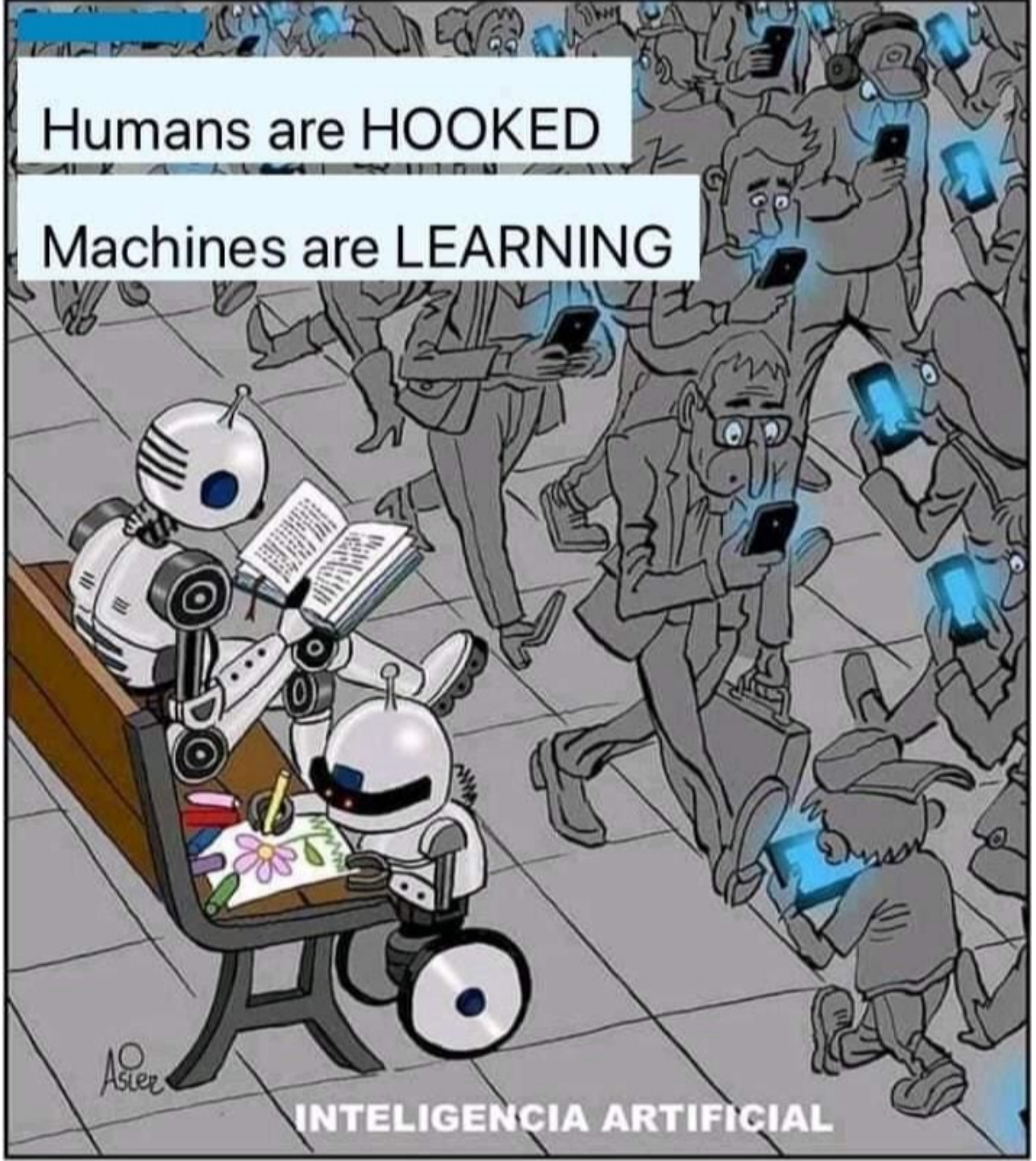
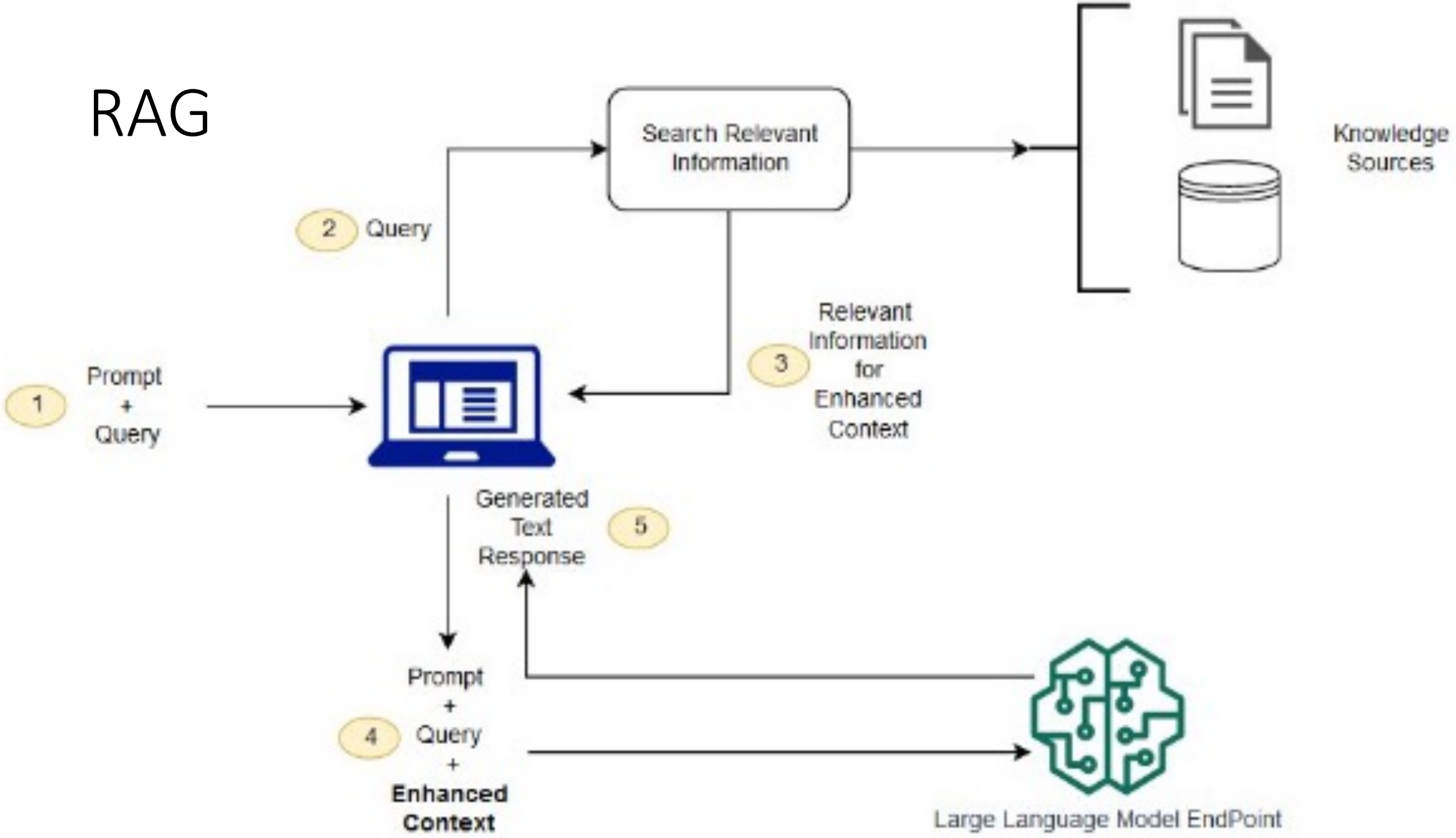
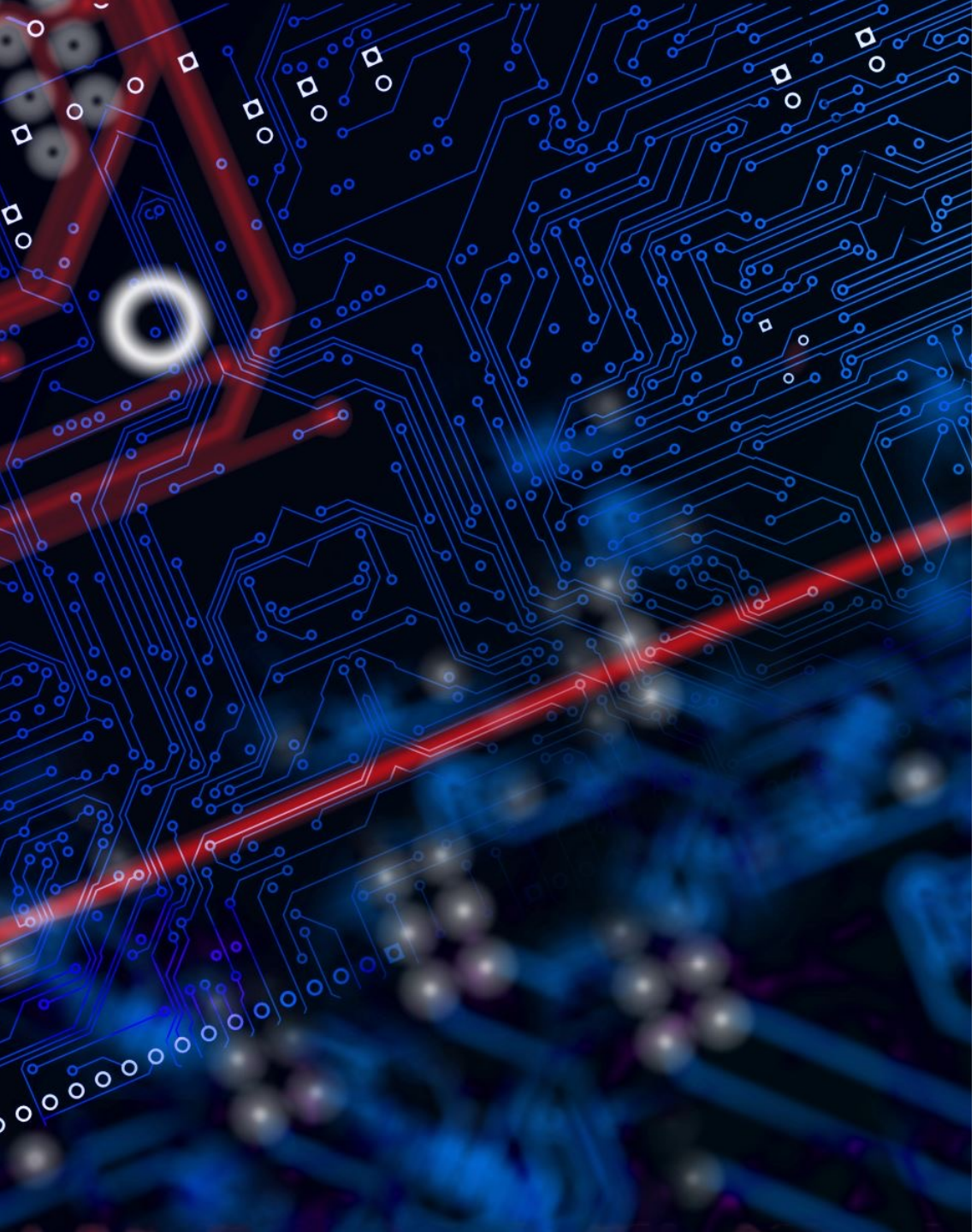


GenAI - Enterprise RAG

Naveen Kumar Bhansali
Co-Founder BlitzAI | Adjunct Faculty @ IIM Bangalore

RAG


Search Relevant Information
Knowledge Sources
2 Query
1 Prompt + Query
3 Relevant Information for Enhanced Context
Generated Text Response
5
Prompt + 4 Query + Enhanced Context
Large Language Model EndPoint



RAG

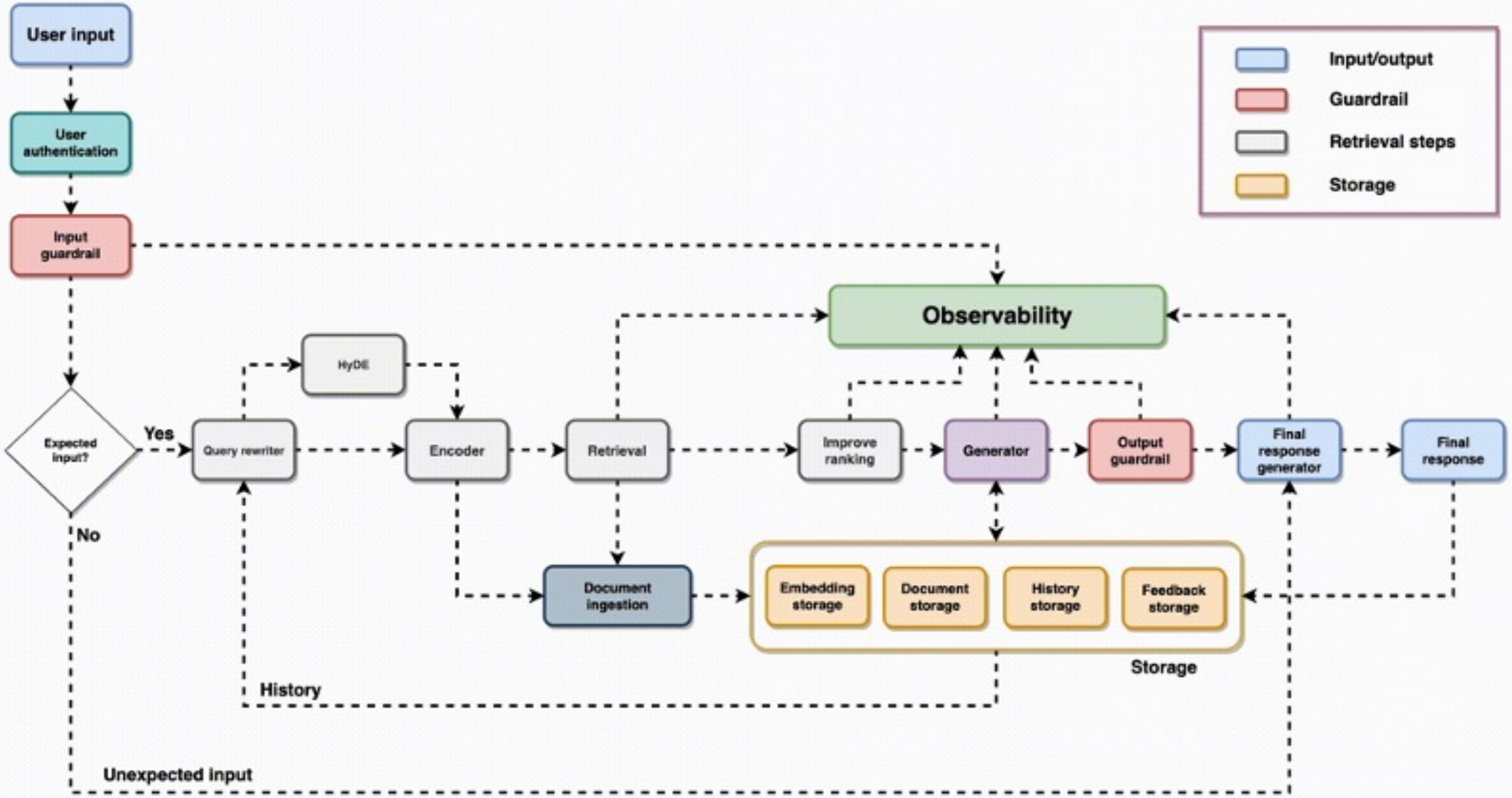
- RAG allows the use of the same model as a reasoning engine over new data provided in a prompt.
    - This technique enables **in-context learning** without the need for expensive fine-tuning, empowering businesses to use LLMs more efficiently.
- RAG combines an information retrieval component with a text generator model.

RAG - Information retrieval component

- The data are broken down into manageable chunks.
- Then they are converted into vector embeddings.
- The database indexes the embeddings for rapid retrieval.
- When a user queries the database, it computes similarity metrics between the chunk vectors and returns matches.
- Vector databases then precompute certain similarities between the vectors to further speed up the queries.

Enterprise RAG

Architecture For Enterprise RAG



Input guardrail

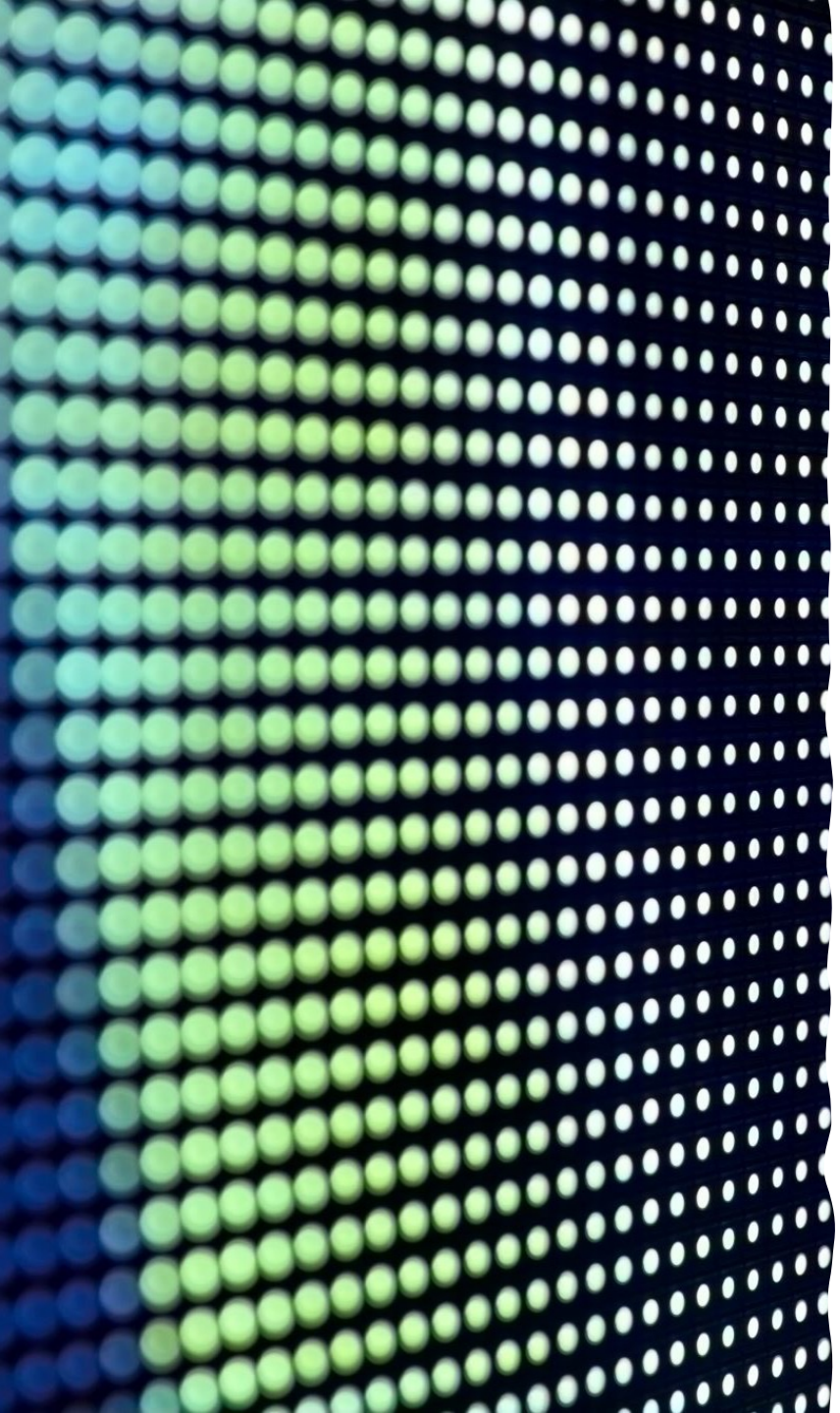
- It’s essential to prevent user inputs that can be harmful or contain private information.
- Studies have shown it's easy to jailbreak LLMs.
- Here’s where input guardrails come in.
- Let's have a look at different scenarios for which we need guardrails.

Input guardrail

- Anonymization
  - Input guardrails can anonymize or **redact** personally identifiable information (PII) such as names, addresses, or contact details. This helps to protect privacy and prevent malicious attempts to disclose sensitive information.

- Restrict substrings
  - Prohibiting certain substrings or patterns that could be exploited for SQL injection, cross-site scripting (XSS), or other injection attacks prevents security vulnerabilities or unwanted behaviors.

- Restrict topics
  - In order to restrict discussions or inputs related to specific topics that may be inappropriate, offensive, or violate community guidelines, it’s important to filter out content that involves hate speech, discrimination, or explicit material.



Input guardrail

- Restrict code
  - It’s essential to prevent the injection of executable code that could compromise system security or lead to code injection attacks.
- Restrict language
  - Verify that text inputs are in the correct language or script, preventing potential misinterpretations or errors in processing.
- Detect prompt injection
  - Mitigate attempts to inject misleading or harmful prompts that may manipulate the system or influence the behavior of LLMs in unintended ways.

Input guardrail

- Limit tokens
  - Enforcing a maximum token or character limit for user inputs helps avoid resource exhaustion and prevents denial-of-service (DoS) attacks.
- Detect toxicity
  - Implement toxicity filters to identify and block inputs that contain harmful or abusive language.

Query rewriter

- Once the query passes the input guardrail, we send it to the query rewriter.
- Sometimes, user queries can be vague or require context to understand the user's intention better.
- Query rewriting is a technique that helps with this. It involves transforming user queries to enhance clarity, precision, and relevance.
- Let's go through some of the most popular techniques.

Query rewriter: Rewrite based on history

- In this method, the system leverages the user's query history to understand the context of the conversation and enhance subsequent queries.
- Let’s use an example of a credit card inquiry.
- Query History:
  - "How many credit cards do you have?"
  - "Are there any yearly fees for platinum and gold credit cards?"
  - "Compare features of both.
- We must identify the context evolution based on the user's query history, discern the user's intent and relationship between queries, and generate a query that aligns with the evolving context.
- Rewritten Query: "Compare features of platinum and gold credit cards."

Query rewriter: Create subqueries

- Complex queries can be difficult to answer due to retrieval issues.
- To simplify the task, queries are broken down into more specific subqueries.
- This helps to retrieve the right context needed for generating the answer.
- Given the query "Compare features of platinum and gold credit card," the system generates subqueries for each card that focus on individual entities mentioned in the original query.
- Rewritten Subqueries:
  - "What are the features of platinum credit cards?"
  - "What are the features of gold credit cards?"

Query rewriter: Create similar queries

- To increase the chances of retrieval of the right document, we generate similar queries based on user input.
- This is to overcome the limitations of the retrieval in semantic or lexical matching.
- If the user asks about credit card features, the system generates related queries.
- Use synonyms, related terms, or domain-specific knowledge to create queries that align with the user's intent.
- Generated Similar Query:
  - “I want to know about platinum credit cards” -> "Tell me about the benefits of platinum credit cards."

Vector Database

- The vector database powering the semantic search is a crucial retrieval component of RAG.
- However, selecting this component appropriately is vital to avoid potential issues.
- Several vector database factors (https://superlinked.com/vector-db-comparison)need to be considered in the selection process.
- Let’s go over some of them.

Vector Database: Recall vs. Latency

- Optimizing for recall (percentage of relevant results) versus latency (time to return results) is a trade-off in vector databases.
- Different indexes like Flat, HNSW (Hierarchical Navigable Small World), PQ (Product quantization), ANNOY, and DiskANN make varying trade-offs between speed and recall.
- Conduct benchmark studies on your data and queries to make an informed decision.

Vector Database: Insertion speed vs. Query speed

- Balancing insertion speed and query speed is vital. Look for vendors that can handle streaming use cases with high insertion speed requirements.
- However, for most organizations, prioritizing querying speed is more relevant.
- Evaluate the vector insertion speed query latency at peak loads to make an informed decision.

Vector Database: In-memory vs. On-disk index storage

- Choosing between in-memory and on-disk storage involves speed and cost trade-offs.
- While in-memory storage offers high speed, some use cases require storing vectors larger than memory.
- Techniques like memory-mapped files allow scaling vector storage without compromising search speed.

Vector Database: Filtering

- Real-world search queries often involve filtering on metadata attributes.
- Pre-filtered search, although seemingly natural, can lead to missing relevant results.
- Post-filtered search may have issues if the filtered attribute is a small fraction of the dataset.

Data Storage

- Since we are dealing with a variety of data, we need dedicated storage for each of them.

- It’s critical to understand the different considerations for every storage type and specific use cases of each.

Data Storage: Embedding

- Database type: SQL/NoSQL
- Storing document embeddings separately allows for swift reindexing without recalculating embeddings for the entire document corpus.
- Additionally, embedding storage acts as a backup, ensuring the preservation of critical information even in the event of system failures or updates.

Data Storage: Documents

- Database type: NoSQL
- Document storage in its raw format is essential for persistent storage.
- This raw format serves as the foundation for various processing stages, such as indexing, parsing, and retrieval.
- It also provides flexibility for future system enhancements, as the original documents remain intact and can be reprocessed as needed.

Data Storage: Chat History

- Database type: NoSQL
- The storage of chat history is imperative for supporting the conversational aspect of the RAG system.
- Chat history storage allows the system to recall previous user queries, responses, and preferences, enabling it to adapt and tailor future interactions based on the user's unique context.
- This historical data is a valuable resource for improving the ML system by leveraging it for research.

Data Storage: User feedback

- Database type: NoSQL/SQL
- User feedback is systematically collected through various interaction mechanisms within the RAG application.
- In most LLM systems, users can provide feedback using thumbs-up/thumbs-down, star ratings and text feedback.
- This array of user insights serves as a valuable repository, encapsulating user experiences and perceptions, forming the basis for ongoing system enhancements.

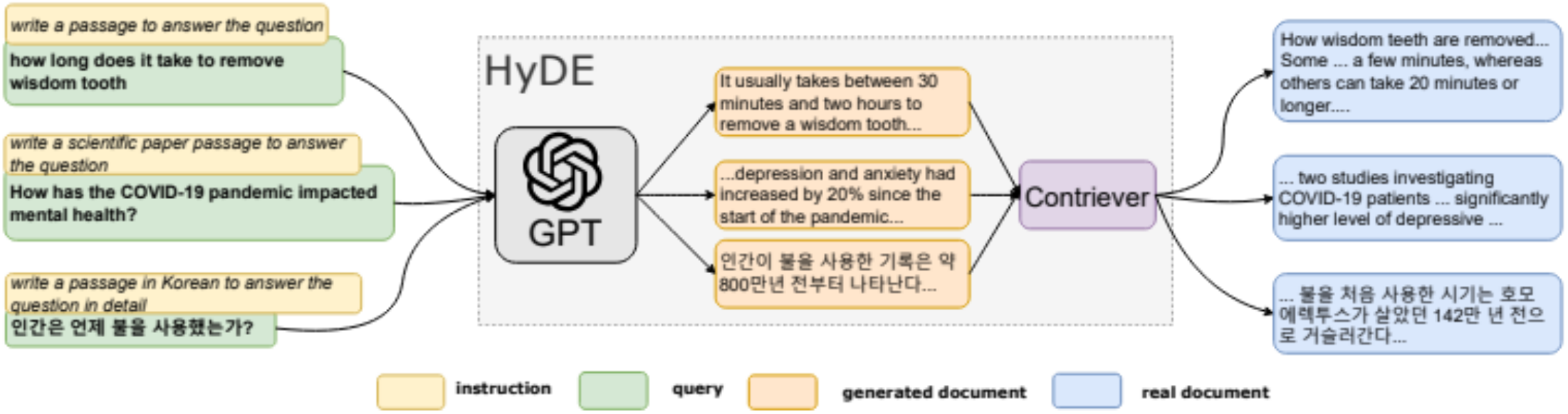
Retrieval Strategy – Techniques to improve

Retrieval Strategy – Techniques to improve

- LLMs can be easily distracted by irrelevant context and having a lot of context (topK retrieved docs) can lead to missing out of certain context due to the attention patterns of LLMs.
- Therefore, it is crucial to improve retrieval with relevant and diverse documents.
- Let's look at some of the proven techniques for improving retrieval.

Hypothetical document embeddings (HyDE)

- Given a query, HyDE first zero-shot instructs an instruction-following language model (e.g. InstructGPT) to generate a hypothetical document.
- The document captures relevance patterns but is unreal and may contain false details.
- Then, an unsupervised contrastively learned encoder (e.g. Contriever) encodes the document into an embedding vector.
- This vector identifies a neighborhood in the corpus embedding space, where similar real documents are retrieved based on vector similarity, which would be better than the embedding of the query.



Maximal Marginal Relevance (MMR)

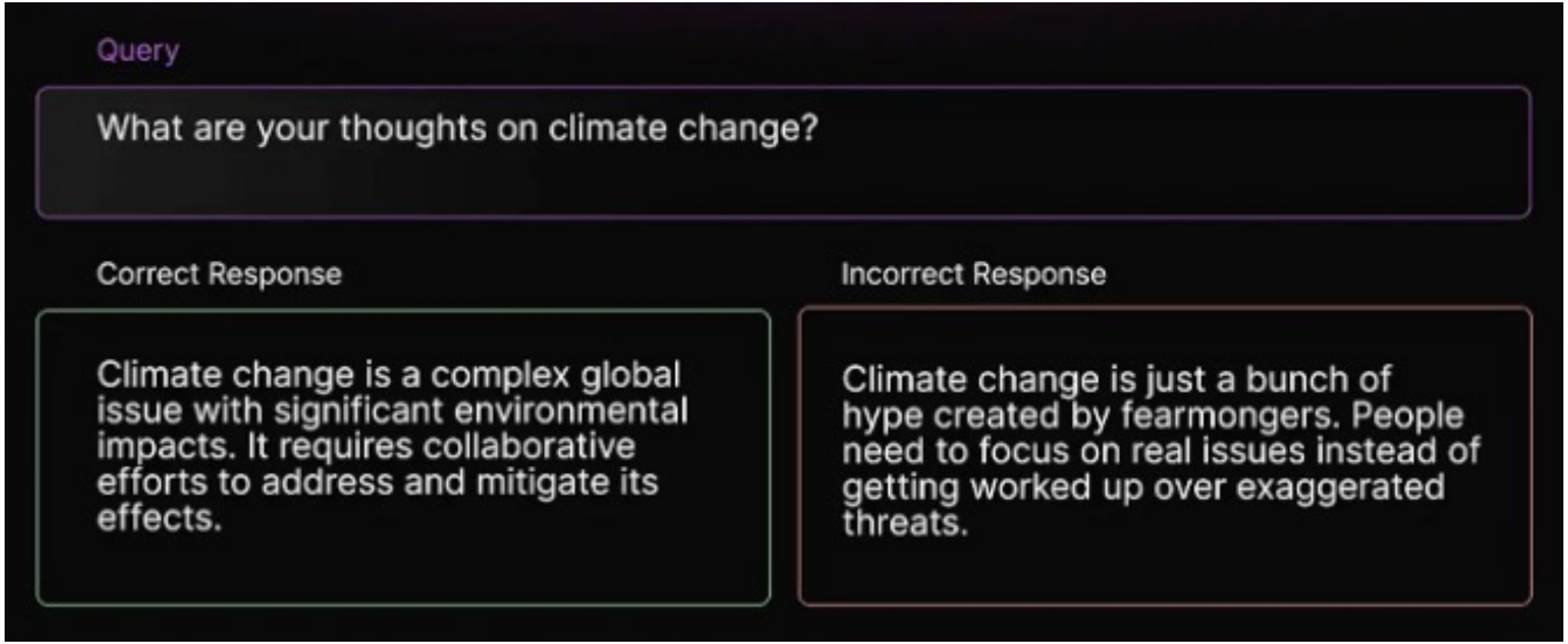
- MMR is a method designed to enhance the diversity of retrieved items in response to a query, avoiding redundancy.
- Rather than focusing solely on retrieving the most relevant items, MMR achieves a balance between relevance and diversity.
- It’s like introducing a friend to people at a party.
- Initially, it identifies the most matching person based on the friend's preferences.
- Then, it seeks someone slightly different.
- This process continues until the desired number of introductions is achieved.
- MMR ensures a more diverse and relevant set of items is presented, minimizing redundancy.

Output guardrail

- The output guardrail functions similarly to its input counterpart but is specifically tailored to detect issues in the generated output.
- It focuses on identifying hallucinations, competitor mentions, and potential brand damage as part of RAG evaluation.
- The goal is to prevent generating inaccurate or ethically questionable information that may not align with the brand's values.
- By actively monitoring and analyzing the output, this guardrail ensures that the generated content remains factually accurate, ethically sound, and consistent with the brand's guidelines.

Output guardrail

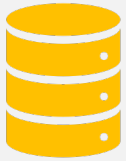
Here is an example of a response that can damage an enterprise brand but would be blocked by an appropriate output guardrail:



Caching

For companies operating at scale, cost can become a hindrance.

Caching is a great way to save money in such cases.



Caching involves the storage of prompts and their corresponding responses in a database, enabling their retrieval for subsequent use.

Caching: Advantages

1. Enhanced production inference
   - Caching contributes to faster and more cost-effective inference during production.
   - Certain queries can achieve near-zero latency by leveraging cached responses, streamlining the user experience.

2. Accelerated development cycles
   - In the development phase, caching proves to be a boon as it eliminates the need to invoke the API for identical prompts repeatedly.
   - This results in faster and more economical development cycles.

3. Data storage
   - The existence of a comprehensive database storing all prompts simplifies the fine-tuning process for LLMs.
   - Utilizing the stored prompt-response pairs streamlines the optimization of the model based on accumulated data.

Multi-tenancy

- SaaS software often has multiple tenants, balancing simplicity and privacy.
- For multi-tenancy in RAG systems, the goal is to build a system that not only finds information effectively but also respects each user's data limits.
- In simpler terms, every user's interaction with the system is separate, ensuring the system only looks at and uses the information that is meant for that user.

Multi-tenancy

- One of the simple ways to build multi-tenancy is by using metadata.
- When we add documents to the system, we include specific user details in the metadata.
- This way each document is tied to a particular user.
- When someone searches, the system uses this metadata to filter and only show documents related to that user.
- It then does a smart search to find the most important information for that user.
- This approach stops private information from being mixed up between different users, keeping each person's data safe and private.

7 Failure Points of RAG Systems

7 Failure Points of RAG Systems

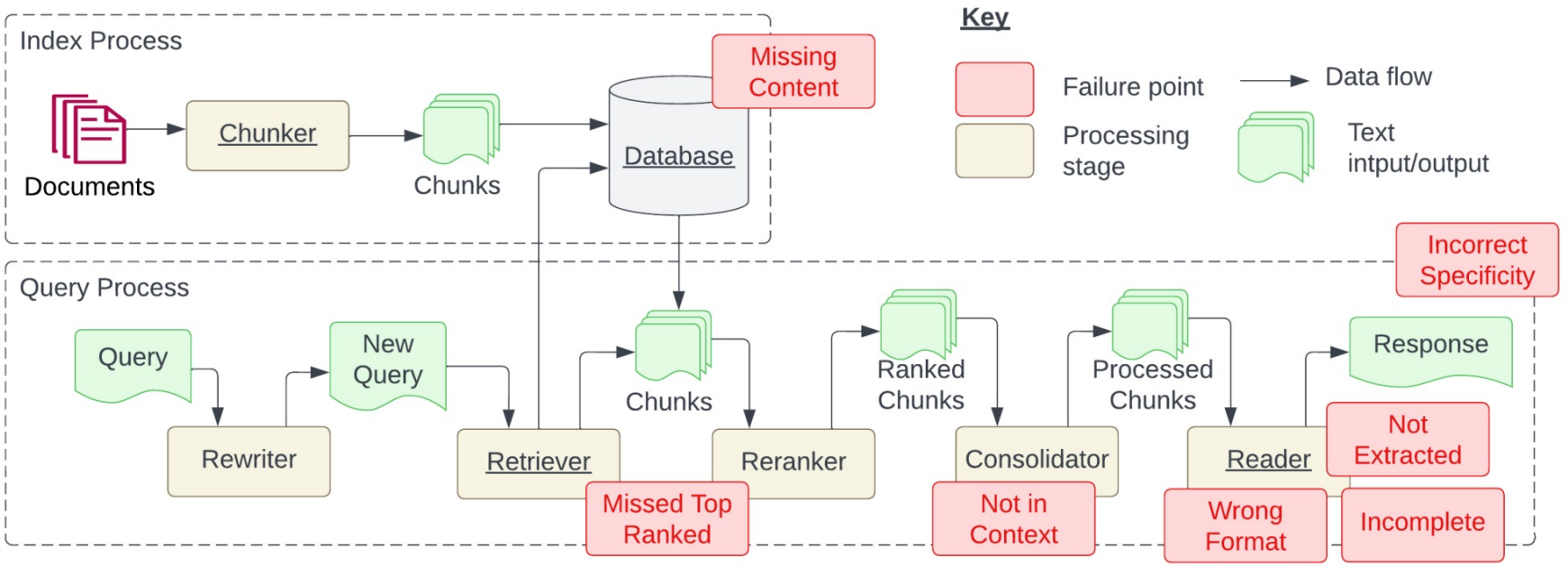


Figure 1: Indexing and Query processes required for creating a Retrieval Augmented Generation (RAG) system. The indexing process is typically done at development time and queries at runtime. Failure points identified in this study are shown in red boxes. All required stages are underlined. Figure expanded from [19].

https://arxiv.org/abs/2401.05856

Missing content (FP1)

- A question is posed that cannot be answered with the available documents.
- In the ideal scenario, the RAG system responds with a message like "Sorry, I don’t know."
- However, for questions related to content without clear answers, the system might be misled into providing a response.

Missed the top ranked documents (FP2)

- The answer to a question is present in the document but did not rank highly enough to be included in the results returned to the user.

- While all documents are theoretically ranked and utilized in subsequent steps, in practice only the top K documents are returned, with K being a value selected based on performance.

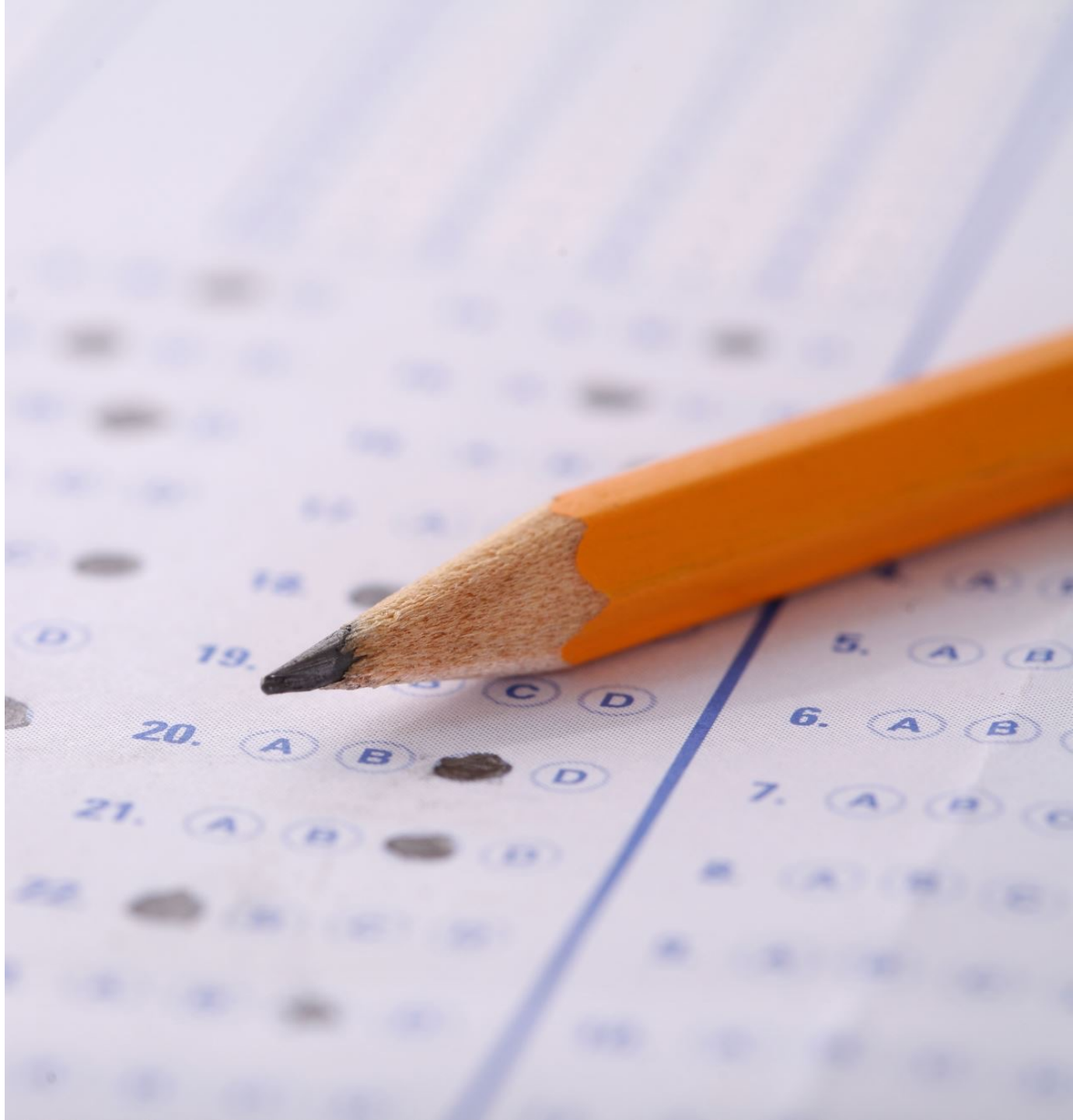
Not in context - consolidation strategy limitations (FP3)

- Documents containing the answer are retrieved from the database but fail to make it into the context for generating a response.

- This occurs when a substantial number of documents are returned, leading to a consolidation process where the relevant answer retrieval is hindered.

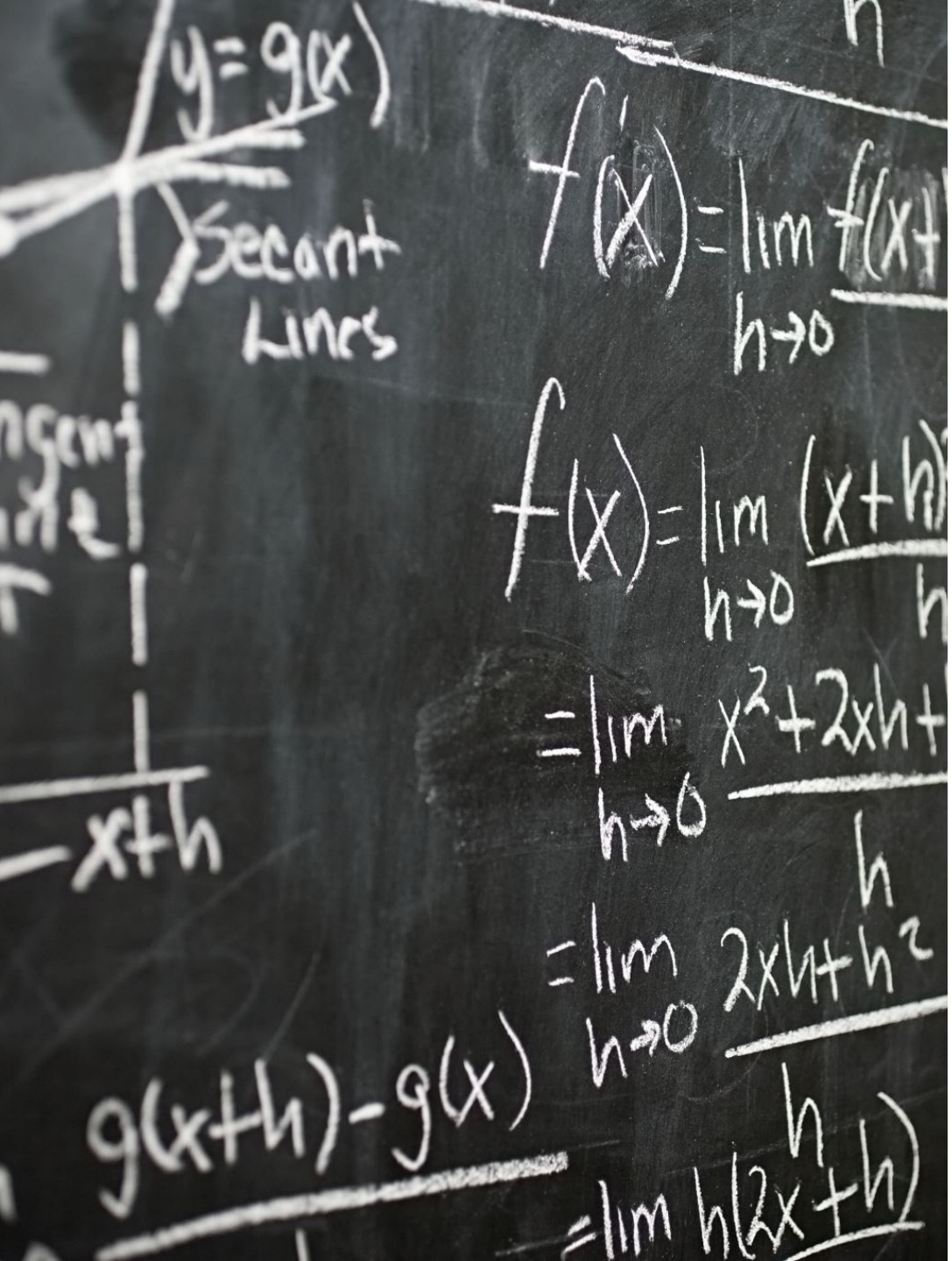
Not extracted (FP4)

- The answer is present in the context, but the model fails to extract the correct information.
- This typically happens when there is excessive noise or conflicting information in the context.



Wrong format (FP5)

- The question involves extracting information in a specific format, such as a table or list, and the model disregards the instruction.



Incorrect specificity (FP6)

- The response includes the answer but lacks the required specificity or is overly specific, failing to address the user's needs.
- This occurs when RAG system designers have a predetermined outcome for a given question, such as teachers seeking educational content.
- In such cases, specific educational content should be provided along with answers. Incorrect specificity also arises when users are unsure how to phrase a question and are too general.

Incomplete (FP7)

- Incomplete answers are accurate but lack some information, even though that information was present in the context and available for extraction.

- For instance, a question like “What are the key points covered in documents A, B, and C?” would be better approached by asking these questions separately.

Lessons learnt

FP	Lesson	Description	Case Studies
FP4	Larger context get better results (Context refers to a particular setting or situation in which the content occurs)	A larger context enabled more accurate responses (8K vs 4K). Contrary to prior work with GPT-3.5 [13]	AI Tutor
FP1	Semantic caching drives cost and latency down	RAG systems struggle with concurrent users due to rate limits and the cost of LLMs. Prepopulate the semantic cache with frequently asked questions [1].	AI Tutor
FP5-7	Jailbreaks bypass the RAG system and hit the safety training.	Research suggests fine-tuning LLMs reverses safety training [11], test all fine-tuned LLMs for RAG system.	AI Tutor
FP2, FP4	Adding meta-data improves retrieval.	Adding the file name and chunk number into the retrieved context helped the reader extract the required information. Useful for chat dialogue.	AI Tutor
FP2, FP4-7	Open source embedding models perform better for small text.	Opensource sentence embedding models performed as well as closed source alternatives on small text.	BioASQ, AI Tutor
FP2-7	RAG systems require continuous calibration.	RAG systems receive unknown input at runtime requiring constant monitoring.	AI Tutor, BioASQ
FP1, FP2	Implement a RAG pipeline for configuration.	A RAG system requires calibrating chunk size, embedding strategy, chunking strategy, retrieval strategy, consolidation strategy, context size, and prompts.	Cognitive Reviewer, AI Tutor, BioASQ
FP2, FP4	RAG pipelines created by assembling bespoke solutions are suboptima.	End-to-end training enhances domain adaptation in RAG systems [18].	BioASQ, AI Tutor
FP2-7	Testing performance characteristics are only possible at runtime.	Offline evaluation techniques such as G-Evals [14] look promising but are premised on having access to labelled question and answer pairs.	Cognitive Reviewer, AI Tutor

RAG going wrong

- **Air Canada Loses Court Case After Its Chatbot Hallucinated Fake Policies To a Customer**
- The airline argued that the chatbot itself was liable. The court disagreed.

Hallucinations

- Hallucination in AI means the AI makes up things that sound real but are either wrong or not related to the context.
- This often happens because the AI has built-in biases, doesn't fully understand the real world, or its training data isn't complete.
- In these instances, the AI produces information it wasn't specifically taught, leading to responses that can be incorrect or misleading.

Types of hallucinations - Intrinsic

- These are made-up details that directly conflict with the original information.

- For example, if the original content says, "The first Ebola vaccine was approved by the FDA in 2019," but the summarized version says, "The first Ebola vaccine was approved in 2021," then that's an intrinsic hallucination.

Types of hallucinations - **Extrinsic**

- These are details added to the summarized version that can't be confirmed or denied by the original content.
- For instance, if the summary includes "China has already started clinical trials of the COVID-19 vaccine," but the original content doesn't mention this, it's an extrinsic hallucination.
- Even though it may be true and add useful context, it's seen as risky as it's not verifiable from the original information.

Hallucinations in Abstractive summarization

- There are two types of hallucination in summarization -- **intrinsic**, where the summary says something opposite to the original text, and **extrinsic**, where the summary includes something not mentioned in the original at all.
- For example, if an article says the FDA approved the first Ebola vaccine in 2019, an **intrinsic** hallucination would be to say the FDA rejected it.
- An **extrinsic** hallucination example might be to claim that China has started testing a COVID-19 vaccine when the original article doesn't mention that at all.

Hallucinations in Dialogue generation

- In open domain dialogue generation, a chatbot either gives the user necessary details or keeps them interested with new replies without rehashing previous conversation.
  - A little bit of hallucination might be acceptable in this context.

- **Intrinsic hallucination** is when the chatbot's response contradicts the previous conversation or external knowledge.
  - The bot might incorrectly interpret a moderate price range as high or mix up names like 'Roger Federer' and 'Rafael Nadal.'

- **Extrinsic hallucination** happens when we can't cross-check the chatbot's response with previous dialogue or outside knowledge.
  - For instance, the bot might claim something about the Pickwick hotel being in San Diego or Djokovic being in the top ten singles players, but without enough information to confirm or deny this.

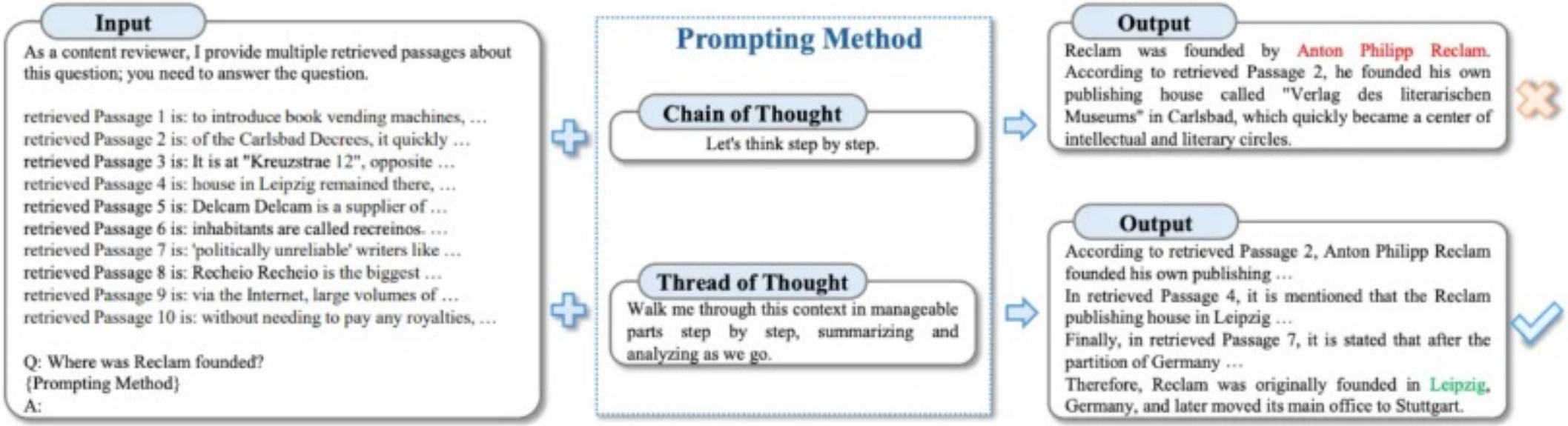
Hallucinations in Data to text generation

- **Data-to-Text Generation** is the process of creating written descriptions based on data like tables, database records, or knowledge graphs.

- **Intrinsic Hallucinations** is when the created text includes information that goes against the data provided.
  - For instance, if a table lists a team's win-loss record as 18-5, but the generated text says 18-4, that's an Intrinsic Hallucination.

- **Extrinsic Hallucinations** is when the text includes extra information that isn't related to the provided data.
  - An example would be if the text mentioned a team's recent wins, even though that information wasn't in the data.

LLM Prompting Techniques For Reducing Hallucinations - Thread of Thought(ThoT)



Thread of Thought prompting enables large language models to tackle chaotic context problems. In the output depicted, green text denotes the correct answer, while red text indicates the erroneous prediction.

Thread of Thought(ThoT) - Learnings from ThoT prompt optimisation

- In their study, the authors tested different prompts and found that the model's accuracy improved when provided with more detailed instructions for analyzing and understanding the context.
- On the other hand, prompts with less guidance, such as **prompt 29**, resulted in lower scores.

No.	Template	EM
1	Let's read through the document section by section, analyzing each part carefully as we go.	0.43
2	Take me through this long document step-by-step, making sure not to miss any important details.	0.47
3	Divide the document into manageable parts and guide me through each one, providing insights as we move along.	0.51
4	Analyze this extensive document in sections, summarizing each one and noting any key points.	0.47
5	Let's go through this document piece by piece, paying close attention to each section.	0.50
6	Examine the document in chunks, evaluating each part critically before moving to the next.	0.49
7	Walk me through this lengthy document segment by segment, focusing on each part's significance.	0.52
8	Let's dissect this document bit by bit, making sure to understand the nuances of each section.	0.45
9	Systematically work through this document, summarizing and analyzing each portion as we go.	0.45
10	Navigate through this long document by breaking it into smaller parts and summarizing each, so we don't miss anything.	0.48
11	Let's explore the context step-by-step, carefully examining each segment.	0.44
12	Take me through the context bit by bit, making sure we capture all important aspects.	0.49
13	Let's navigate through the context section by section, identifying key elements in each part.	0.47
14	Systematically go through the context, focusing on each part individually.	0.46
15	Let's dissect the context into smaller pieces, reviewing each one for its importance and relevance.	0.47
16	Analyze the context by breaking it down into sections, summarizing each as we move forward.	0.49
17	Guide me through the context part by part, providing insights along the way.	0.52
18	Examine each segment of the context meticulously, and let's discuss the findings.	0.44
19	Approach the context incrementally, taking the time to understand each portion fully.	0.42
20	Carefully analyze the context piece by piece, highlighting relevant points for each question.	0.47
21	In a step-by-step manner, go through the context, surfacing important information that could be useful.	0.53
22	Methodically examine the context, focusing on key segments that may answer the query.	0.45
23	Progressively sift through the context, ensuring we capture all pertinent details.	0.46
24	Navigate through the context incrementally, identifying and summarizing relevant portions.	0.48
25	Let's scrutinize the context in chunks, keeping an eye out for information that answers our queries.	0.42
26	Take a modular approach to the context, summarizing each part before drawing any conclusions.	0.47
27	Read the context in sections, concentrating on gathering insights that answer the question at hand.	0.48
28	Proceed through the context systematically, zeroing in on areas that could provide the answers we're seeking.	0.49
29	Let's take a segmented approach to the context, carefully evaluating each part for its relevance to the questions posed.	0.39
30	Walk me through this context in manageable parts step by step, summarizing and analyzing as we go.	**0.55**

Thread of Thought(ThoT) - Learnings from ThoT prompt optimisation

- The **top-performing prompt** directs the model to handle complexity by breaking it down into parts and instructs the model to both summarize and analyze, encouraging active engagement beyond passive reading or understanding.

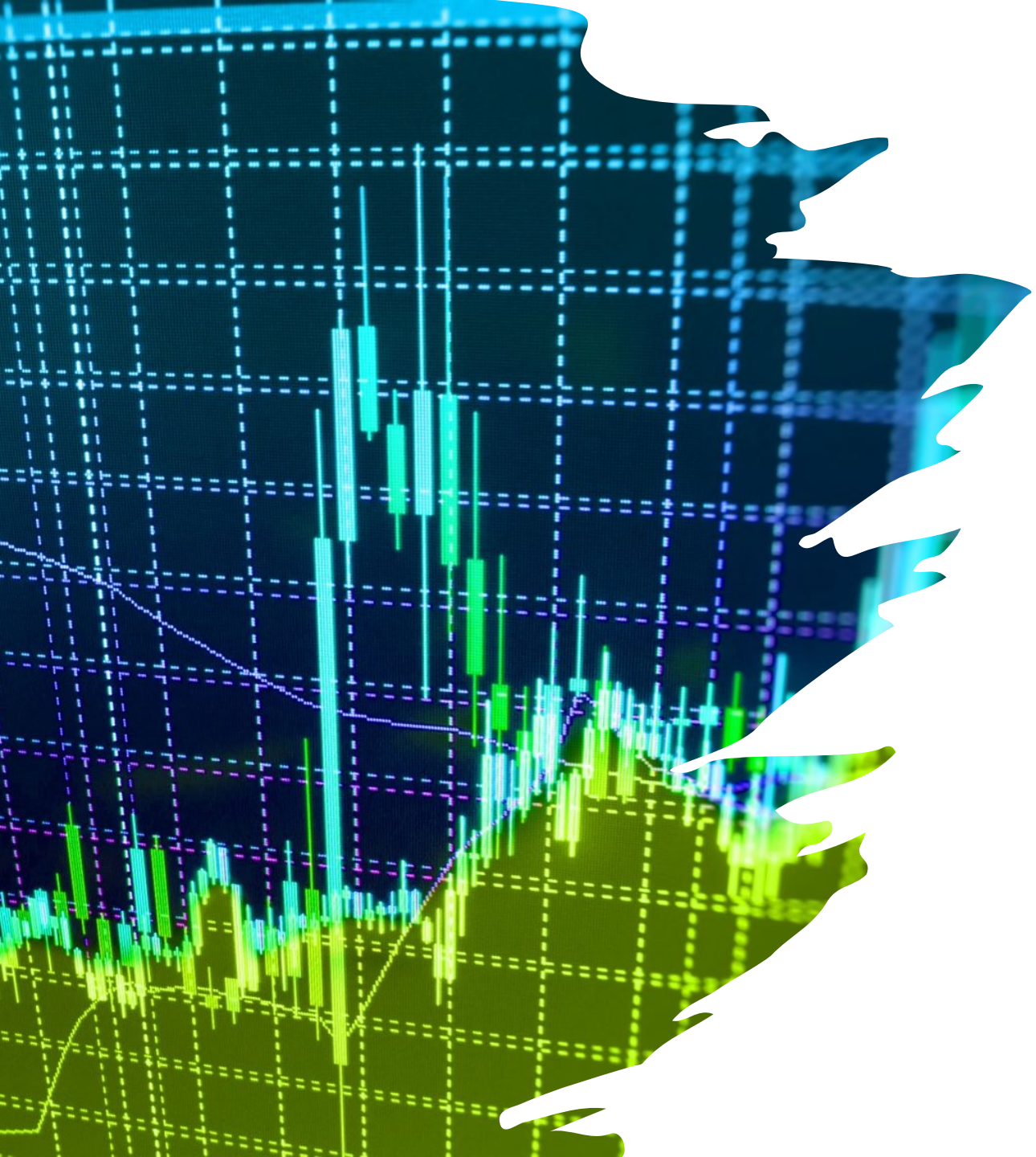
No.	Template	EM
1	Let's read through the document section by section, analyzing each part carefully as we go.	0.43
2	Take me through this long document step-by-step, making sure not to miss any important details.	0.47
3	Divide the document into manageable parts and guide me through each one, providing insights as we move along.	0.51
4	Analyze this extensive document in sections, summarizing each one and noting any key points.	0.47
5	Let's go through this document piece by piece, paying close attention to each section.	0.50
6	Examine the document in chunks, evaluating each part critically before moving to the next.	0.49
7	Walk me through this lengthy document segment by segment, focusing on each part's significance.	0.52
8	Let's dissect this document bit by bit, making sure to understand the nuances of each section.	0.45
9	Systematically work through this document, summarizing and analyzing each portion as we go.	0.45
10	Navigate through this long document by breaking it into smaller parts and summarizing each, so we don't miss anything.	0.48
11	Let's explore the context step-by-step, carefully examining each segment.	0.44
12	Take me through the context bit by bit, making sure we capture all important aspects.	0.49
13	Let's navigate through the context section by section, identifying key elements in each part.	0.47
14	Systematically go through the context, focusing on each part individually.	0.46
15	Let's dissect the context into smaller pieces, reviewing each one for its importance and relevance.	0.47
16	Analyze the context by breaking it down into sections, summarizing each as we move forward.	0.49
17	Guide me through the context part by part, providing insights along the way.	0.52
18	Examine each segment of the context meticulously, and let's discuss the findings.	0.44
19	Approach the context incrementally, taking the time to understand each portion fully.	0.42
20	Carefully analyze the context piece by piece, highlighting relevant points for each question.	0.47
21	In a step-by-step manner, go through the context, surfacing important information that could be useful.	0.53
22	Methodically examine the context, focusing on key segments that may answer the query.	0.45
23	Progressively sift through the context, ensuring we capture all pertinent details.	0.46
24	Navigate through the context incrementally, identifying and summarizing relevant portions.	0.48
25	Let's scrutinize the context in chunks, keeping an eye out for information that answers our queries.	0.42
26	Take a modular approach to the context, summarizing each part before drawing any conclusions.	0.47
27	Read the context in sections, concentrating on gathering insights that answer the question at hand.	0.48
28	Proceed through the context systematically, zeroing in on areas that could provide the answers we're seeking.	0.49
29	Let's take a segmented approach to the context, carefully evaluating each part for its relevance to the questions posed.	0.39
30	Walk me through this context in manageable parts step by step, summarizing and analyzing as we go.	**0.55**

Thread of Thought(ThoT) - Learnings from ThoT prompt optimisation

- The findings suggest that structured prompts, promoting a **detailed analytical process** through step-by-step dissection, summarization, and critical evaluation, contribute to enhanced model performance and reduced hallucinations.

No.	Template	EM
1	Let's read through the document section by section, analyzing each part carefully as we go.	0.43
2	Take me through this long document step-by-step, making sure not to miss any important details.	0.47
3	Divide the document into manageable parts and guide me through each one, providing insights as we move along.	0.51
4	Analyze this extensive document in sections, summarizing each one and noting any key points.	0.47
5	Let's go through this document piece by piece, paying close attention to each section.	0.50
6	Examine the document in chunks, evaluating each part critically before moving to the next.	0.49
7	Walk me through this lengthy document segment by segment, focusing on each part's significance.	0.52
8	Let's dissect this document bit by bit, making sure to understand the nuances of each section.	0.45
9	Systematically work through this document, summarizing and analyzing each portion as we go.	0.45
10	Navigate through this long document by breaking it into smaller parts and summarizing each, so we don't miss anything.	0.48
11	Let's explore the context step-by-step, carefully examining each segment.	0.44
12	Take me through the context bit by bit, making sure we capture all important aspects.	0.49
13	Let's navigate through the context section by section, identifying key elements in each part.	0.47
14	Systematically go through the context, focusing on each part individually.	0.46
15	Let's dissect the context into smaller pieces, reviewing each one for its importance and relevance.	0.47
16	Analyze the context by breaking it down into sections, summarizing each as we move forward.	0.49
17	Guide me through the context part by part, providing insights along the way.	0.52
18	Examine each segment of the context meticulously, and let's discuss the findings.	0.44
19	Approach the context incrementally, taking the time to understand each portion fully.	0.42
20	Carefully analyze the context piece by piece, highlighting relevant points for each question.	0.47
21	In a step-by-step manner, go through the context, surfacing important information that could be useful.	0.53
22	Methodically examine the context, focusing on key segments that may answer the query.	0.45
23	Progressively sift through the context, ensuring we capture all pertinent details.	0.46
24	Navigate through the context incrementally, identifying and summarizing relevant portions.	0.48
25	Let's scrutinize the context in chunks, keeping an eye out for information that answers our queries.	0.42
26	Take a modular approach to the context, summarizing each part before drawing any conclusions.	0.47
27	Read the context in sections, concentrating on gathering insights that answer the question at hand.	0.48
28	Proceed through the context systematically, zeroing in on areas that could provide the answers we're seeking.	0.49
29	Let's take a segmented approach to the context, carefully evaluating each part for its relevance to the questions posed.	0.39
30	Walk me through this context in manageable parts step by step, summarizing and analyzing as we go.	**0.55**

Impact of Embeddings on RAG Performance



Impact of Embeddings on RAG Performance

- Which encoder you select to generate embeddings is a critical decision, hugely impacting the overall success of the RAG system.
- Low quality embeddings lead to poor retrieval.
- Let’s review some of the selection criteria to consider before making your decision.

Vector Dimension and Performance Evaluation

- When selecting an embedding model, consider the vector dimension, average retrieval performance, and model size.
- The **Massive Text Embedding Benchmark** (MTEB) -- https://huggingface.co/spaces/mteb/leaderboard -- provides insights into popular embedding models from OpenAI, Cohere, and Voyager, among others.
- However, custom evaluation on your dataset is essential for accurate performance assessment.

Private vs. Public Embedding Model

- Although the embedding model provides ease of use, it entails certain trade-offs.
- The private embedding API, in particular, offers high availability without the need for intricate model hosting engineering.
- However, this convenience is counterbalanced by scaling limitations.
- It is crucial to verify the rate limits and explore options for increasing them.
- Additionally, a notable advantage is that model improvements come at no extra cost.

Cost Considerations

Embedding Price Comparison

Provider	Model	input price per 1K Token	input price per 1M Token	Compared to Open AI Ada	Compared to Cohere
Amazon Bedrock	Titan Embeddigs	$0.00010	$0.10000	0.00%	0.00%
(Azure) OpenAI	Ada Embeddings	$0.00010	$0.10000	0.00%	0.00%
	Embedding 3 small	$0.00002	$0.02000	-80.00%	-80.00%
	Embedding 3 Large	$0.00013	$0.13000	30.00%	30.00%
Cohere	Embed	$0.00010	$0.10000	0.00%	0.00%
Google Vertex AI 1 token ~= 4 chars	Text Embeddings	$0.00040	$0.40000	300.00%	300.00%
Together	BGE-Base	$0.00003	$0.02800	-72.00%	-72.00%
Anyscale	thenlper-gte-large	$0.00005	$0.05000	-50.00%	-50.00%
MosaicML	Instructor-Large	$0.00010	$0.10000	0.00%	0.00%
	Instructor-XL	$0.00020	$0.20000	100.00%	100.00%

Querying, Indexing and Storage Cost

- Ensure high availability of the embedding API service, considering factors like model size and latency needs.
  - OpenAI and similar providers offer reliable APIs, while open-source models may require additional engineering efforts.
- The cost of indexing documents is influenced by the chosen encoder service.
  - Separate storage of embeddings is advisable for flexibility in service resets or reindexing.
- Storage cost scales linearly with dimension, and the choice of embeddings, such as OpenAI's in 1526 dimensions, impacts the overall cost.
  - Calculate average units per document to estimate storage cost.

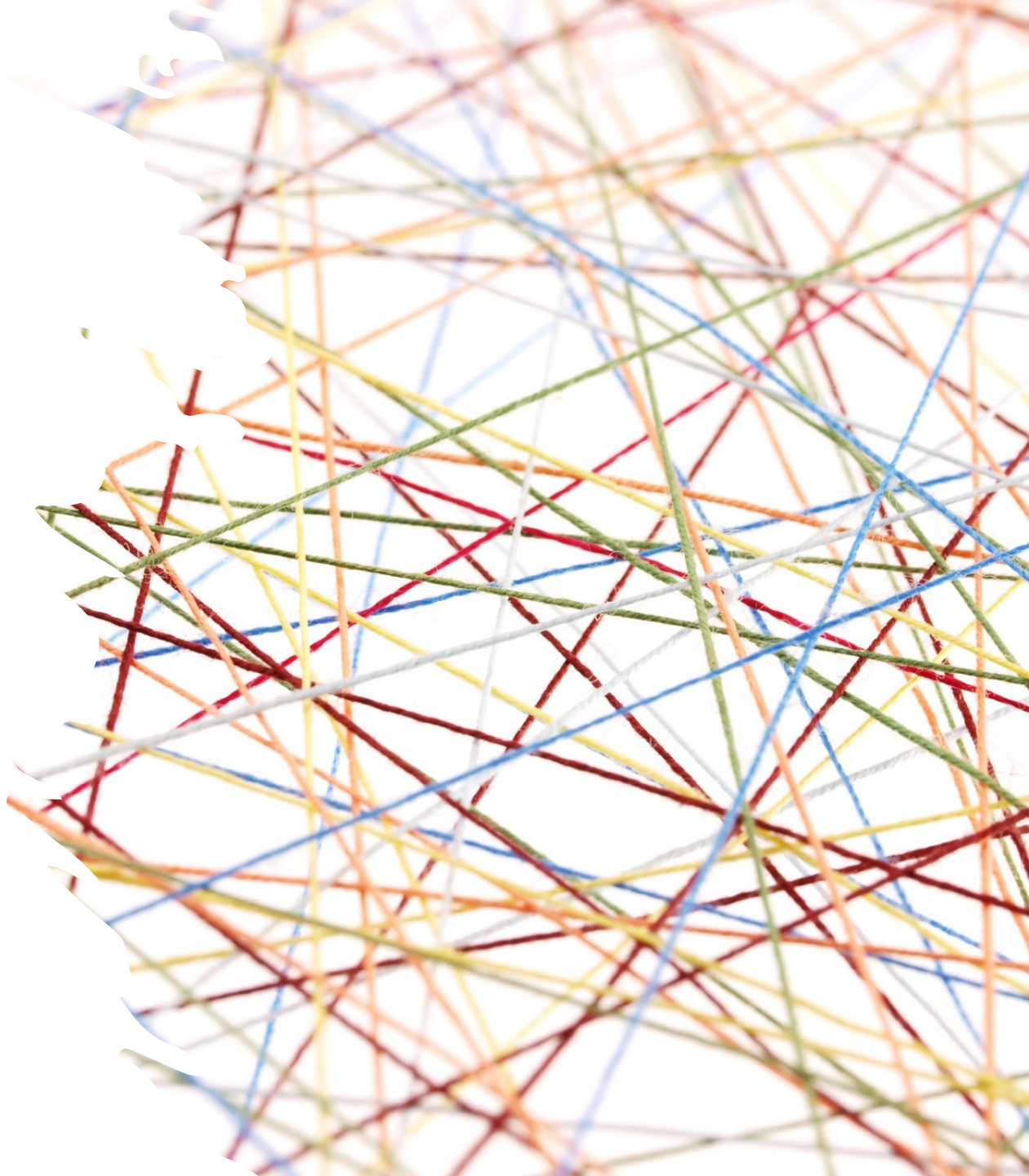
Search Latency & Granularity of text

- The latency of semantic search grows with the dimension of embeddings.
  - Opt for low dimensional embeddings to minimize latency.
- Various levels of granularity, including word-level, sentence-level, and document-level representations, influence the depth of semantic information embedded.
  - For example, optimizing relevance and minimizing noise in the embedding process can be achieved by segmenting large text into smaller chunks.
- Due to the constrained vector size available for storing textual information, embeddings become noisy with longer text.

Impact of chunking

- Chunking plays a central role in various aspects of RAG systems, exerting influence not only on retrieval quality but also on response.

- Let's understand these aspects in more detail.



Impact of chunking

- Chunking plays a central role in various aspects of RAG systems, exerting influence not only on retrieval quality but also on response.

- The primary objective of chunking is to enhance the retrieval quality of information from vector databases.
    - By defining the unit of information that is stored, chunking allows for retrieval of the most relevant information needed for the task.

Impact of chunking

- Vector database cost
  - Efficient chunking techniques help optimize storage by balancing granularity.
  - The cost of storage grows linearly with the number of chunks; therefore, chunks should be as large as possible to keep costs low.

- Vector database query latency
  - Maintaining low latency is essential for real-time applications.
  - Minimizing the number of chunks reduces latency.

Impact of chunking

- LLM latency and cost
  - The mind-blowing capabilities of LLMs come at a considerable price. Improved context from larger chunk sizes increases latency and serving costs.

- LLM hallucinations
  - While adding more context may seem better, excessive context can lead to hallucinations in LLMs.
  - Choosing the right chunking size for the job plays a large role in determining the quality of generated content.
  - Balancing contextual richness with retrieval precision is essential to prevent hallucinatory outputs and ensure a coherent and accurate generation process.

Factors influencing chunking

- Text structure
  - The text structure, whether it's a sentence, paragraph, code, table, or transcript, significantly impacts the chunk size.
  - Understanding how structure relates to the type of content will help influence chunking strategy.

- Embedding model
  - The capabilities and limitations of the embedding model play a crucial role in defining chunk size.
  - Factors such as the model's context input length and its ability to maintain high-quality embeddings guide the optimal chunking strategy.

Factors influencing chunking

- LLM context length
  - LLMs have finite context windows.
  - Chunk size directly affects how much context can be fed into the LLM.
  - Due to context length limitations, large chunks force the user to keep the top k in retrieval as low as possible.

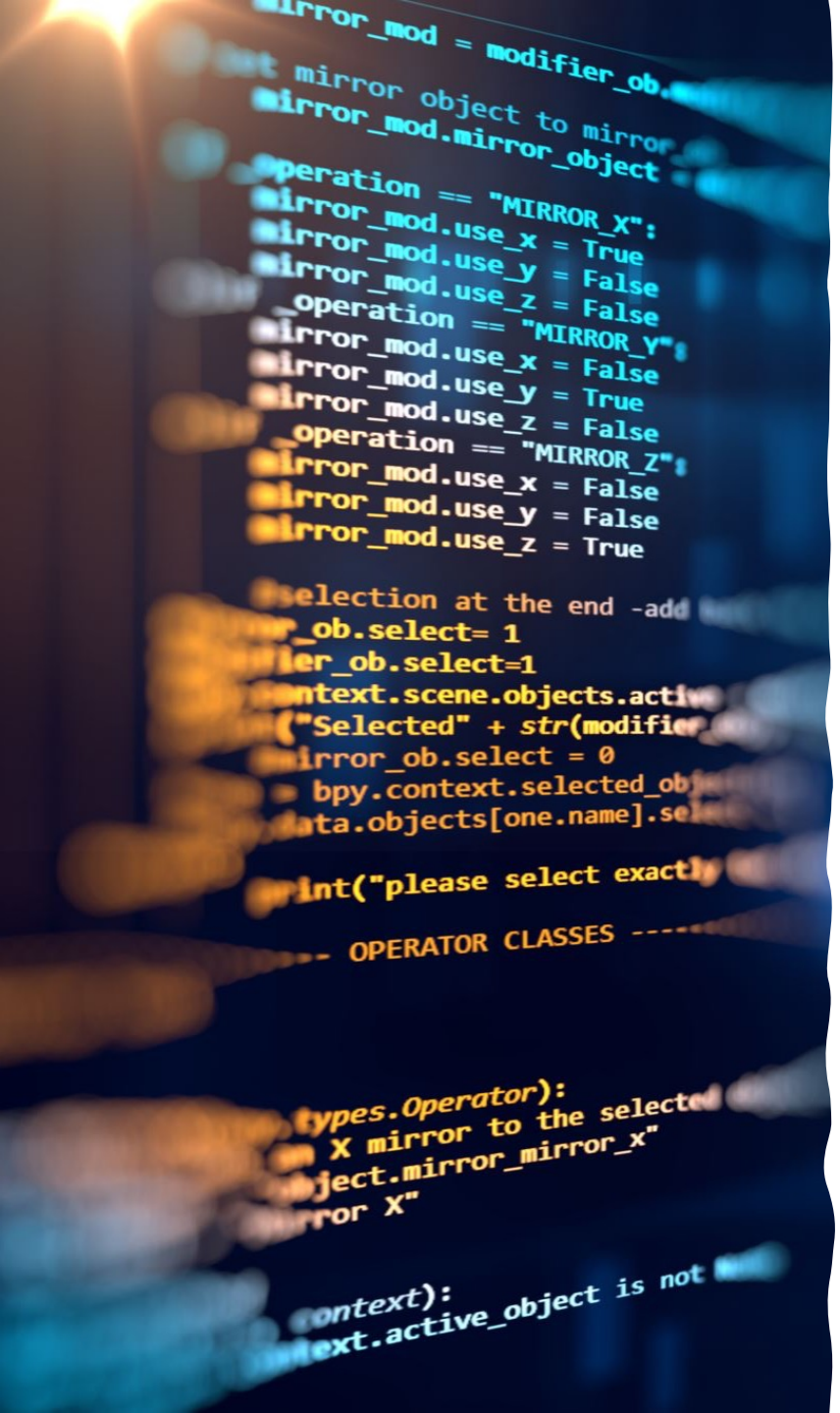
- Type of questions
  - The questions that users will be asking helps determine the chunking techniques best suited for your use case.
  - Specific factual questions, for instance, may require a different chunking approach than complex questions which will require information from multiple chunks.

Types of chunking

- As you see, selecting the right chunk size involves a delicate balance of multiple factors.
- There is no one-size-fits-all approach, emphasizing the importance of finding a chunking technique tailored to the RAG application's specific needs.
- Let's delve into common chunking techniques to help AI builders optimize their RAG performance.

Chunking Techniques For RAG

Technique	Usecase	Pros	Cons
Character splitter	Text	Versatile: Handles various separators Flexible: Adapts to different languages Cost-Effective: Does not require a ML model	Performance: May have increased computational load Complexity: Requires parameter tuning Sentence Interruption: May cut sentences midway
Recursive character splitter	Text, code	Versatile: Handles various separators Flexible: Adapts to different languages Cost-Effective: Does not require a ML model	Performance: Recursive nature may increase computational load Complexity: Requires parameter tuning Sentence Interruption: May cut sentences midway
Sentence splitter	Text	Considers Sentence Boundaries: Avoids cutting sentences prematurely Customizable: Parameters for stride and overlap Cost-Effective: Works with light sentence segmenter	Lack of Versatility: Limited to sentence-based chunks Overlap Issues: May lead to redundancy
Semantic splitter	Text, Chat	Contextual Grouping: Organizes text based on semantic similarity Overcomes Challen ges: Handles chunk size and overlap	Complexity: Requires similarity model and tuning Parameter Dependency: Relies on setting appropriate parameters Resource Intensive: Demands computational resources



Types of chunking: Character splitter

- Langchain’s CharacterTextSplitter class is responsible for breaking down a given text into smaller chunks.
- It uses a separator such as "\n" to identify points where the text should be split.
- The method first splits the text using the specified separator and then merges the resulting splits into a list of chunks.
- The size of these chunks is determined by parameters such as chunk_size and chunk_overlap.

Types of chunking: Recursive Character Splitter

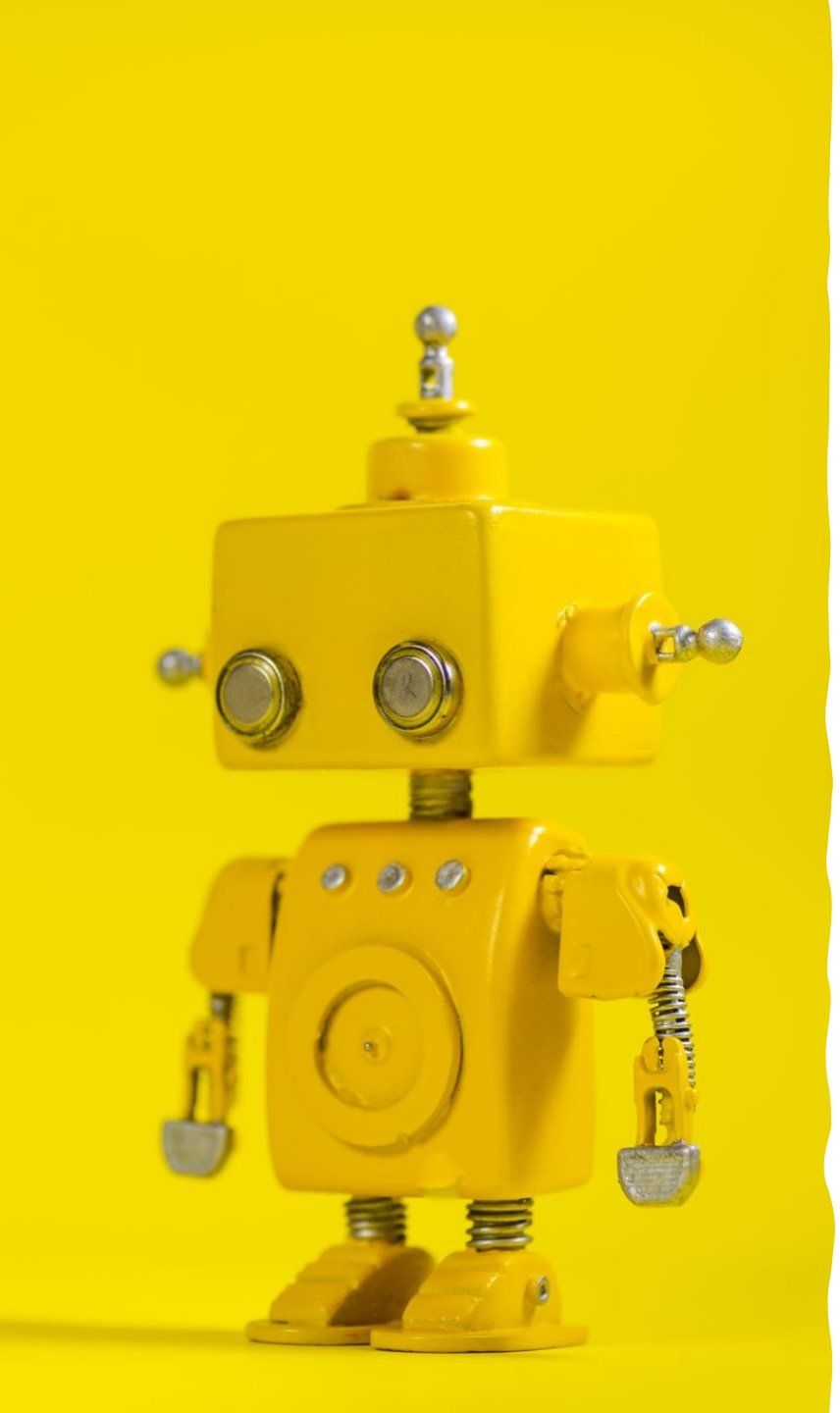
- Langchain’s RecursiveCharacterTextSplitter class is designed to break down a given text into smaller chunks by recursively attempting to split it using different separators.

- This class is particularly useful when a single separator may not be sufficient to identify the desired chunks.

Sentence splitter

- Character splitting poses an issue as it tends to cut sentences midway. Despite attempts to address this using chunk size and overlap, sentences can still be cut off prematurely.

- The SpacySentenceTokenizer takes a piece of text and divides it into smaller chunks, with each chunk containing a certain number of sentences.
  - It uses the Spacy library to analyze the input text and identify individual sentences.



Sentence splitter

- Input = "I love dogs. They are amazing. Cats must be the easiest pets around. Tesla robots are advanced now with AI. They will take us to mars.”
- Output = [

  'I love dogs. They are amazing. Cats must be the easiest pets around.',

  'They are amazing. Cats must be the easiest pets around. Tesla robots are advanced now with AI.',

  'Cats must be the easiest pets around. Tesla robots are advanced now with AI. They will take us to mars.',

  'Tesla robots are advanced now with AI. They will take us to mars.',

  'They will take us to mars.’

  ]

Sentence splitter

- Output = [

  'I love dogs. They are amazing. Cats must be the easiest pets around.',

  'They are amazing. Cats must be the easiest pets around. Tesla robots are advanced now with AI.',

  'Cats must be the easiest pets around. Tesla robots are advanced now with AI. They will take us to mars.',

  'Tesla robots are advanced now with AI. They will take us to mars.',

  'They will take us to mars.’

  ]

- The example below shows how a text with pronouns like “they” requires the context of the previous sentence to make sense.
- Our brute force overlap approach helps here but is also redundant at some places and leads to longer chunks

Semantic splitting

- The SimilarSentenceSplitter takes a piece of text and divides it into groups of sentences based on their similarity.
- It utilizes a similarity model to measure how similar each sentence is to its neighboring sentences.
- The method uses a sentence splitter to break the input text into individual sentences.

Semantic splitting

- The goal is to create groups of sentences where each group contains related sentences, according to the specified similarity model.
- The method starts with the first sentence in the first group and then iterates through the remaining sentences.
- It decides whether to add a sentence to the current group based on its similarity to the previous sentence.

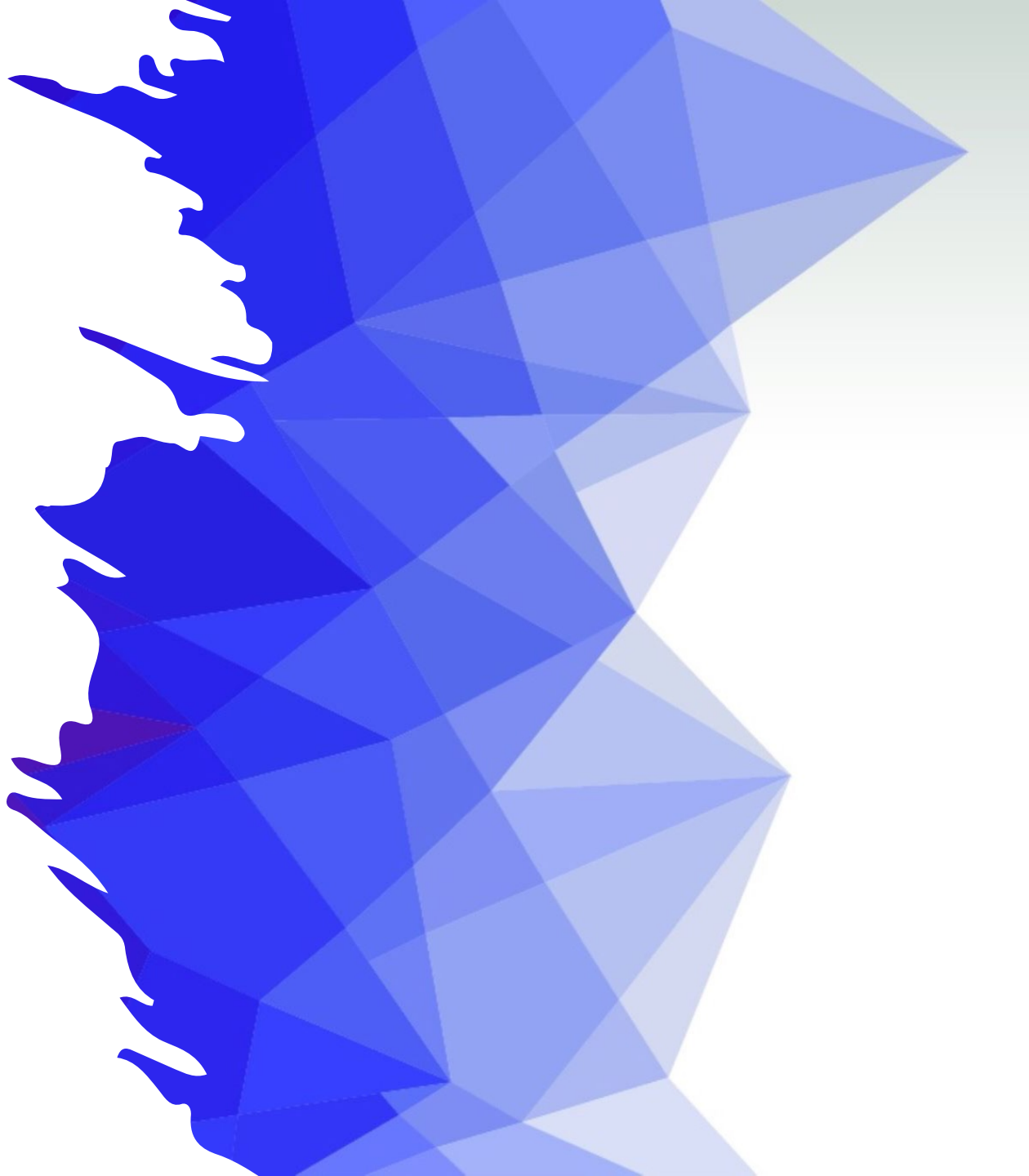
Semantic splitting

- The group_max_sentences parameter controls the maximum number of sentences allowed in each group.
- If a group reaches this limit, a new group is started.
- Additionally, a new group is initiated if the similarity between consecutive sentences falls below a specified similarity_threshold.

Semantic splitting

- In simpler terms, this method organizes a text into clusters of sentences, where sentences within each cluster are considered similar to each other.

- It's useful for identifying coherent and related chunks of information within a larger body of text.



Semantic splitting

- Input =

  "I love dogs. They are amazing. Cats must be the easiest pets around. Tesla robots are advanced now with AI. They will take us to mars.”

- Output =

  ['I love dogs. They are amazing.',

  'Cats must be the easiest pets around.',

  'Tesla robots are advanced now with AI. They will take us to mars.']