024

New embedding models and API updates

We are launching a new generation of embedding models, new GPT-4 Turbo and moderation models, new API usage management tools, and soon, lower pricing on GPT-3.5 Turbo.

SentenceTransformers Documentation

Edit on GitHub

Note

Sentence Transformers v3.2 recently released, introducing the ONNX and OpenVINO backends for Sentence Transformer models. Read SentenceTransformer > Usage > Speeding up Inference to learn more about the new backends and what they can mean for your inference speed.

Note

Sentence Transformers v3.3 just released, introducing training with Prompts. Read SentenceTransformer > Training Examples > Training with Prompts to learn more about how you can use them to train stronger models.

SentenceTransformers Documentation

Sentence Transformers (a.k.a. SBERT) is the go-to Python module for accessing, using, and training state-of-the-art text and image embedding models. It can be used to compute embeddings using Sentence Transformer models (quickstart) or to calculate similarity scores using Cross-Encoder models (quickstart). This unlocks a wide range of applications, including semantic search, semantic textual similarity, and paraphrase mining.

Off the Shelf Tools for Doing This

Retrieval Augmented Generation (RAG) in Azure AI Search

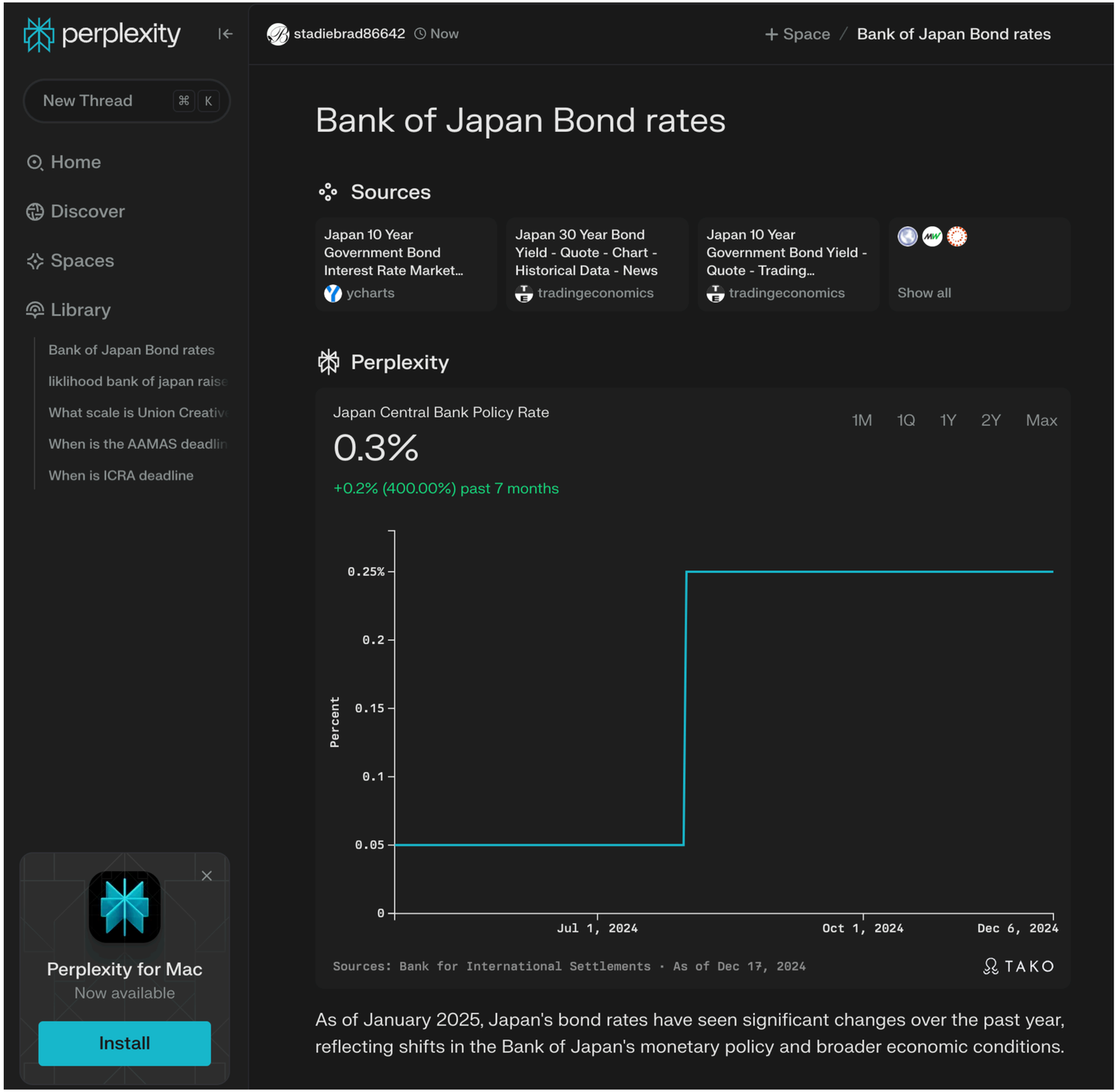
Article • 12/18/2024 • 6 contributors

Feedback

In this article

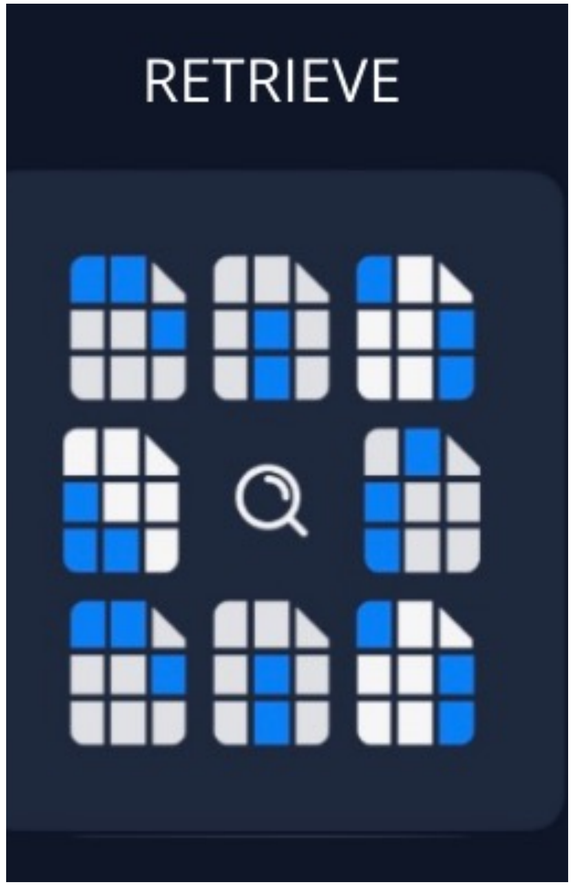
Retrieval Augmented Generation (RAG) is an architecture that augments the capabilities of a Large Language Model (LLM) like ChatGPT by adding an information retrieval system that provides grounding data. Adding an information retrieval system gives you control over

Off the Shelf Tools for Doing This



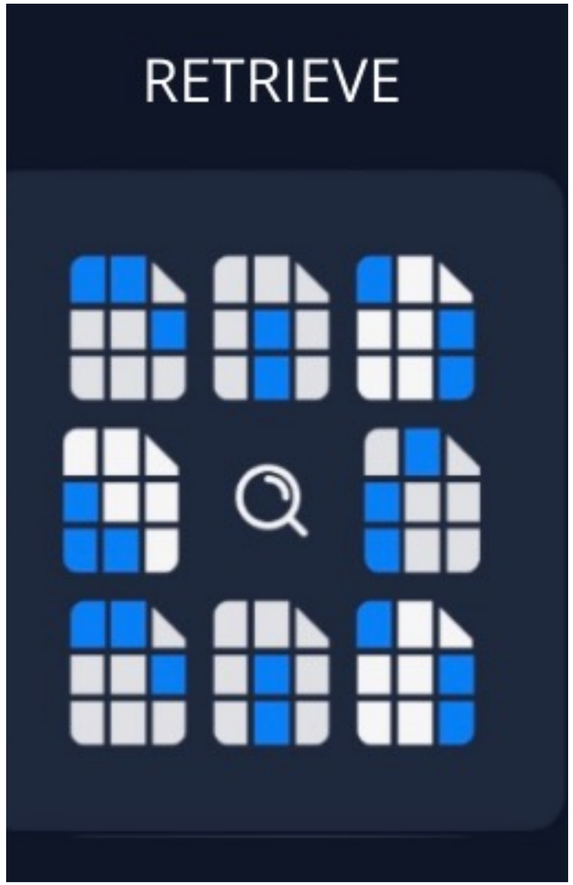
What differentiates our efforts?

- Most obvious thing: We have really good data.
- You need to retrieve over something!
- We have thousands of wires, dealer reports, radar pieces, and so on.
- This data is not generally available.
- Retrieval is a function of your inputs. You'll never find data that doesn't exist.



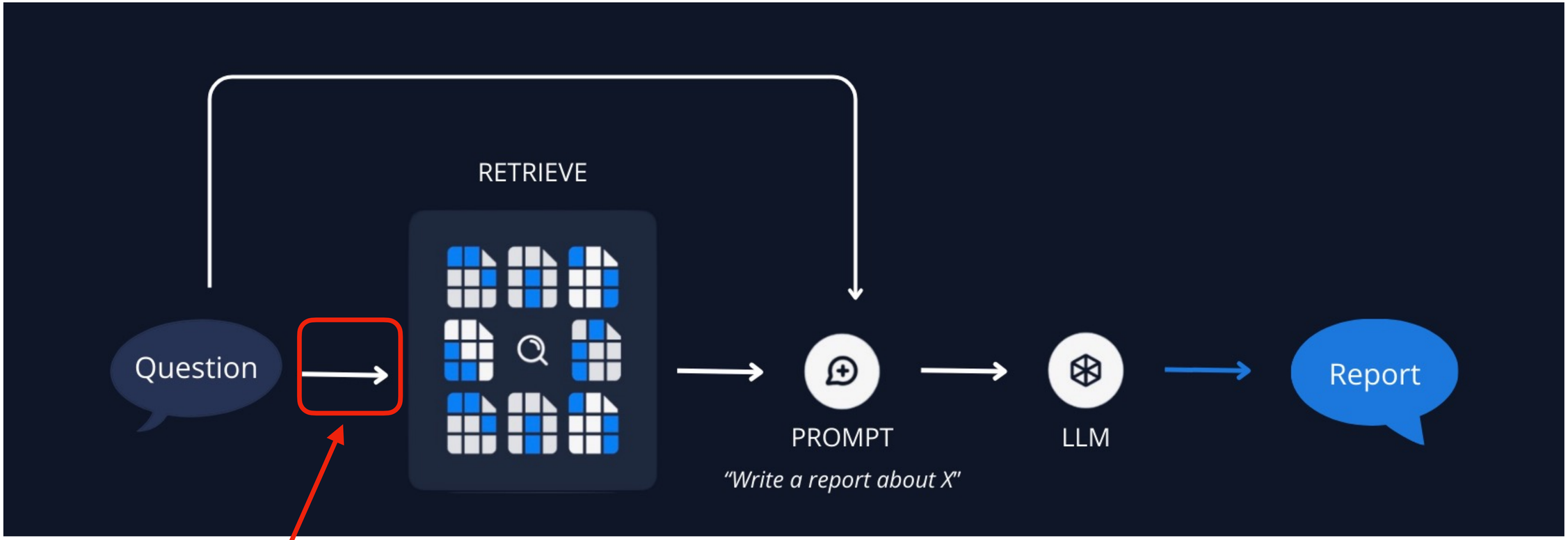
What differentiates our efforts?

- Second point of differentiation.
- It’s actually really hard to measure the strength of a retrieval pipeline.
- You need a lot of hand labeled data to evaluate it.
  - Sets of queries, paired with correct documents.
- We have a lot of expertise in collecting this type of data.
- We also have expertise on modifying the retrieval pipeline with specialized economics knowledge.
  - We send it to the economics gym.

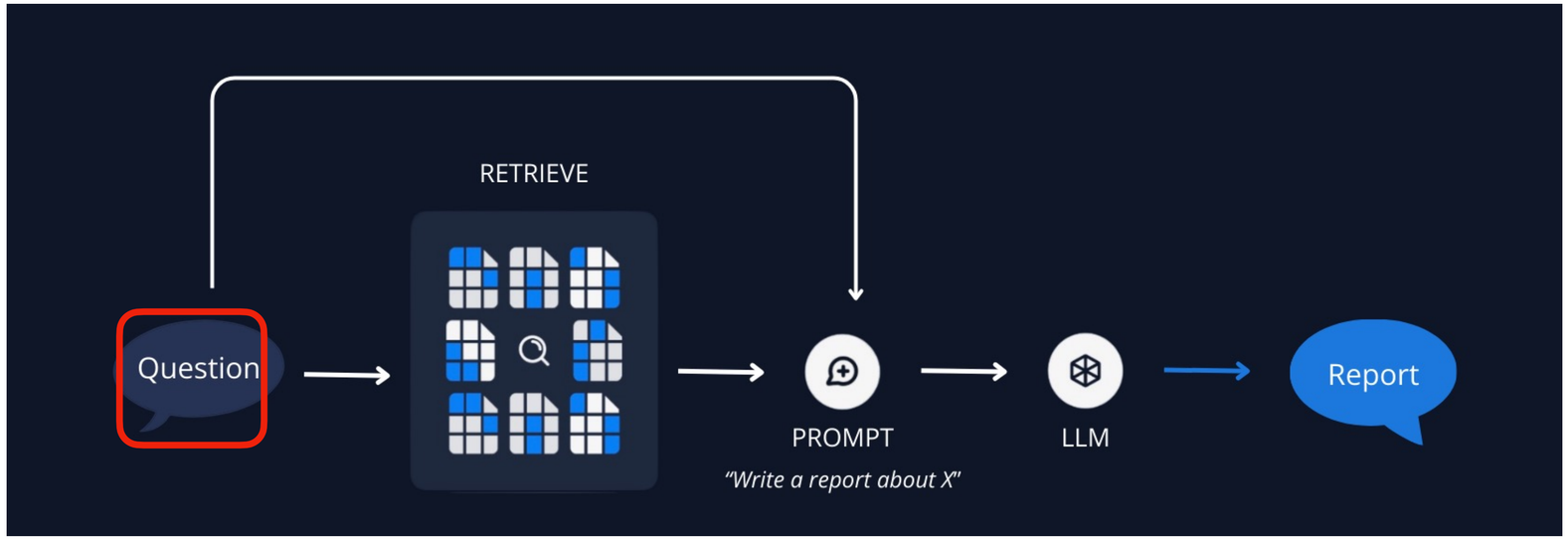


Advanced Report Generation

Advanced Report Generation

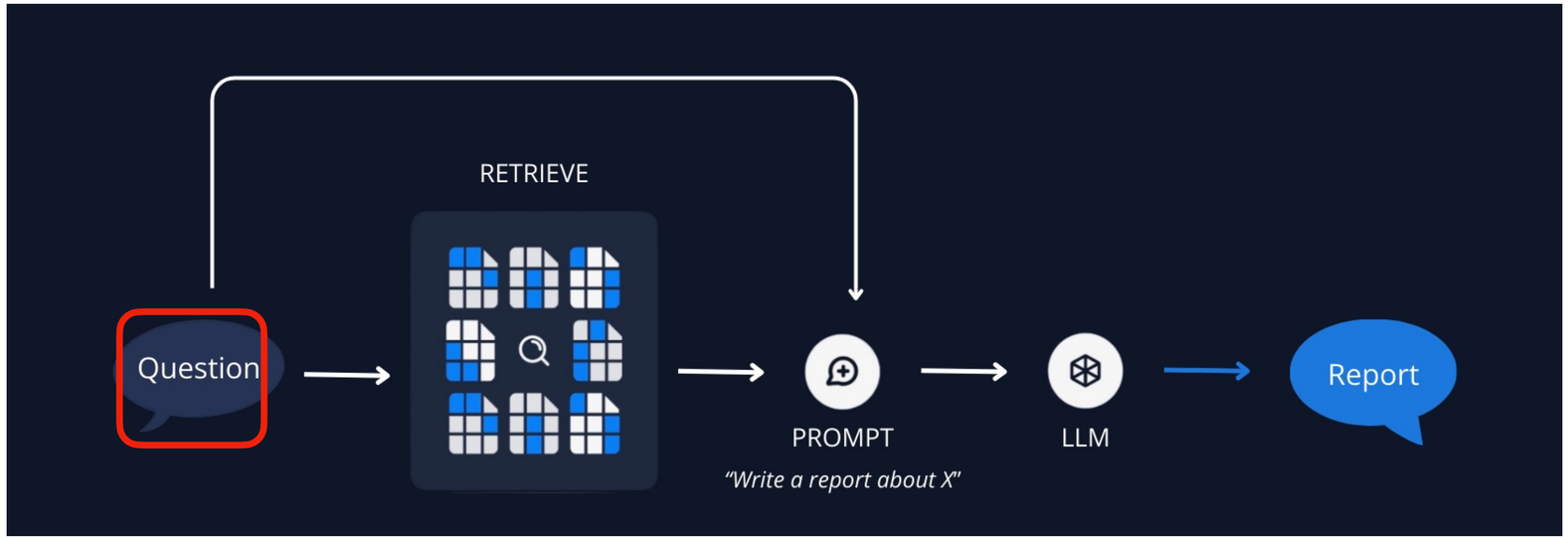


- What question should we feed into the retrieval pipeline?
- Naive answer — Feed in the user query directly.
  - e.g. “Report on Japanese Bond Rates”



- More advanced answer.

- Ask the LLM to output five queries that can be used for retrieval.



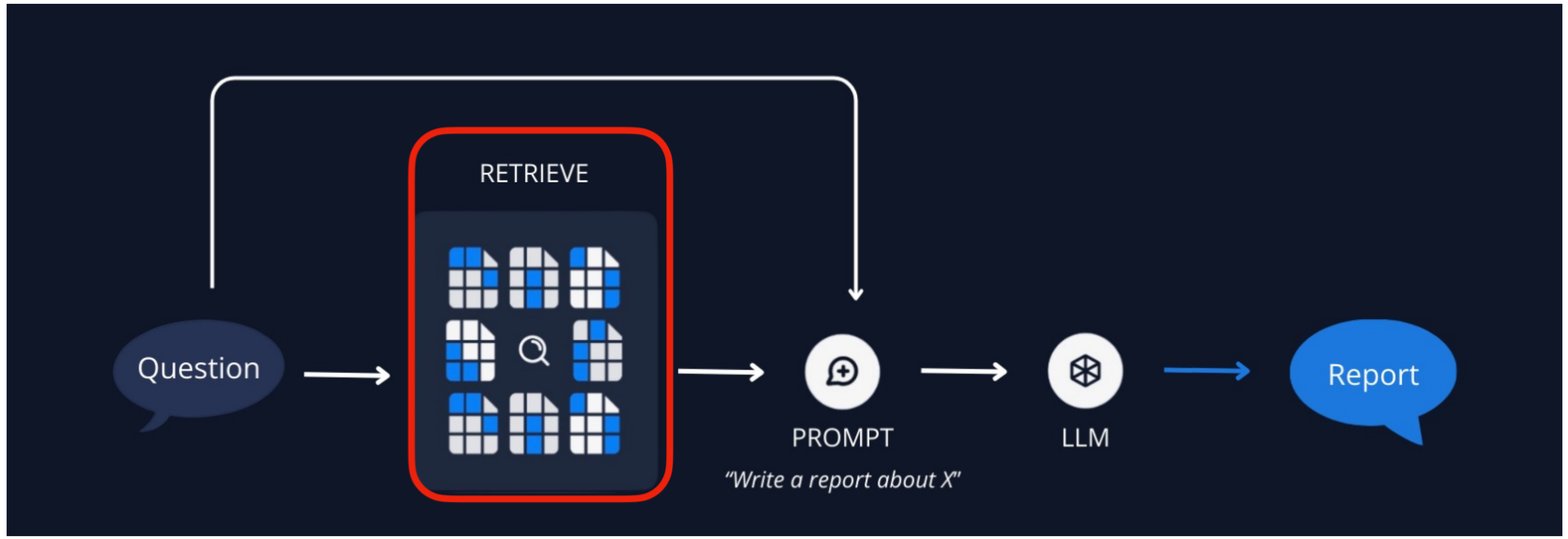
- A user has requested a report on the topic: inflation in Japan.

  As part of the report, we need to query a RAG pipeline to fetch relevant news.

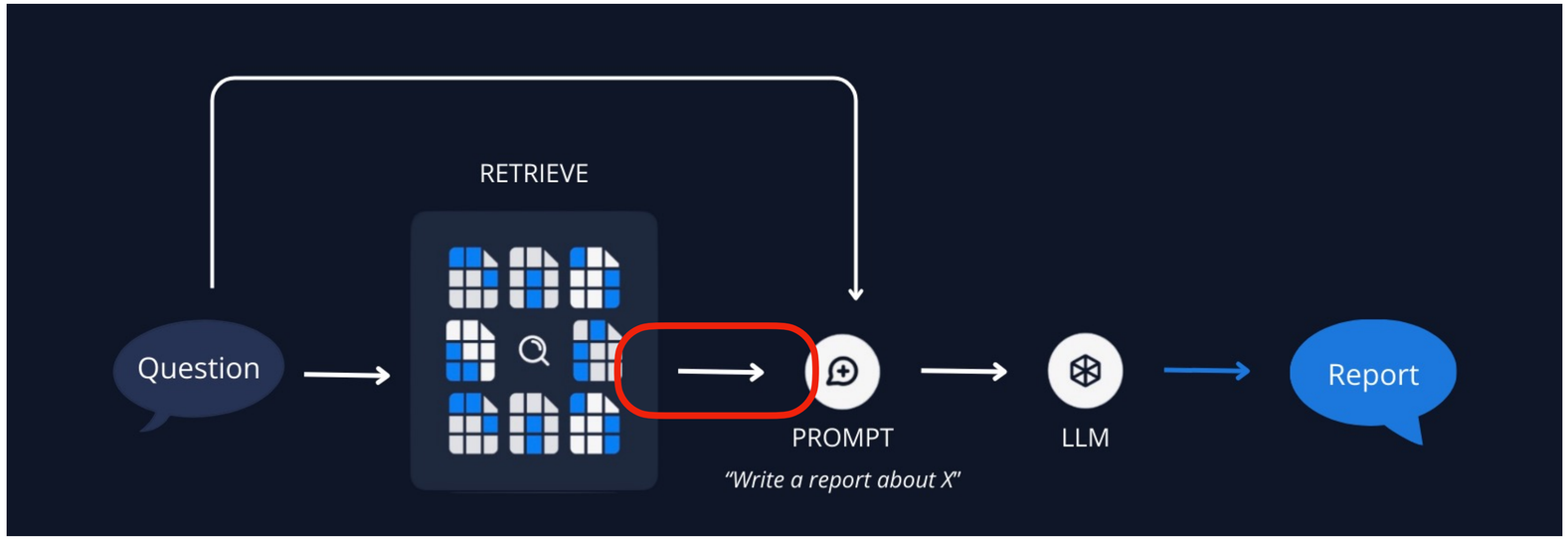
  Please output five queries that are optimized for semantic similarity search.

  These queries should help to fetch information for writing the report.

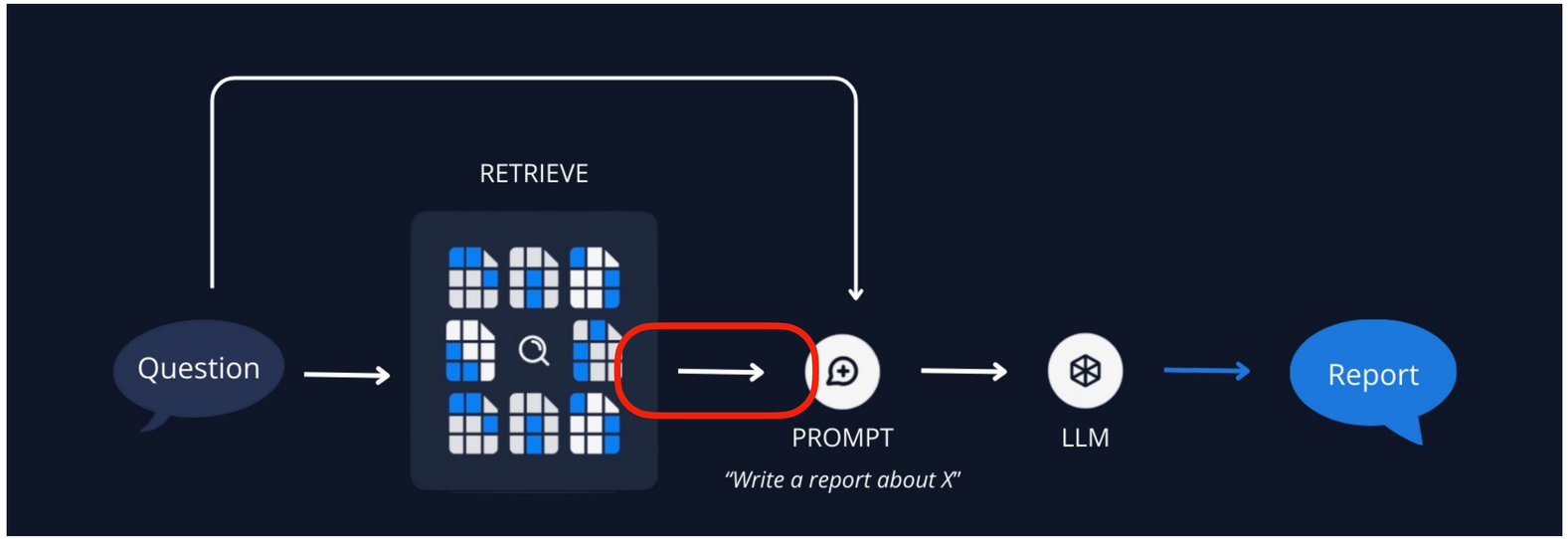
  Please output your answer as a JSON formatted list.



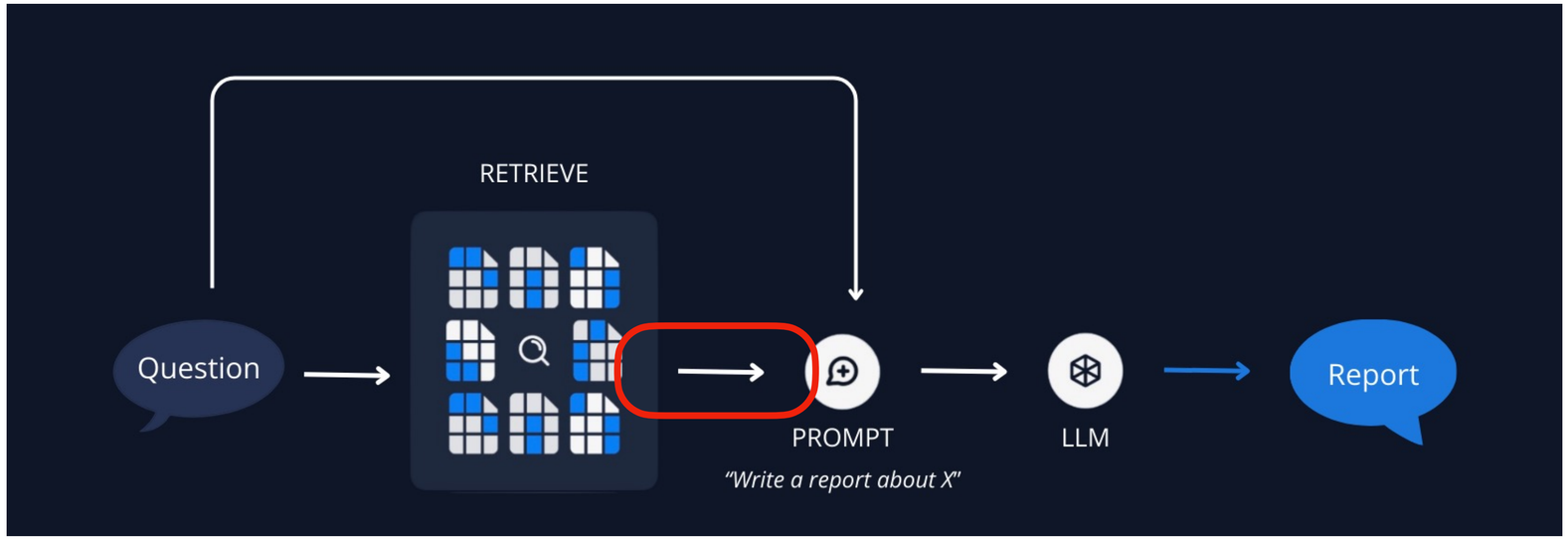

```
{
    "optimized_queries": [
        "Japan's current inflation rate and its impact on consumer prices and spending patterns",
        "Bank of Japan monetary policy decisions and their effects on inflation management",
        "Wage growth trends in Japan and their relationship with inflationary pressures",
        "Japanese supply chain disruptions and their contribution to price increases",
        "Historical comparison of Japan's inflation rates and economic recovery measures"
    ]
}
```



- So now we have a collection of articles from our 5 optimized LLM queries.
- Should we just pass the articles directly into the final report generation prompt?
- No.



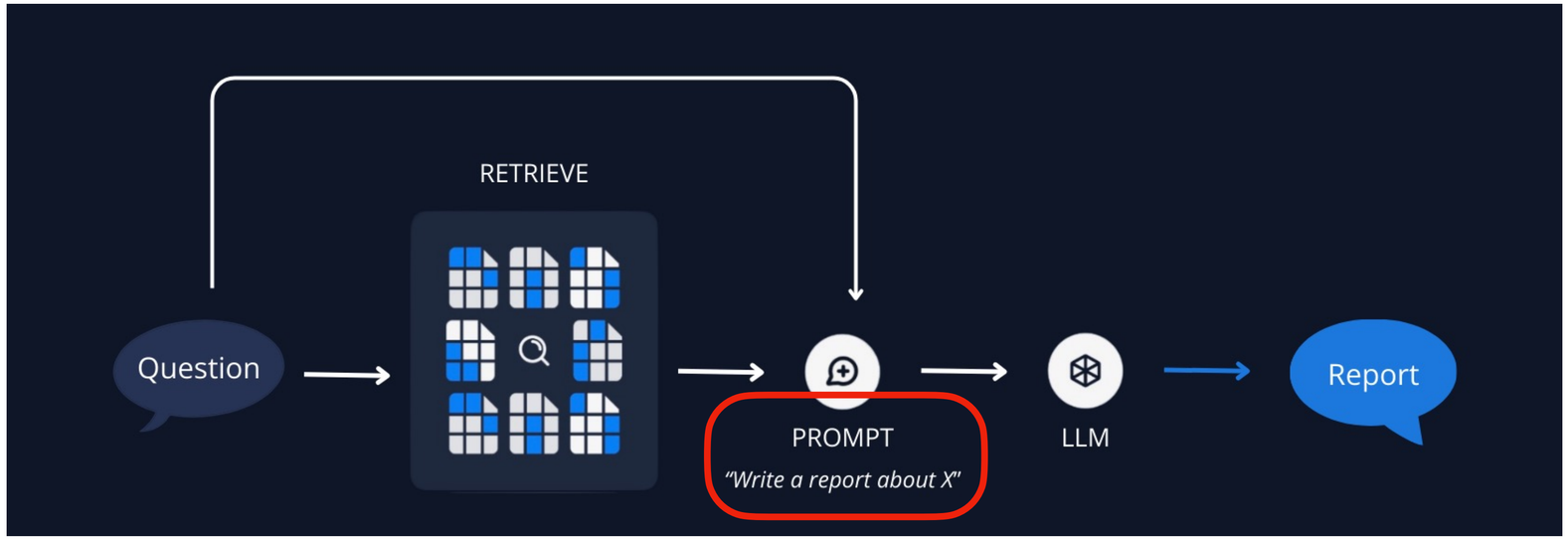
- Often, you will find the same article several times.
- Even with good semantic search, only 50% of the articles will be irrelevant.
- Solution: Ask an LLM to remove the duplicated articles, and ask another LLM to summarize the content.

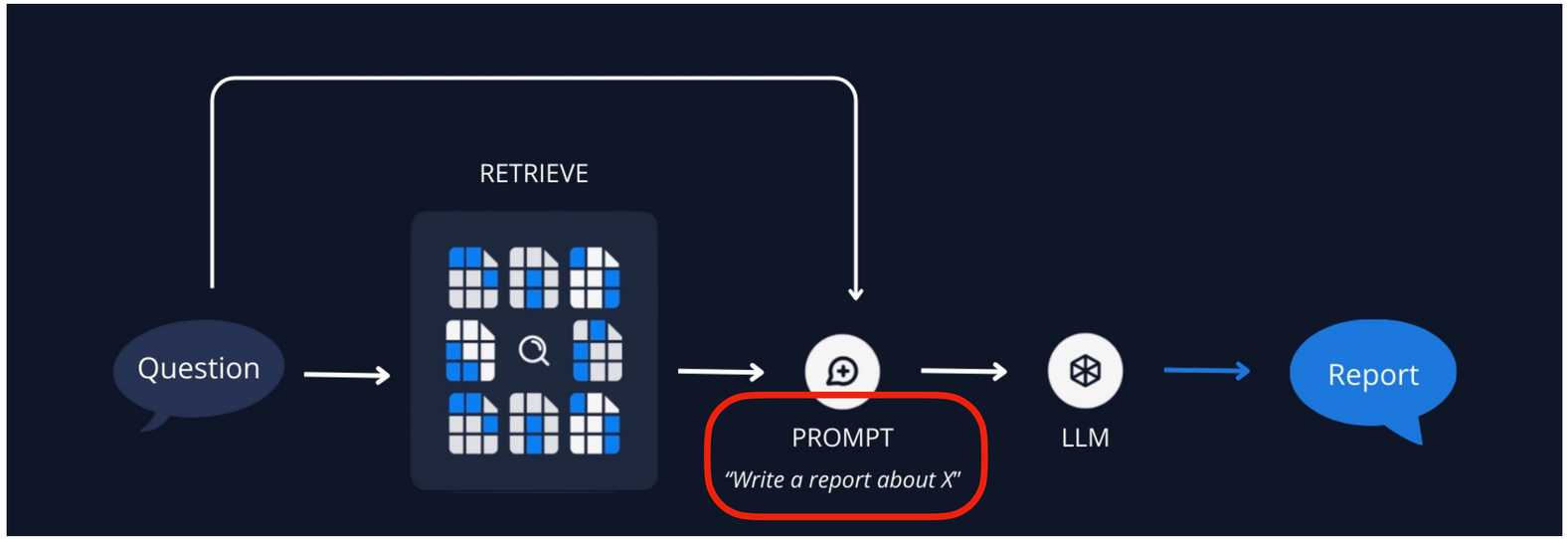


- Alternative solution: Reranking.
- ColBERT is a method that uses expensive bigger models to achieve a higher quality ranking.

  Do an initial RAG query, and then a COLBERT reranking.

- See bonus slides at the end of the deck for an explanation of how ColBERT works.

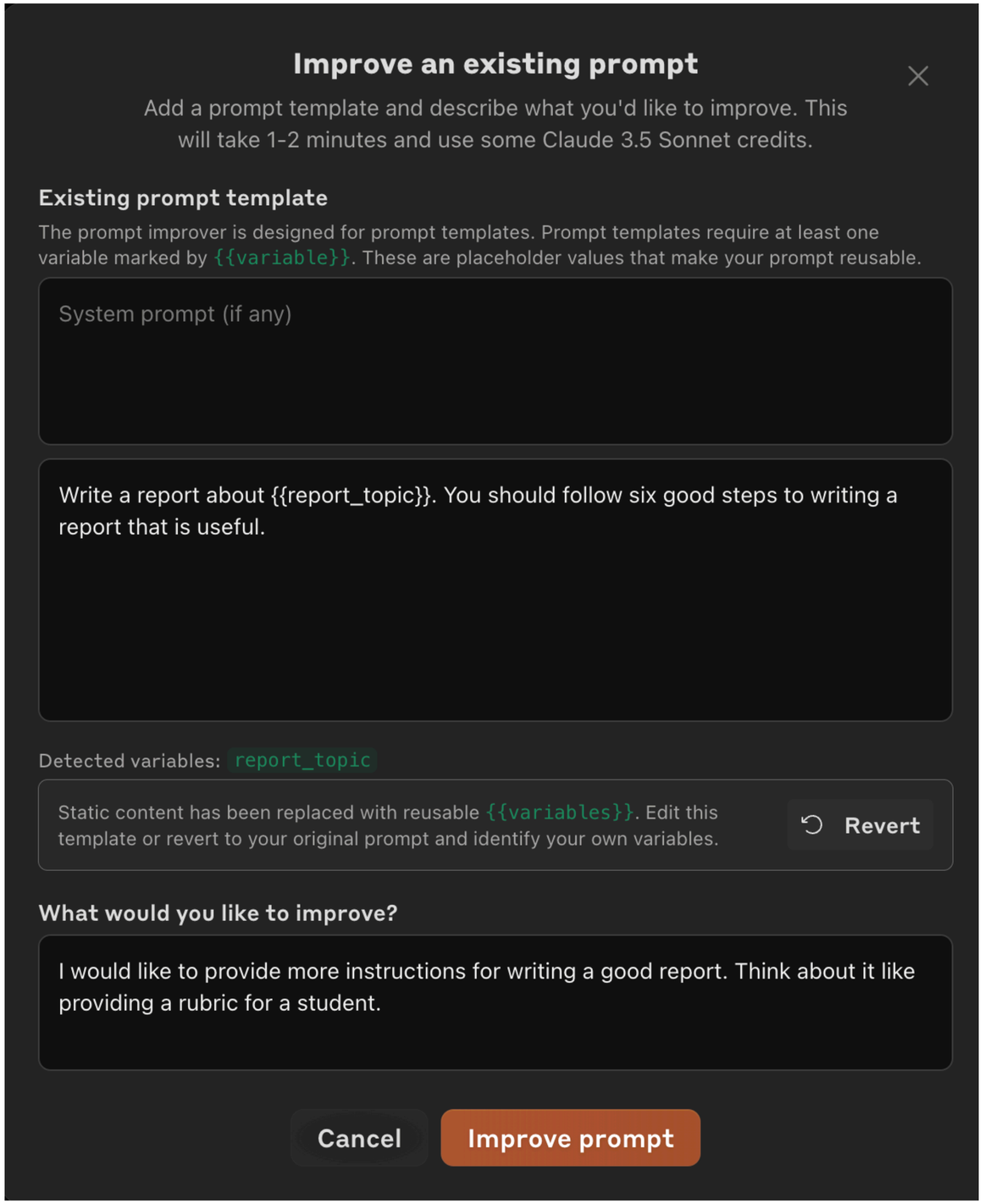

RETRIEVE
Question
PROMPT
"Write a report about X"
LLM
Report

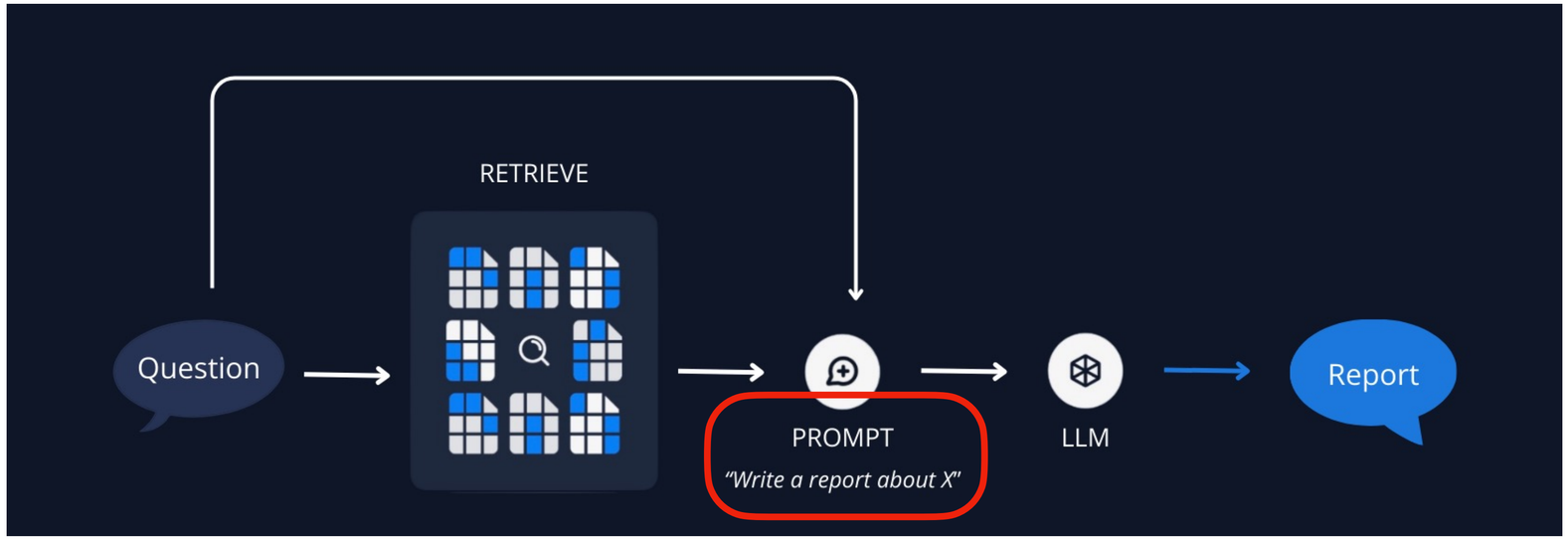


- Can we use a more detailed prompt?
- Sure.
- Giving the LLM instructions about how to write a good report is essential.

```
You are an expert analyst tasked with generating clear, actionable reports. Using recent news and data, create a report that follows these 6 steps:

1. First, provide CONTEXT:
What are the key events and trends everyone needs to know? What timeframe are we examining?

2. Then, present DATA POINTS:
What are the most important metrics? How do they compare to historical data? What stands out?

3. Move to ANALYSIS:
How do these data points connect? What's causing these trends? What assumptions should we question?

4. Explain IMPLICATIONS:
Who is affected? What are the short and long-term impacts? What scenarios should we consider?

5. Offer RECOMMENDATIONS:
What specific actions should be taken? In what order? What resources are needed?

6. End with a SUMMARY:
What are the 3 key takeaways? What deadlines matter? What decisions need to be made?

Format your response in clear, concise language. Support each point with specific examples from recent news or data. Focus on actionable insights.
```

- If you need help writing good prompts, there are tools called “prompt optimizers”
- They help take your prompts and transform them into focus, detailed prompts.





- We can also replace a single stage prompt with a more complex chain of prompts.

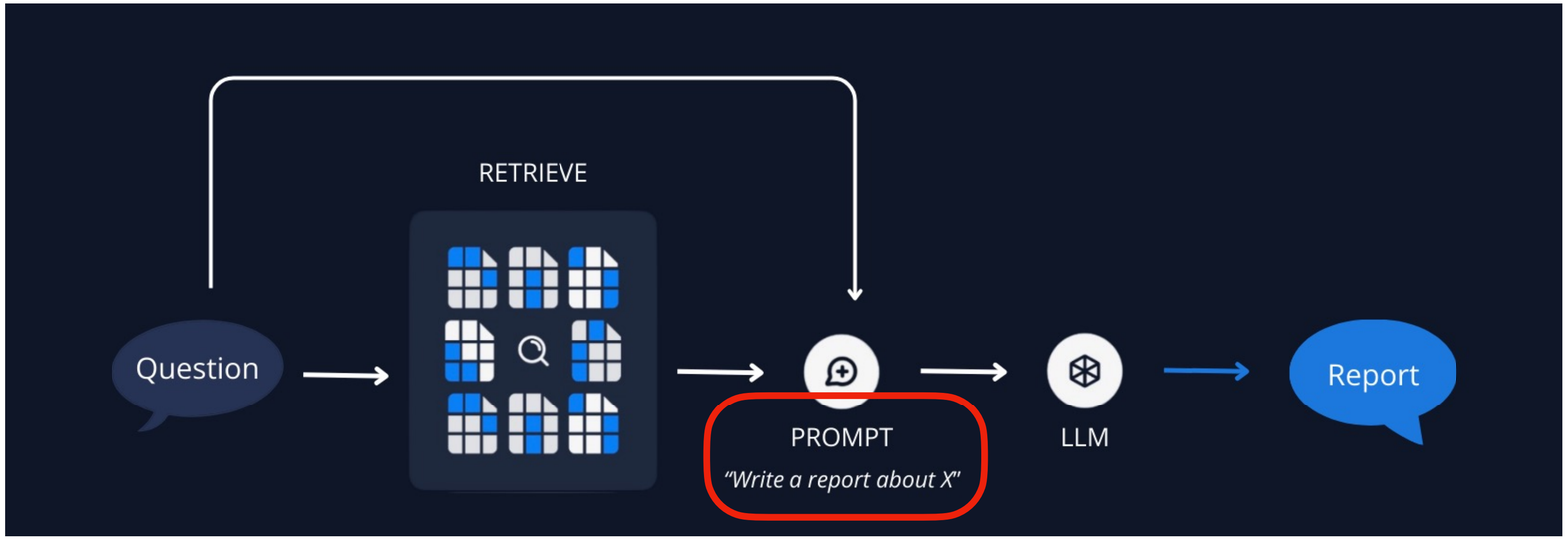
📝 REPORT GENERATION CHAIN

1. PLAN
   "Outline key sections, identify data sources, set scope"

2. PLAN REVIEW
   "Challenge assumptions, identify gaps, strengthen logic flow"

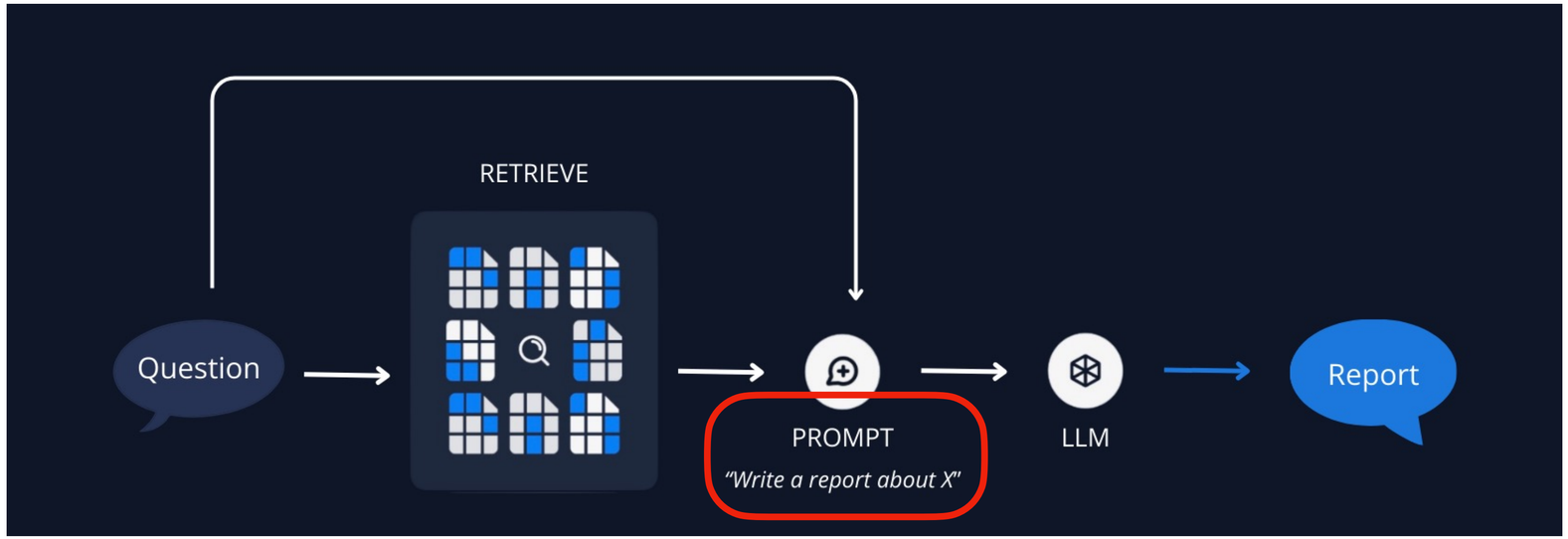
3. DRAFT
   "Write detailed analysis following 6-step framework"

4. FINAL POLISH
   "Tighten language, verify claims, ensure actionability"



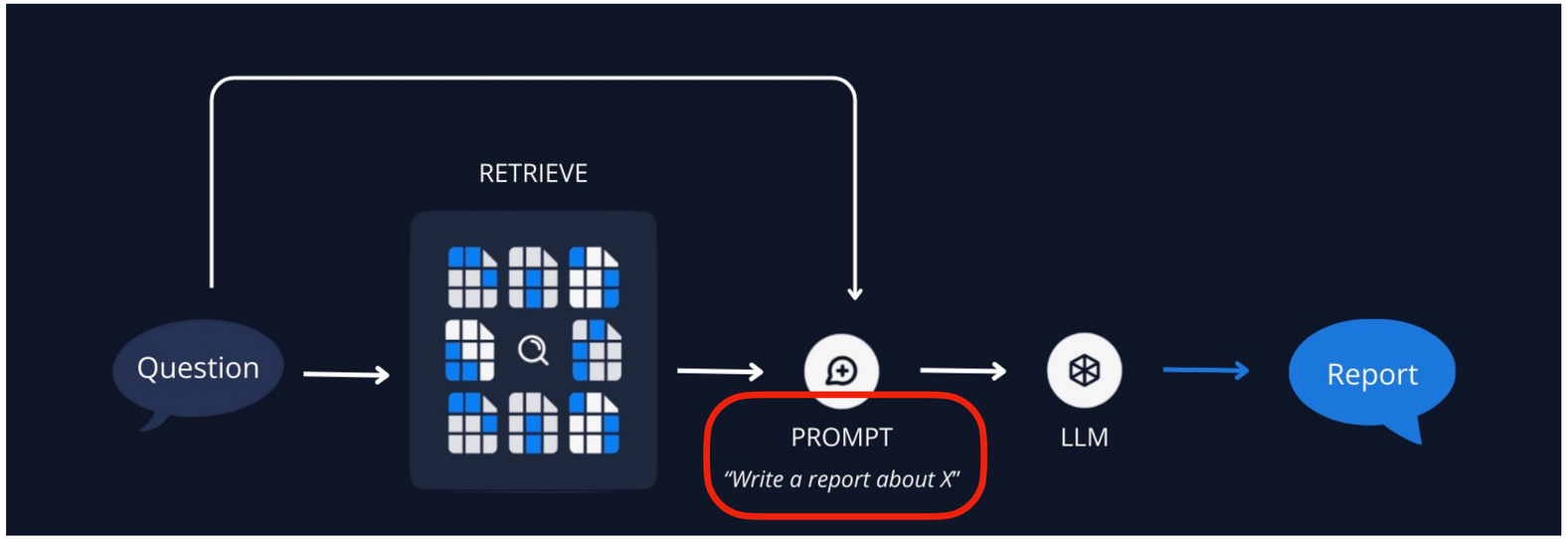
First plan your report -> revise your planned report -> draft your report -> revise into a final draft

- This is intellectually satisfying, but I have yet to see it help very much.



- There is a major difficulty with prompt optimization.
- Changing prompts is very easy.
- How do I know that modifying my prompt is actually helping?

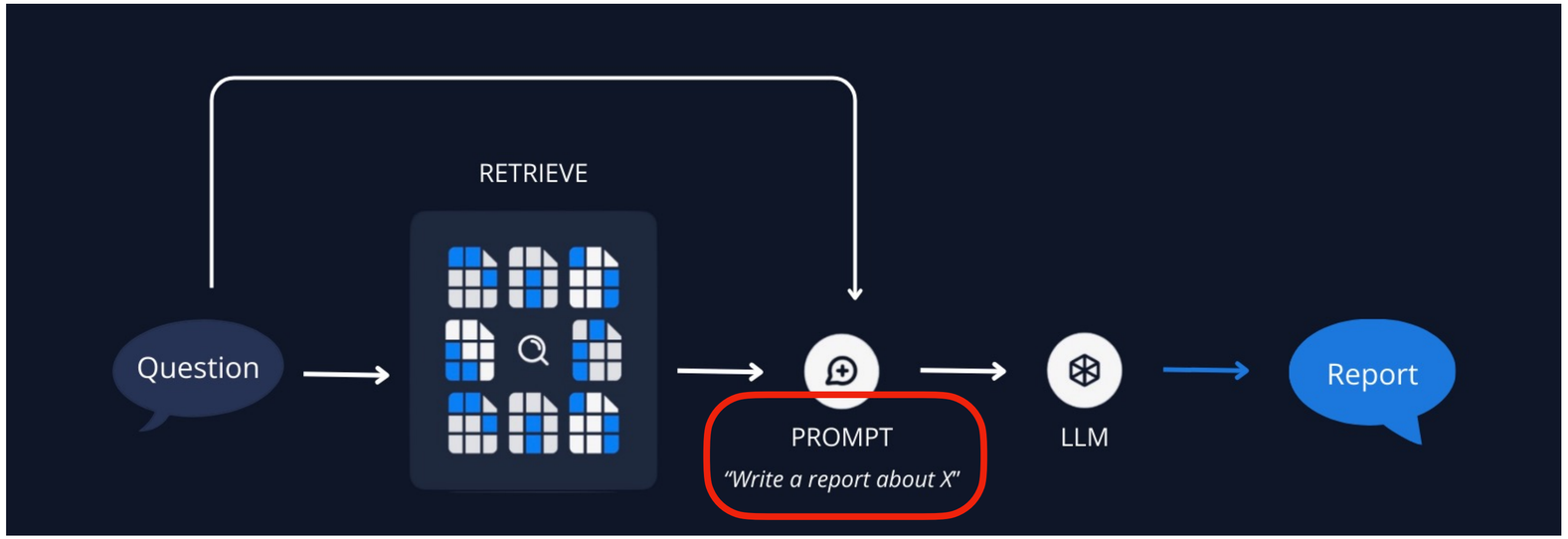
Advanced Report Generation



- There is no easy answer here.

- Generally, you have two options.

Advanced Report Generation

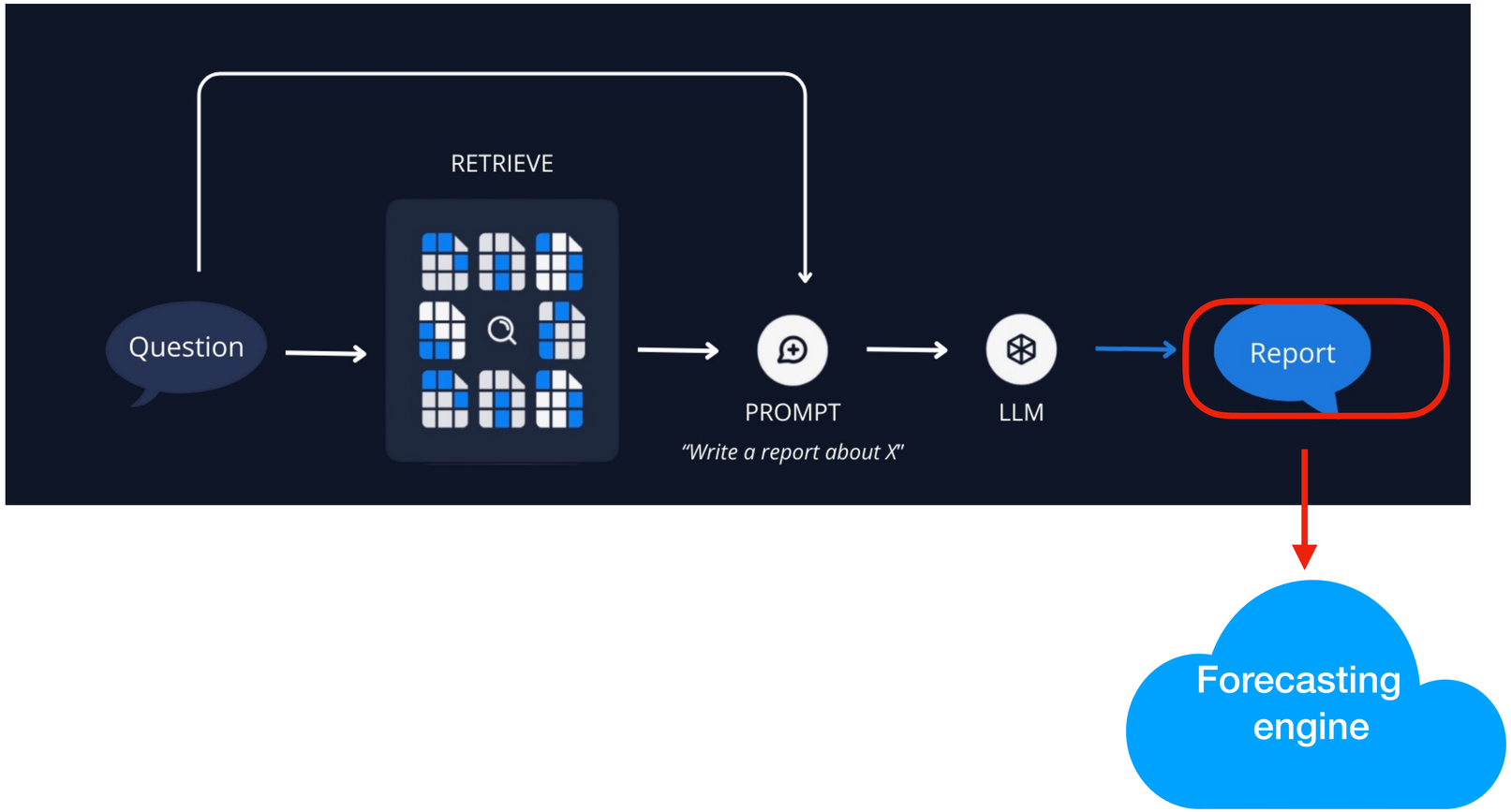


- Option 1: Use the reports on some downstream problem you care about.

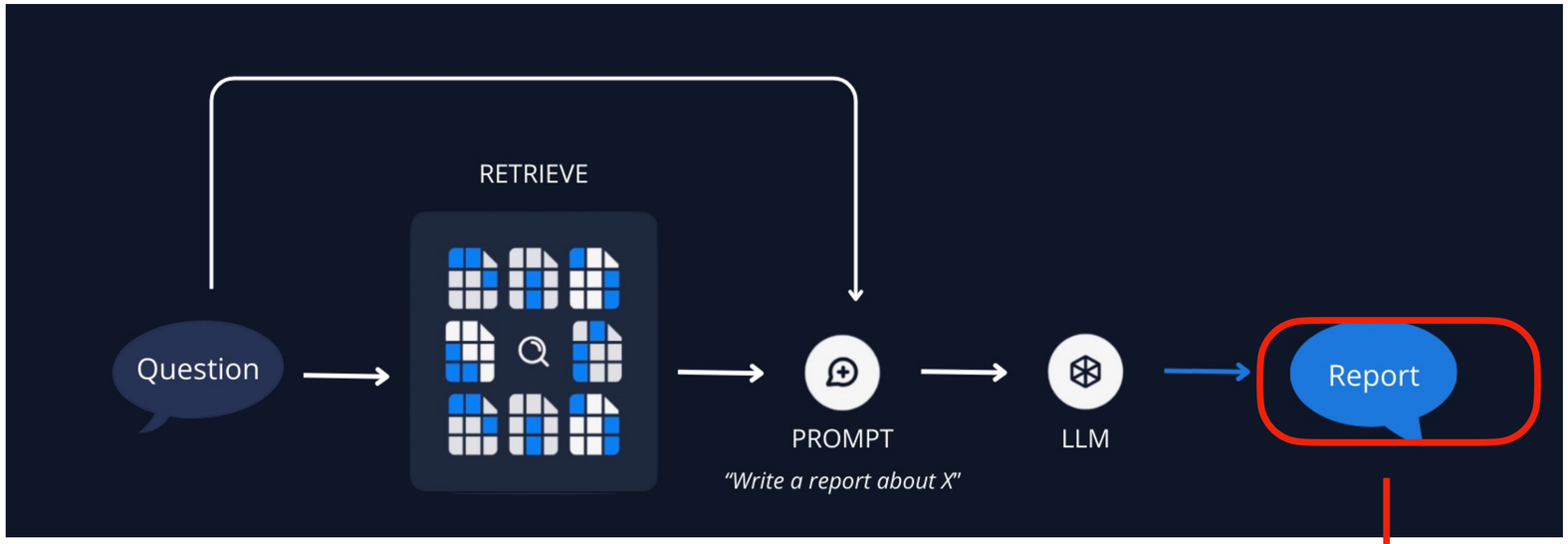
  For example, use the report to help with forecasting.

- Make sure that all prompt changes are helping with your downstream metric.

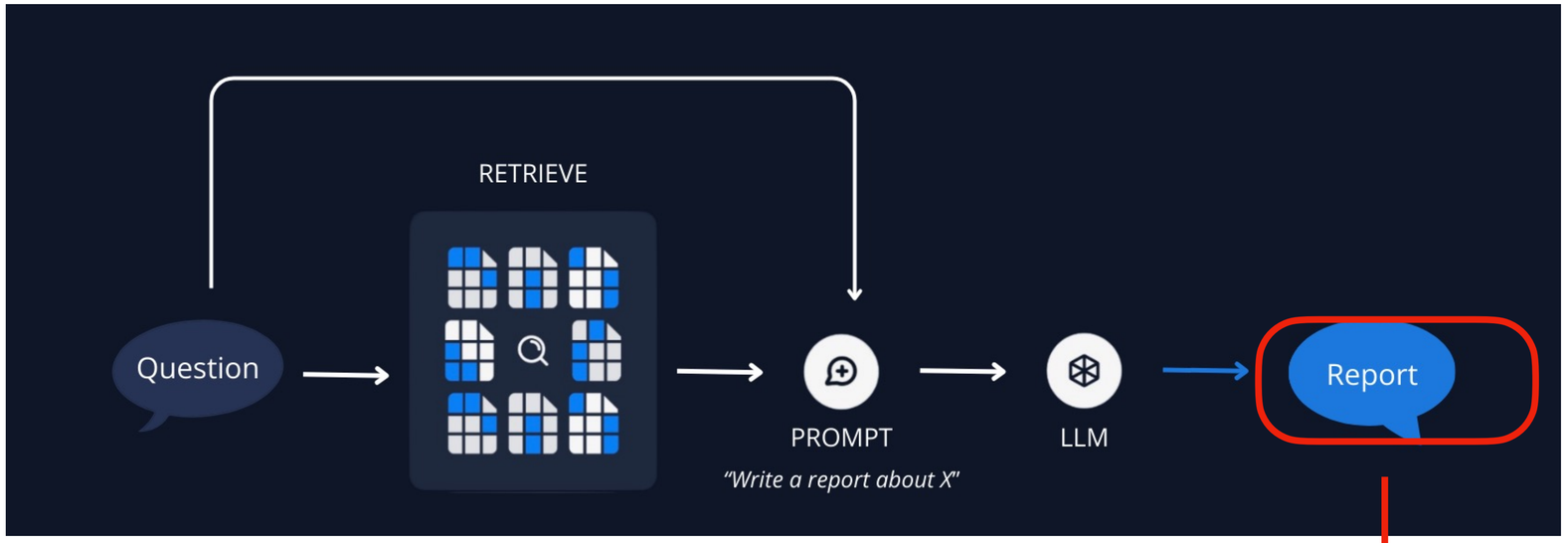
Advanced Report Generation



Advanced Report Generation




```
🔄 FULL PROMPT:
"You are a forecasting expert. Using {REPORT} as your foundation, generate probabilistic forecasts that:
1. Identify key uncertainties in {REPORT}'s data
2. Assign confidence intervals to each prediction
3. Explain which parts of {REPORT} support or challenge your estimates
4. Consider multiple scenarios based on {REPORT} evidence
5. Highlight where {REPORT}'s data might be insufficient
6. Update predictions if new information conflicts with {REPORT} findings

Express all forecasts as probability ranges and explain your reasoning."
```



Forecasting engine

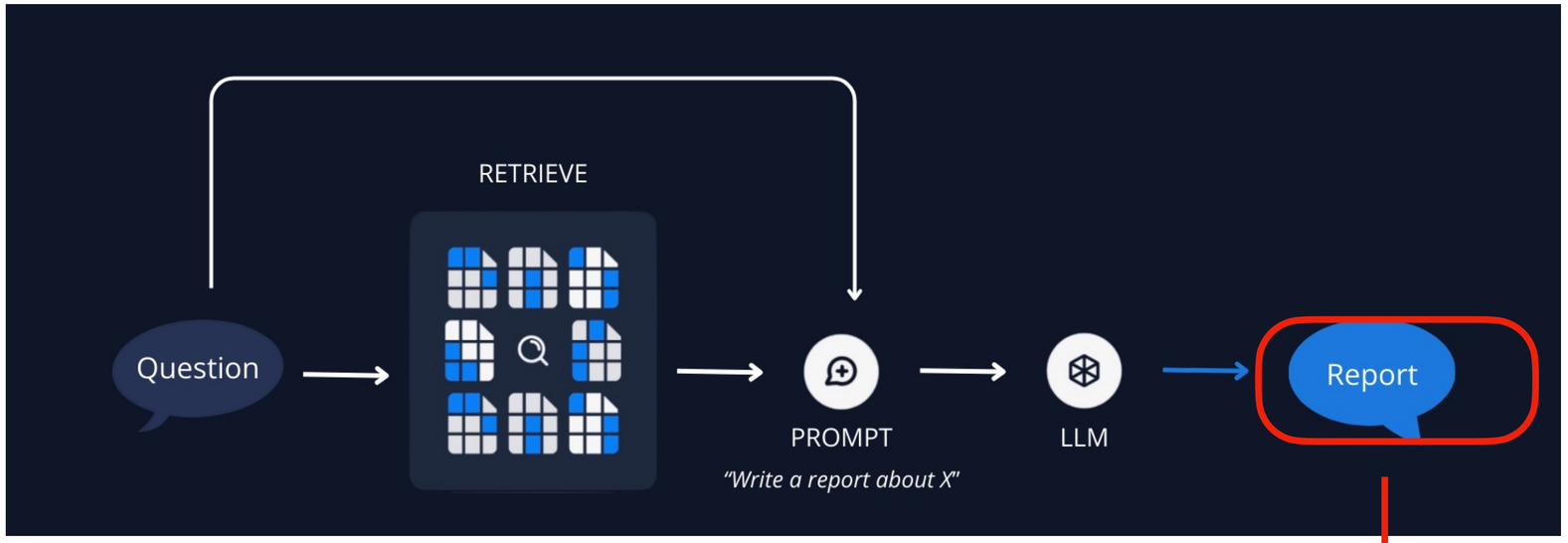
Will Nvidia stock price increase in 2024?

Yes

p=0.9

No

p=0.1



Brier score

Forecasting engine

Will Nvidia stock price increase in 2024?

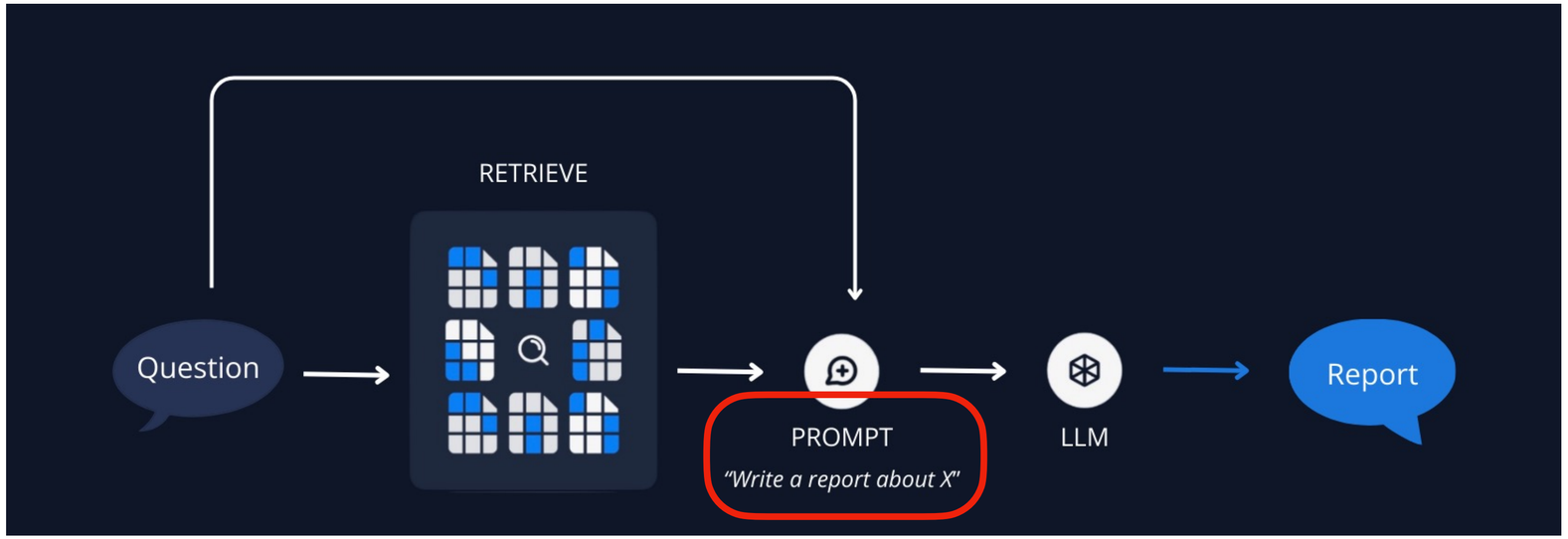
Yes

p=0.9

No

p=0.1

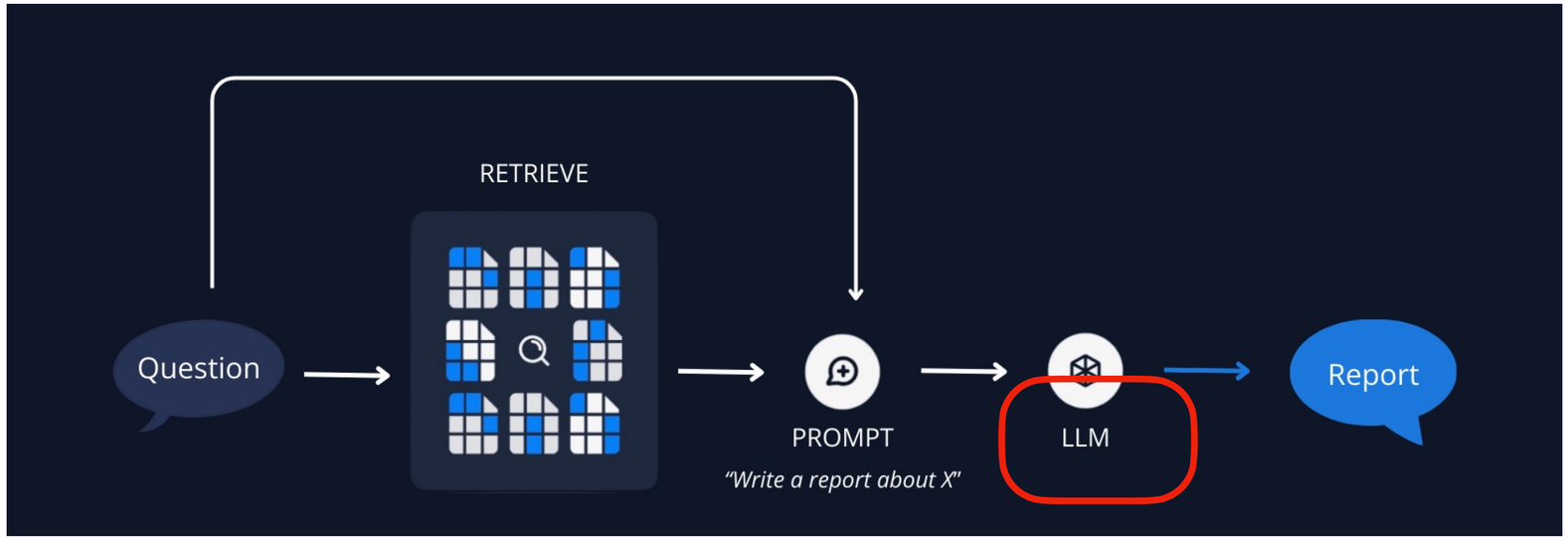
Advanced Report Generation

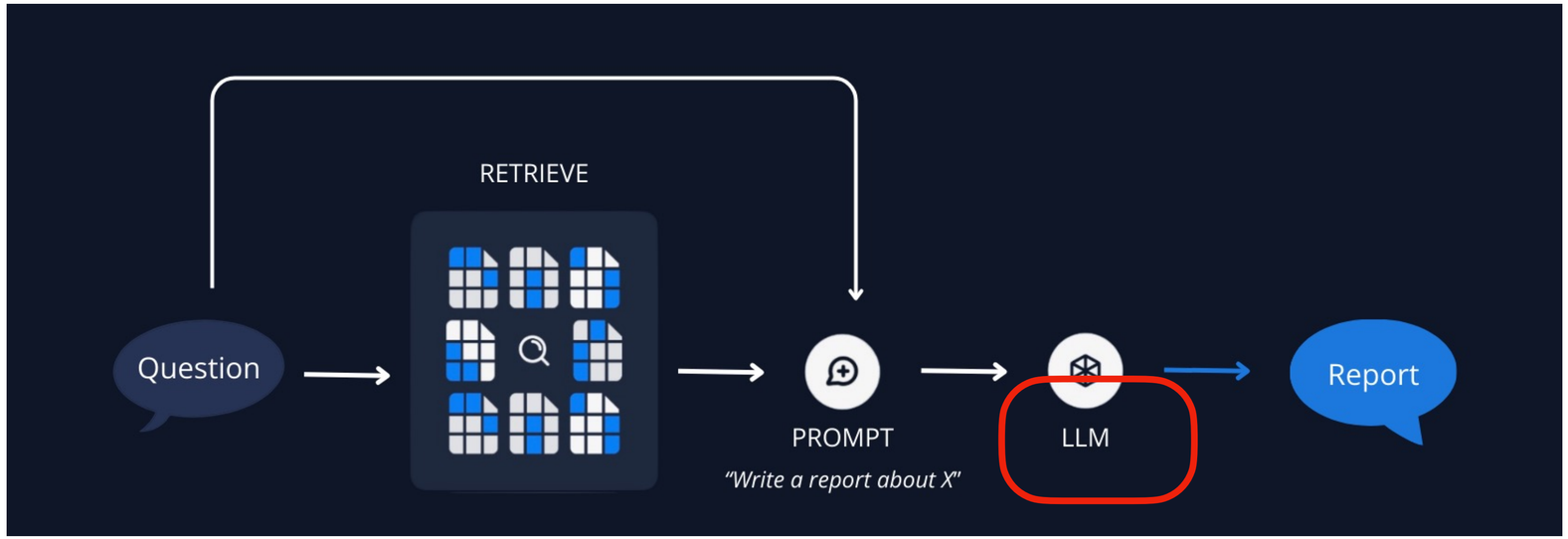


- Option 2 for report evaluation: Give the system an example of “gold standard” reports.
- Use LLMs to grade the report generated with your prompt and the gold standard reports.
- Only accept prompt changes if they bring your generated reports closer to “gold standard”

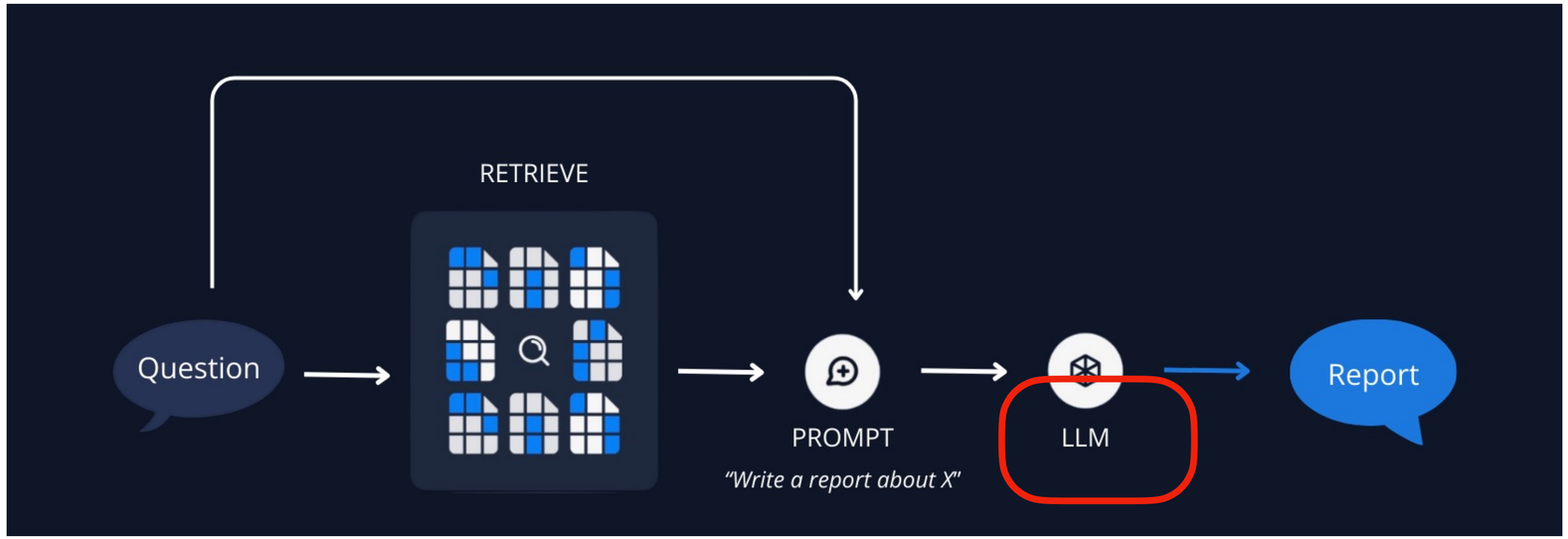
Advanced Report Generation

```
🎯 DIFFERENTIAL ANALYSIS PROMPT:
"Compare reports {A} and {B}:

1. DELTA IDENTIFICATION
List all substantive differences between {A} and {B}

2. IMPACT ASSESSMENT
For each delta:
- What information is gained/lost?
- How does this affect conclusions?
- Which version better serves the report's purpose?

Base all judgments on {B} as gold standard."
```


RETRIEVE
Question
PROMPT
"Write a report about X"
LLM
Report

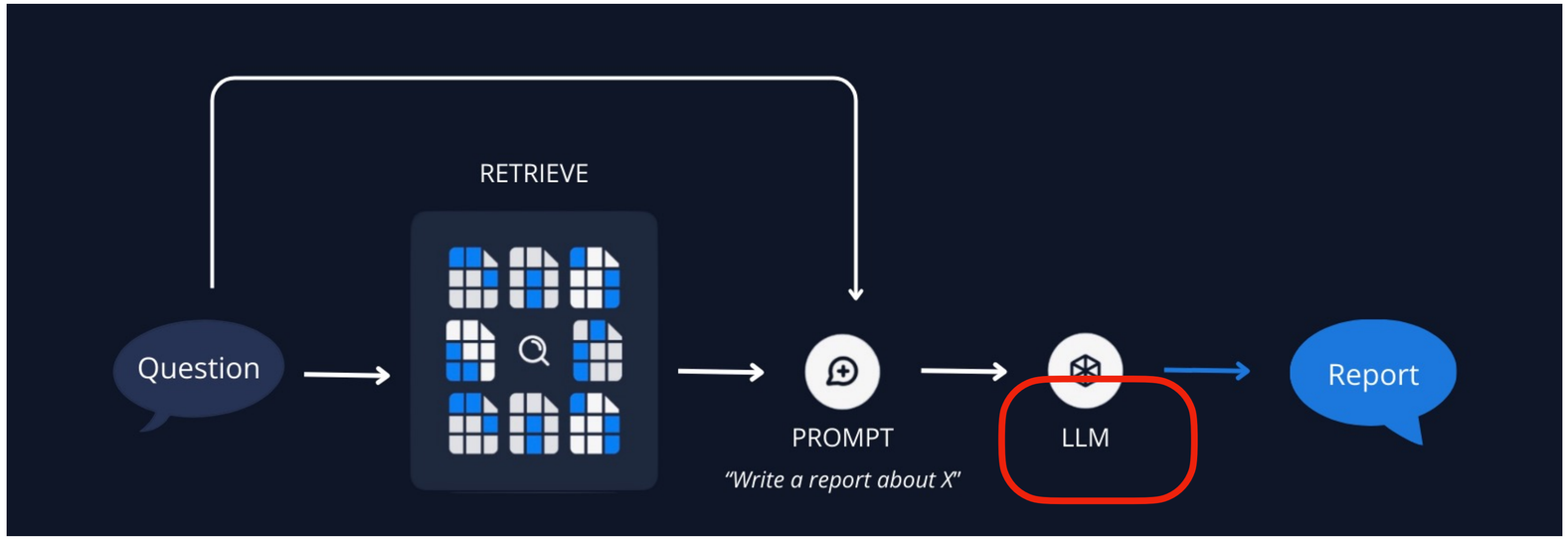


- What LLM should I use?
- Generally, GPT-4o is good.
- The writing style of Claude is a little better now, on January 5, 2025.
- I have found both models generate reports that yield comparable forecasts.

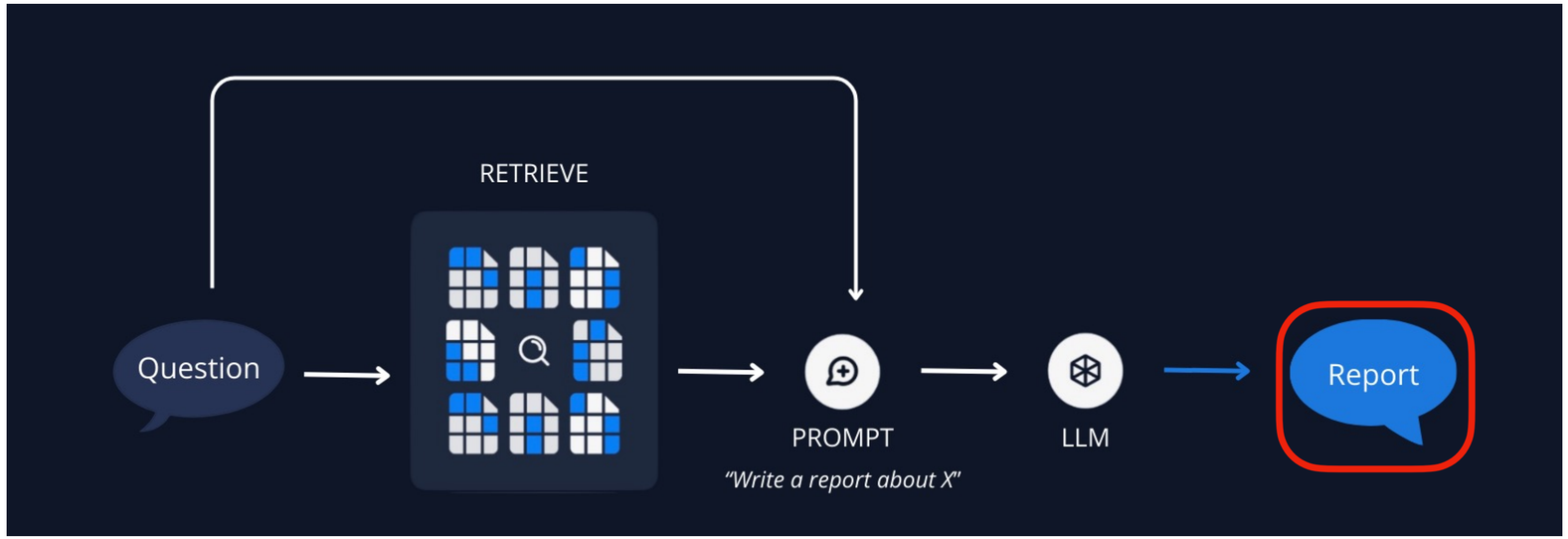


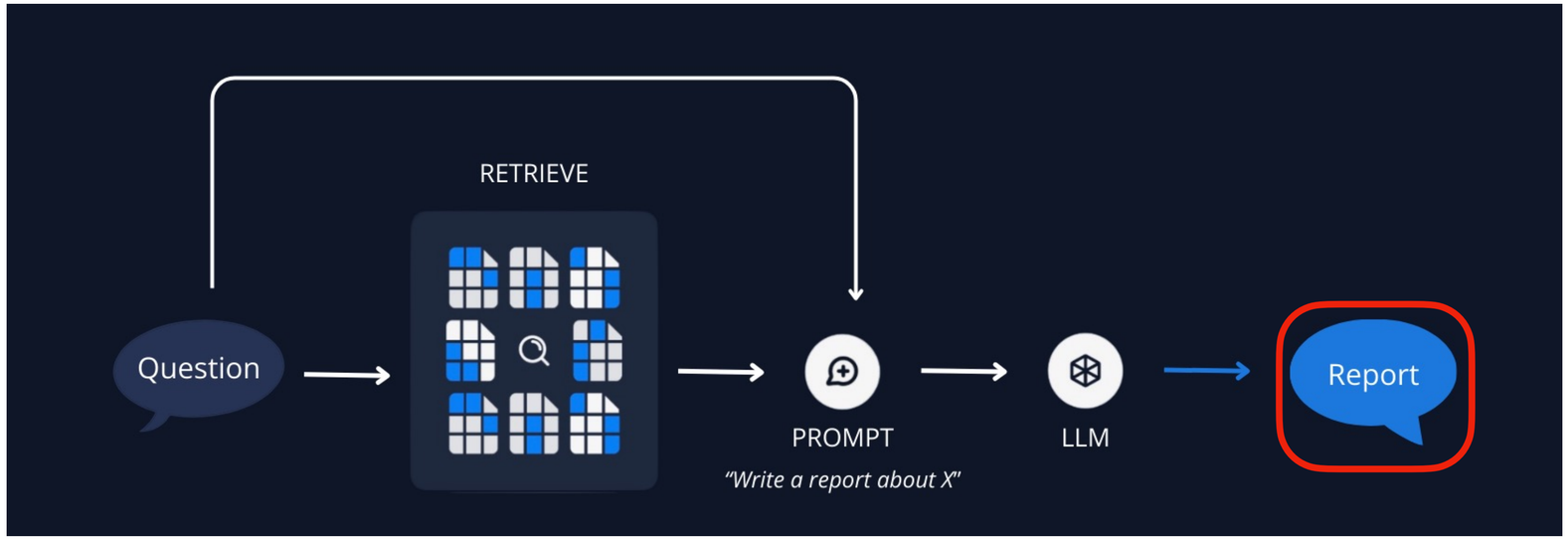
- Should you use GPT-o1 or GPT-o3?
- Probably not.
- Those models are designed for complex reasoning and multi-step problem solving.

  This is great for math problems.
- Report synthesis is often negatively impacted from this. It leads to “overthinking”

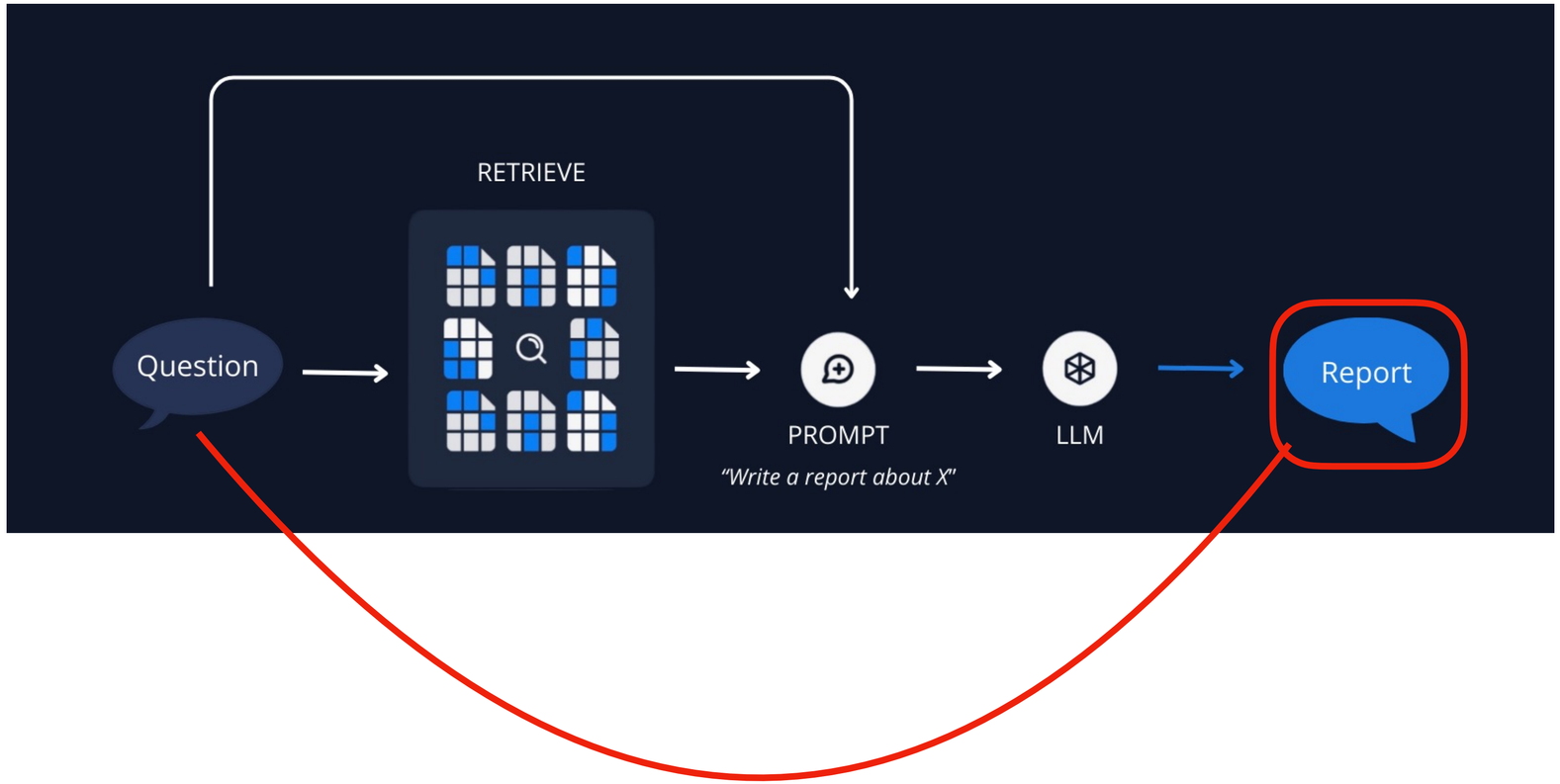


- However, you should feel free to try out GPT-o1 if you want.
- I would just advise that you have some clear metric of tracking if the o1 models are actually improving your outcome.

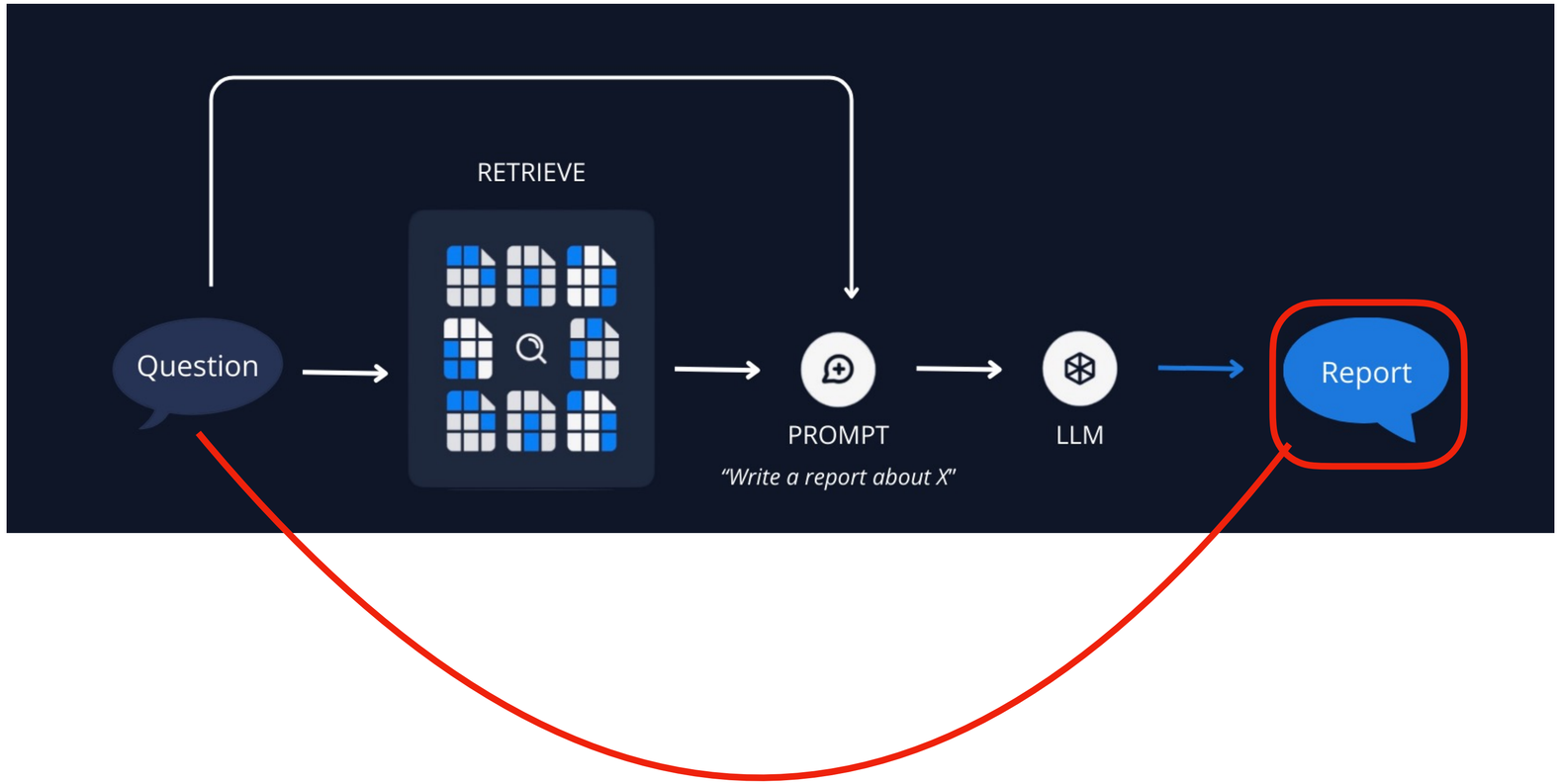

RETRIEVE
Question
PROMPT
"Write a report about X"
LLM
Report



- We have a final report. What now?
- Usually, the final report is good enough.
- You can feed it into any downstream task you care about.
- However, if you want, you can also try iterative improvement techniques.



- Feed the generated report back into an LLM.
- Ask it to come up with some research questions that would improve the report.
- Run the whole pipeline again, this time iterating on the first report.



- This is sometimes called iterative RAG or multi-stage RAG.
- Does it help?
- The report will usually get more detailed. But not better.

🌟 IDEAL SCENARIO: Japan Inflation Report

Initial Report: "Japan's inflation hit 2.8% in 2023"

LLM's High-Value Follow-up:
"Given Japan's aging population and recent policy shift from decades of deflation, how might this inflation impact pension funds' investment strategies?"

Why this is ideal:
- Connects multiple complex factors
- Identifies downstream effects
- Surfaces non-obvious stakeholders
- Leads to actionable insights

🌟 IDEAL SCENARIO: Japan Inflation Report

Initial Report: "Japan's inflation hit 2.8% in 2023"

LLM's High-Value Follow-up:
"Given Japan's aging population and recent policy shift from decades of deflation, how might this inflation impact pension funds' investment strategies?"

Why this is ideal:
- Connects multiple complex factors
- Identifies downstream effects
- Surfaces non-obvious stakeholders
- Leads to actionable insights

📉 TYPICAL SCENARIO: Japan Inflation Report

Initial Report: "Japan's inflation hit 2.8% in 2023"

LLM's Basic Follow-up:
"Can you please provide more information about this inflation rate?"

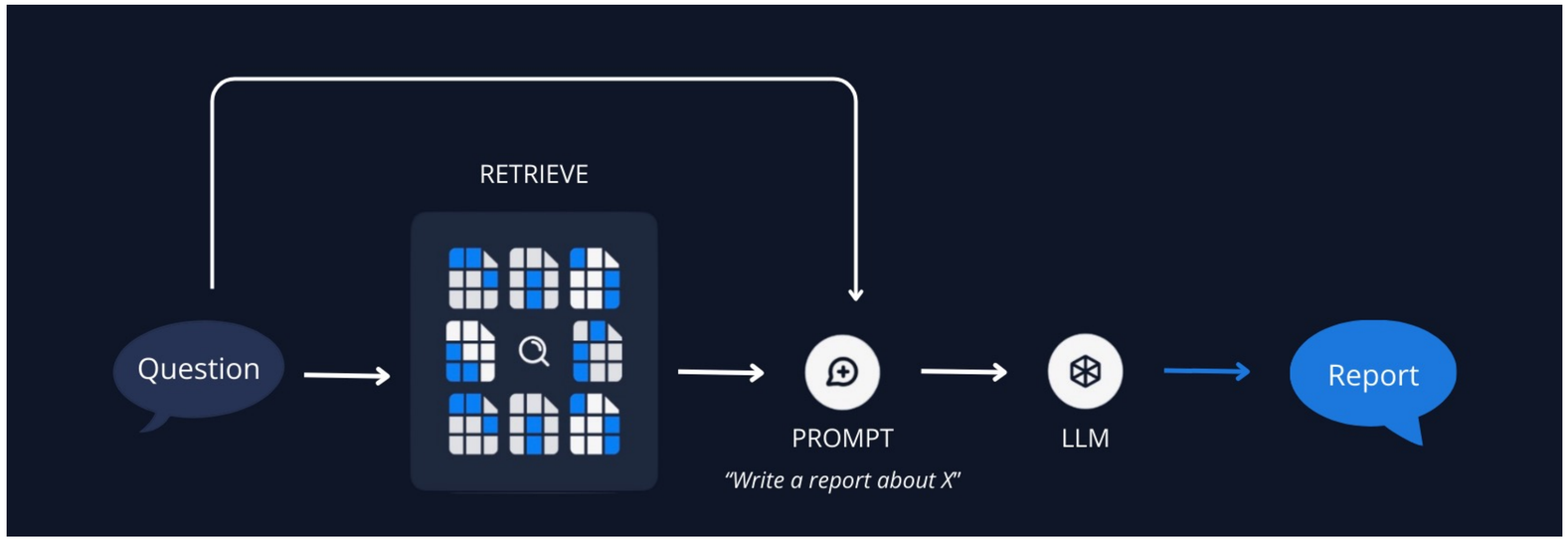
Why this is suboptimal:
- Vague and unfocused
- Doesn't specify what information is needed

Conclusion

Conclusion

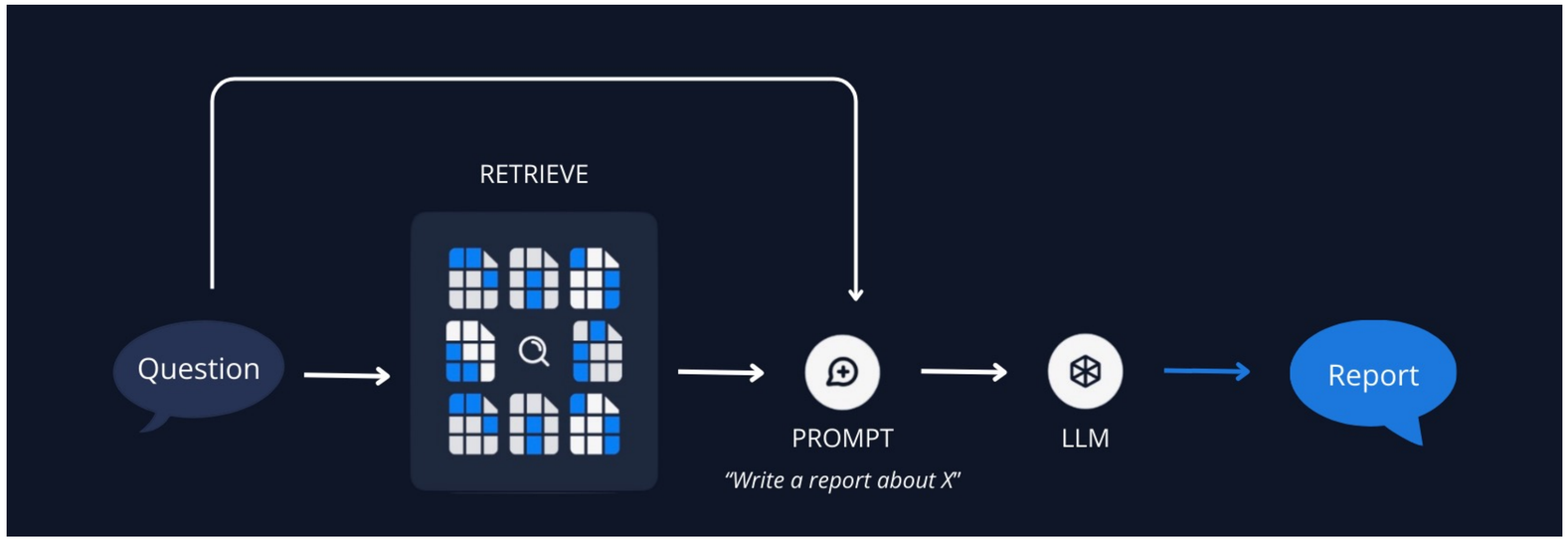
- What is the most important part of the report generation pipeline?
- Providing access to good high quality data for retrieval.
- Many things can go wrong during retrieval.

  For example, good data doesn’t exist. Or good data does exist but you fail to find it.



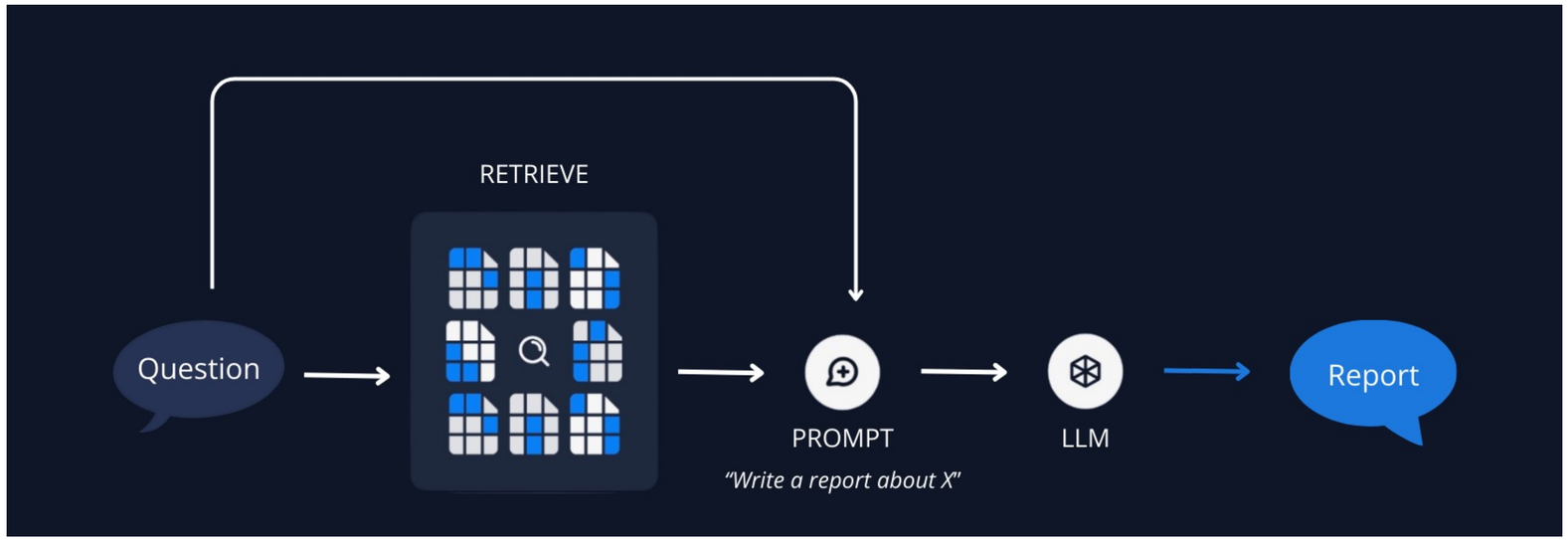
Conclusion

- You should always set up an evaluation pipeline to test the robustness of data retrieval.
- The reports will reflect the quality of the input data.



Conclusion

- A key lesson from this lecture
- It’s essential to have a method of scoring reports.
- Many pieces of the report generation pipeline can be changed. You need to track if those changes are effective.



The End

Bonus Slides

How ColBERT works

How ColBERT works

```
# Simplified representation
query_tokens = ["inflation", "in", "japan"]
doc_tokens   = ["japanese", "prices", "increased", "significantly"]

# Each token is actually a vector (~32 dimensions)
```

How ColBERT works

```
# Simplified representation
query_tokens = ["inflation", "in", "japan"]
doc_tokens   = ["japanese", "prices", "increased", "significantly"]

# Each token is actually a vector (~32 dimensions)
```

```
# In reality these are 32-dim vectors, but let's use 4-dim for illustration
token_vectors = {
    "japan": [0.8, -0.2, 0.5, 0.1],
    "japanese": [0.7, -0.3, 0.6, 0.2],
    "inflation": [0.4, 0.8, -0.3, 0.1],
    "prices": [0.3, 0.7, -0.2, 0.0]
}
```

How ColBERT works

```
# Simplified representation
query_tokens = ["inflation", "in", "japan"]
doc_tokens   = ["japanese", "prices", "increased", "significantly"]

# Each token is actually a vector (~32 dimensions)
# Let's say we compute similarity scores between each pair:

similarity_matrix = [
    #japanese  prices  increased  significantly
    [0.8,      0.3,    0.2,       0.1],    # inflation
    [0.1,      0.1,    0.1,       0.1],    # in
    [0.9,      0.2,    0.1,       0.1]     # japan
]
```

How ColBERT works

```
# In reality these are 32-dim vectors, but let's use 4-dim for illustration
token_vectors = {
    "japan": [0.8, -0.2, 0.5, 0.1],
    "japanese": [0.7, -0.3, 0.6, 0.2],
    "inflation": [0.4, 0.8, -0.3, 0.1],
    "prices": [0.3, 0.7, -0.2, 0.0]
}

# When we say "compute similarity", we're actually doing:
# dot product between these vectors (or cosine similarity)
# japan · japanese = (0.8 × 0.7) + (-0.2 × -0.3) + (0.5 × 0.6) + (0.1 × 0.2) = 0.9
```

```
similarity_matrix = [
    #japanese  prices  increased  significantly
    [0.8,      0.3,    0.2,       0.1],    # inflation
    [0.1,      0.1,    0.1,       0.1],    # in
    [0.9,      0.2,    0.1,       0.1]     # japan
]
```

The MaxSim Operation:

1. For each query token, find its BEST match in the document

- `inflation` → max(0.8, 0.3, 0.2, 0.1) = 0.8
- `in` → max(0.1, 0.1, 0.1, 0.1) = 0.1
- `japan` → max(0.9, 0.2, 0.1, 0.1) = 0.9

2. Sum these maximum similarities:

- Final Score = 0.8 + 0.1 + 0.9 = 1.8

```
similarity_matrix = [
    #japanese  prices  increased  significantly
    [0.8,      0.3,    0.2,       0.1],    # inflation
    [0.1,      0.1,    0.1,       0.1],    # in
    [0.9,      0.2,    0.1,       0.1]     # japan
]
```

Why This Works:

Why This Works:

- Each query term finds its most relevant match

Why This Works:

- Each query term finds its most relevant match
- Doesn't require exact matches ("japanese" matches well with "japan")

Why This Works:

- Each query term finds its most relevant match
- Doesn't require exact matches ("japanese" matches well with "japan")
- Preserves fine-grained semantic relationships

Why This Works:

- Each query term finds its most relevant match
- Doesn't require exact matches ("japanese" matches well with "japan")
- Preserves fine-grained semantic relationships
- More robust than averaging or single-vector approaches

Why This Works:

- Each query term finds its most relevant match
- Doesn't require exact matches ("japanese" matches well with "japan")
- Preserves fine-grained semantic relationships
- More robust than averaging or single-vector approaches

This is why ColBERT can catch subtle relevance that might be missed by simpler embedding approaches.

RAG

- Talk about Eres and RAGAS and different evaluation systems. Have a whole section dedicated to how we can evaluate these systems.