AMRITA
VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

RAG

Retrieval-Augmented Generation (RAG) with Large Language Models

Courtesy:, deeplearning.ai , https://arxiv.org/abs/2005.11401

RAG, short for Retrieval-Augmented Generation, helps large language models by giving them access to relevant information during text generation. This allows them to be more accurate and informative, especially for tasks that require real-world knowledge.

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks

Patrick Lewis†‡, **Ethan Perez**⋆,

Aleksandra Piktus†, **Fabio Petroni**†, **Vladimir Karpukhin**†, **Naman Goyal**†, **Heinrich Küttler**†,

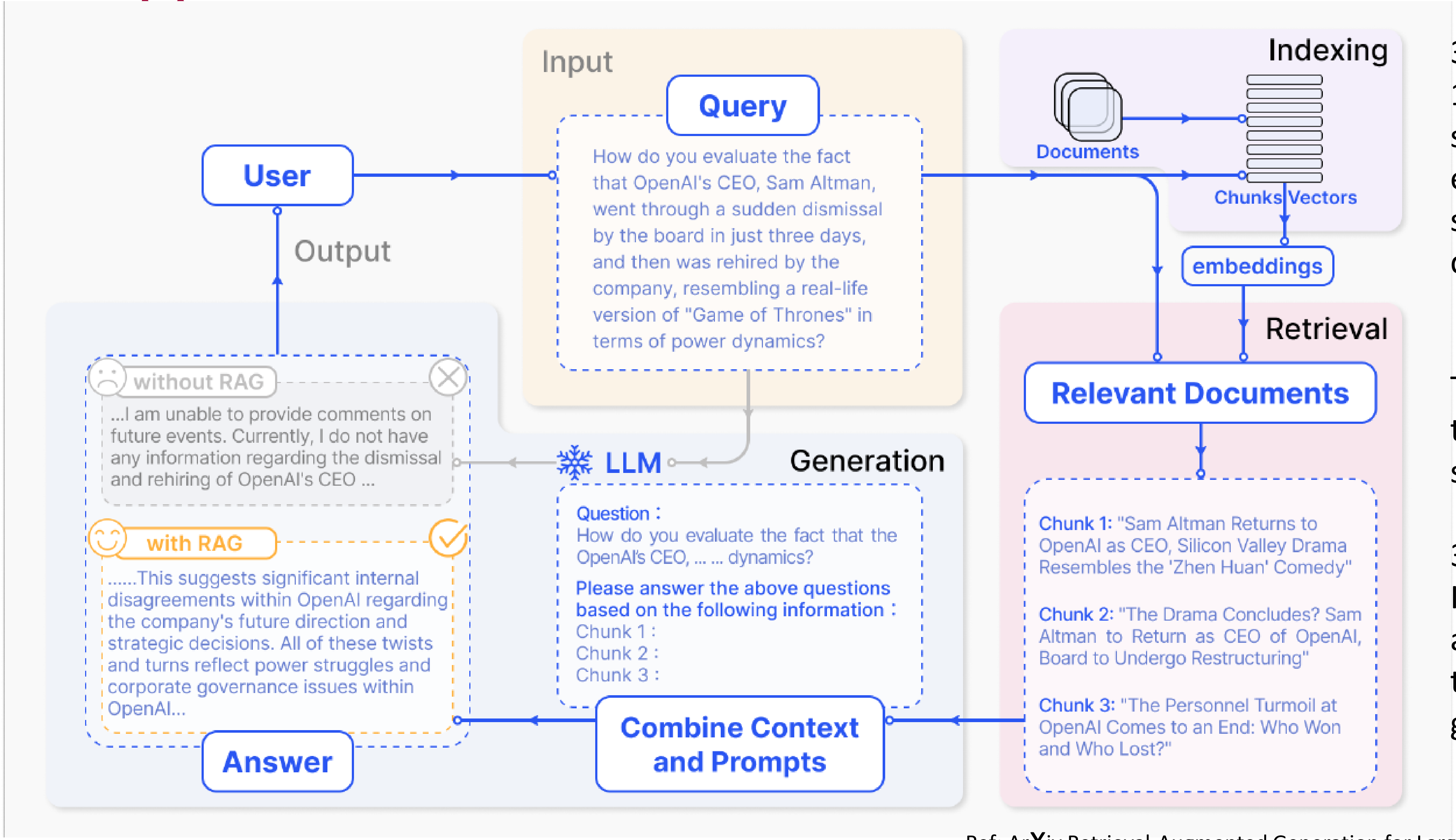
Mike Lewis†, **Wen-tau Yih**†, **Tim Rocktäschel**†‡, **Sebastian Riedel**†‡, **Douwe Kiela**†

†Facebook AI Research; ‡University College London; ⋆New York University;
plewis@fb.com

Abstract

Large pre-trained language models have been shown to store factual knowledge in their parameters, and achieve state-of-the-art results when fine-tuned on downstream NLP tasks. However, their ability to access and precisely manipulate knowledge is still limited, and hence on knowledge-intensive tasks, their performance lags behind task-specific architectures. Additionally, providing provenance for their decisions and updating their world knowledge remain open research problems. Pre-trained models with a differentiable access mechanism to explicit non-parametric memory have so far been only investigated for extractive downstream tasks. We explore a general-purpose fine-tuning recipe for retrieval-augmented generation (RAG) — models which combine pre-trained parametric and non-parametric memory for language generation. We introduce RAG models where the parametric memory is a pre-trained seq2seq model and the non-parametric memory is a dense vector index of Wikipedia, accessed with a pre-trained neural retriever. We compare two RAG formulations, one which conditions on the same retrieved passages across the whole generated sequence, and another which can use different passages per token. We fine-tune and evaluate our models on a wide range of knowledge-intensive NLP tasks and set the state of the art on three open domain QA tasks, outperforming parametric seq2seq models and task-specific retrieve-and-extract architectures. For language generation tasks, we find that RAG models generate more specific, diverse and factual language than a state-of-the-art parametric-only seq2seq baseline.

Application of RAG



3 Steps

1) Indexing. Documents are split into chunks, encoded into vectors, and stored in a vector database.

2) Retrieval. Retrieve the Top k chunks most relevant to the question based on semantic similarity.

3) Generation.
Input the original question and the retrieved chunks together into LLM to generate the final answer

Ref: ArXiv Retrieval-Augmented Generation for Large Language Models: A Survey

Why RAG

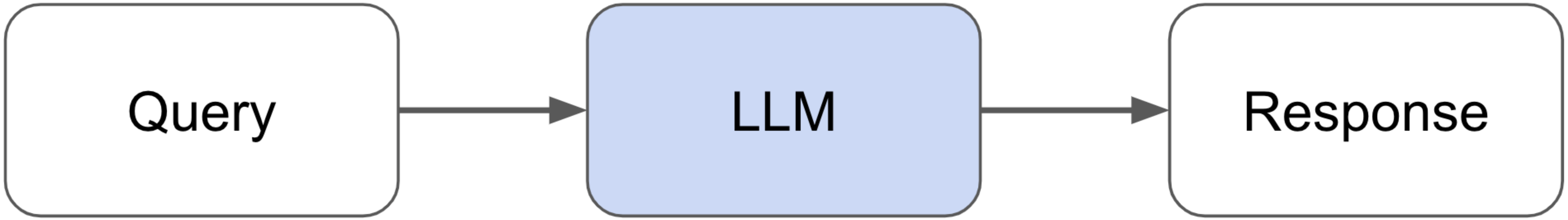
LLMs like GPT, LLaMA, or PaLM store information implicitly in their parameters (e.g., weights of transformer layers) after being trained on massive corpora.

Limitations:

- **Stale Knowledge**: These models cannot access knowledge after their training cutoff (e.g., GPT-3.5 knows nothing after 2021).
- **Hallucination Risk**: When queried about specific facts or less common domains, they may generate plausible but incorrect answers.
- **No Transparency**: The user cannot trace where a generated answer came from.
- **Inefficiency in Updating**: Updating knowledge requires full retraining or fine-tuning, which is costly.

What is RAG?

Large language models (LLMs) have undoubtedly changed the way we interact with information. However, they come with their fair share of limitations as to what we can ask of them. Base LLMs (ex. Llama-2-70b, gpt-4, etc.) **are only aware of the information that they've been trained on** and will fall short when we require them to know information beyond that. **Retrieval augmented generation (RAG)** based LLM applications address this exact issue and **extend the utility of LLMs to our specific data sources.**



RAG- steps

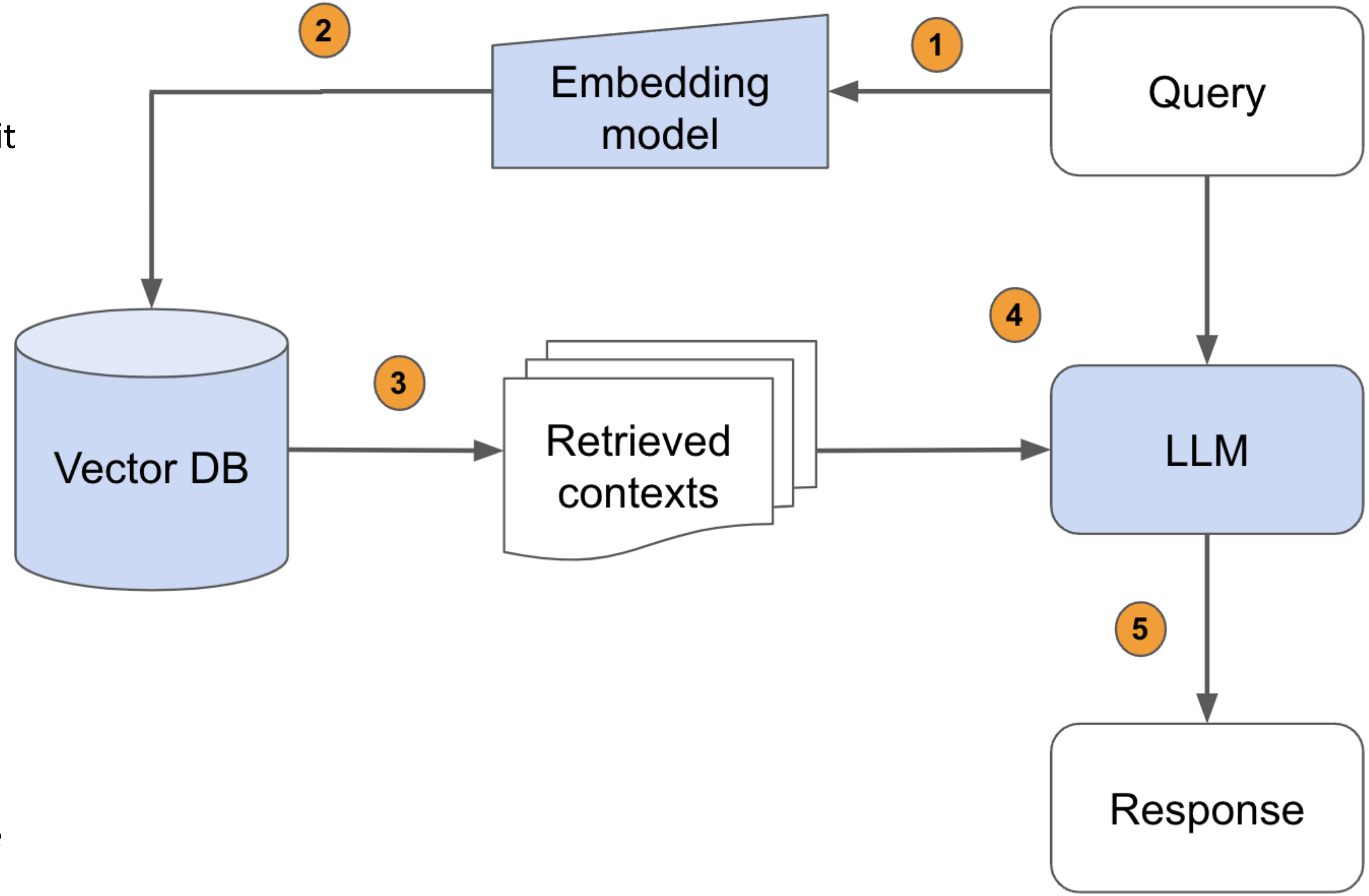
1. Pass the query to the embedding model to semantically represent it as an embedded query vector.

2. Pass the embedded query vector to our vector DB.

3. Retrieve the top-k relevant contexts – measured by distance between the query embedding and all the embedded chunks in our knowledge base.

4. Pass the query text and retrieved context text to our LLM.

5. The LLM will generate a response using the provided content.



Benefits of RAG

Cost-Effective Implementation

- Avoids expensive retraining of foundation models (FMs).
- Enables domain-specific answers using external data.
- Makes generative AI more accessible and adaptable.

Current Information

- Allows LLMs to access **up-to-date sources** (e.g., news, social media).
- Ensures relevance by supplementing outdated model knowledge.
- Ideal for fast-changing domains like finance, health, or policy.

More Developer Control

- Developers can update or switch knowledge sources easily.
- Enables **fine-grained access control** (e.g., by user role).
- Facilitates **debugging** and **custom tuning** for better outputs.

Enhanced User Trust

- Outputs can include **source citations** for transparency.
- Users can verify facts by checking referenced documents.
- Builds confidence in AI-generated responses.

How does RAG work?

1. Create External Data

- External data lies **outside the LLM's original training**.
- Can come from **APIs, databases, PDFs, wikis, internal documents**.
- May exist in **varied formats**: files, tables, long-form text.
- **Embedding models** convert this data into vector form (numerical representation).
- Stored in a **vector database** to build a searchable **knowledge library**.

2. Retrieve Relevant Information

- **User query** is converted into a vector.
- Matched against vectors in the **vector database** using similarity search.
- Returns **contextually relevant documents** (e.g., policies + employee records).
- Relevancy is calculated using **mathematical similarity metrics** (e.g., cosine similarity).

3. Augment the LLM Prompt

- Retrieved data is **injected into the user prompt**.
- Uses **prompt engineering** to format input for the LLM.
- The augmented prompt helps the LLM **generate accurate, grounded answers**.

4. Update External Data

- External data can **go stale**; needs to be refreshed.
- Updates involve:
  - **Refreshing source documents**
  - **Recomputing embeddings**
  - **Reindexing the vector database**
- Can be done via **real-time pipelines** or **batch processing**.
- Requires **data-change tracking and update strategies** from data engineering.

Step 2: Query Sent for Retrieval

The system extracts the query and sends it to the retrieval module.

The retriever may:

Convert the query into an embedding (numerical vector).

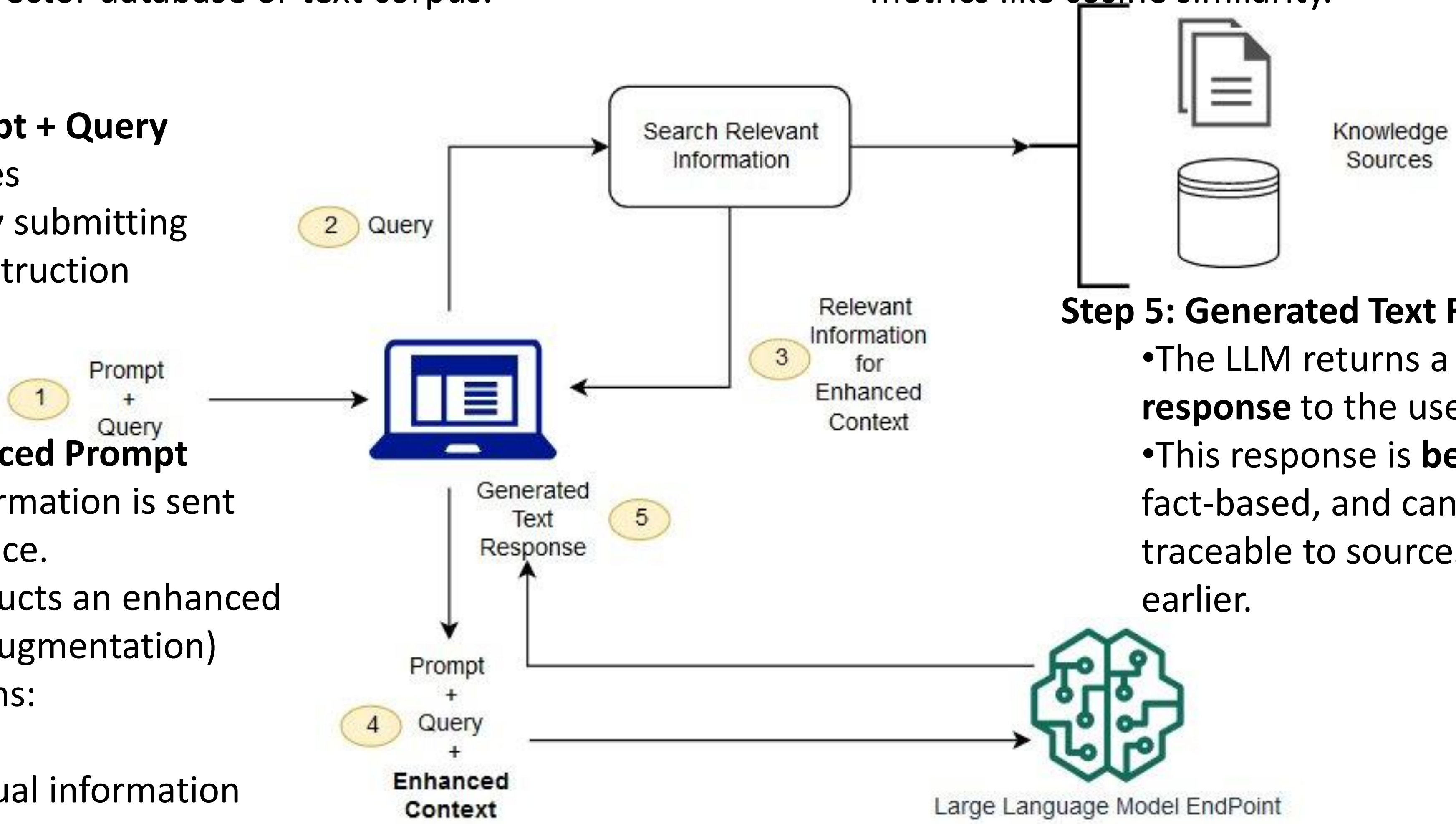
Search against a vector database or text corpus.

Step 3: Retrieve Relevant Information

- The retriever identifies documents, passages, or records **relevant to the query**.
- This is done by computing similarity between the **query vector** and **document vectors** using metrics like cosine similarity.

Step 1: Prompt + Query

A user initiates interaction by submitting a query or instruction (prompt),



Step 4: Create Enhanced Prompt

- The retrieved information is sent back to the interface.
- The system constructs an enhanced prompt (Prompt augmentation) which now contains:
- Original query
- Retrieved contextual information

Step 5: Generated Text Response

- The LLM returns a **final generated response** to the user.
- This response is **better informed**, fact-based, and can include or be traceable to sources retrieved earlier.

The integration of RAG into LLMs

The integration of RAG into LLMs involves two main components: the retriever and the generator.

Retriever: The retriever takes the input query, converts it into a vector using the query encoder, and then finds the most similar document vectors in the corpus. The documents associated with these vectors are then passed to the generator.

Generator : The generator in a RAG-LLM setup is a large transformer model, such as GPT3.5, GPT4, Llama2, Falcon, PaLM, and BERT. The generator takes the input query and the retrieved documents, and generates a response.

Training RAG-LLM Models

Training a RAG-LLM model involves fine-tuning both the retriever and the generator on a question-answering dataset. The retriever is trained to retrieve documents that are relevant to the input query, while the generator is trained to generate accurate responses based on the input query and the retrieved documents.

Drawbacks of Naïve RAG

1.Retrieval Challenges

- Low Precision and Recall: Retrieved chunks may be irrelevant or miss key information.
- Misalignment: Retrieved documents may not align with user intent.
- Single-query Limitation: One-shot retrieval may fail to gather sufficient context for complex queries.

2. Generation Difficulties

- Hallucination: The model may invent facts not grounded in retrieved content.
- Toxicity or Bias: Outputs may reflect harmful or biased language.
- Irrelevance: Generated responses may stray from both the query and the retrieved documents.
- Over-Reliance on Retrieval: The model may simply echo retrieved text without adding synthesis or insight.

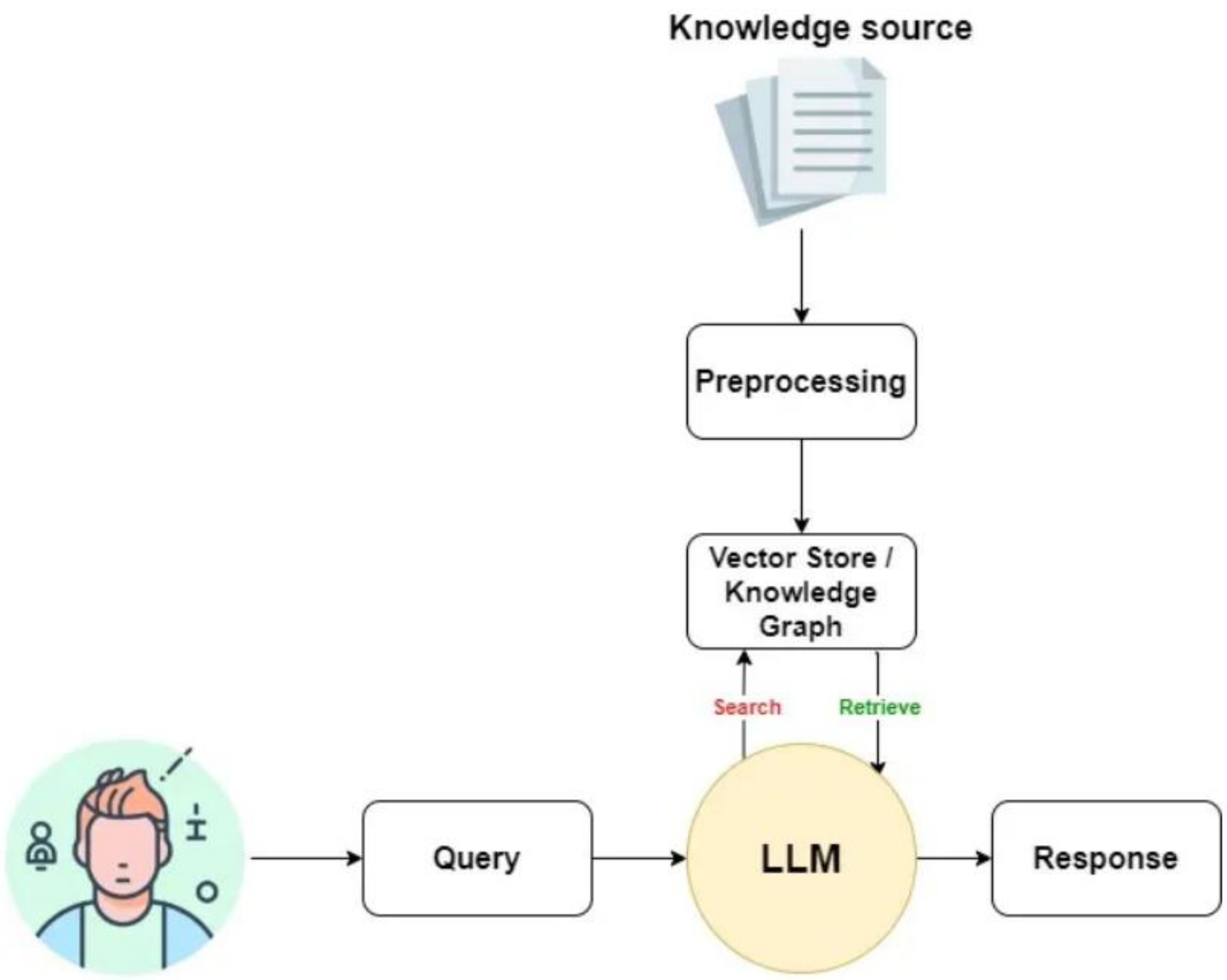
3. Augmentation Hurdles

- Redundancy: Similar or duplicate content from multiple sources can cause repetition.
- Coherence Issues: Difficulty in integrating disparate documents may lead to disjointed responses.
- Stylistic Inconsistency: Challenges in maintaining tone and style when stitching together external content.
- Content Ranking: Determining importance and relevance of retrieved passages is non-trivial.

Functional RAG strategies or architectural enhancements, based on the role, control logic, or LLM coordination mechanism

- Simple RAG
- Self RAG
- Corrective RAG
- Fusion RAG
- Speculative RAG
- Agentic RAG

Simple RAG



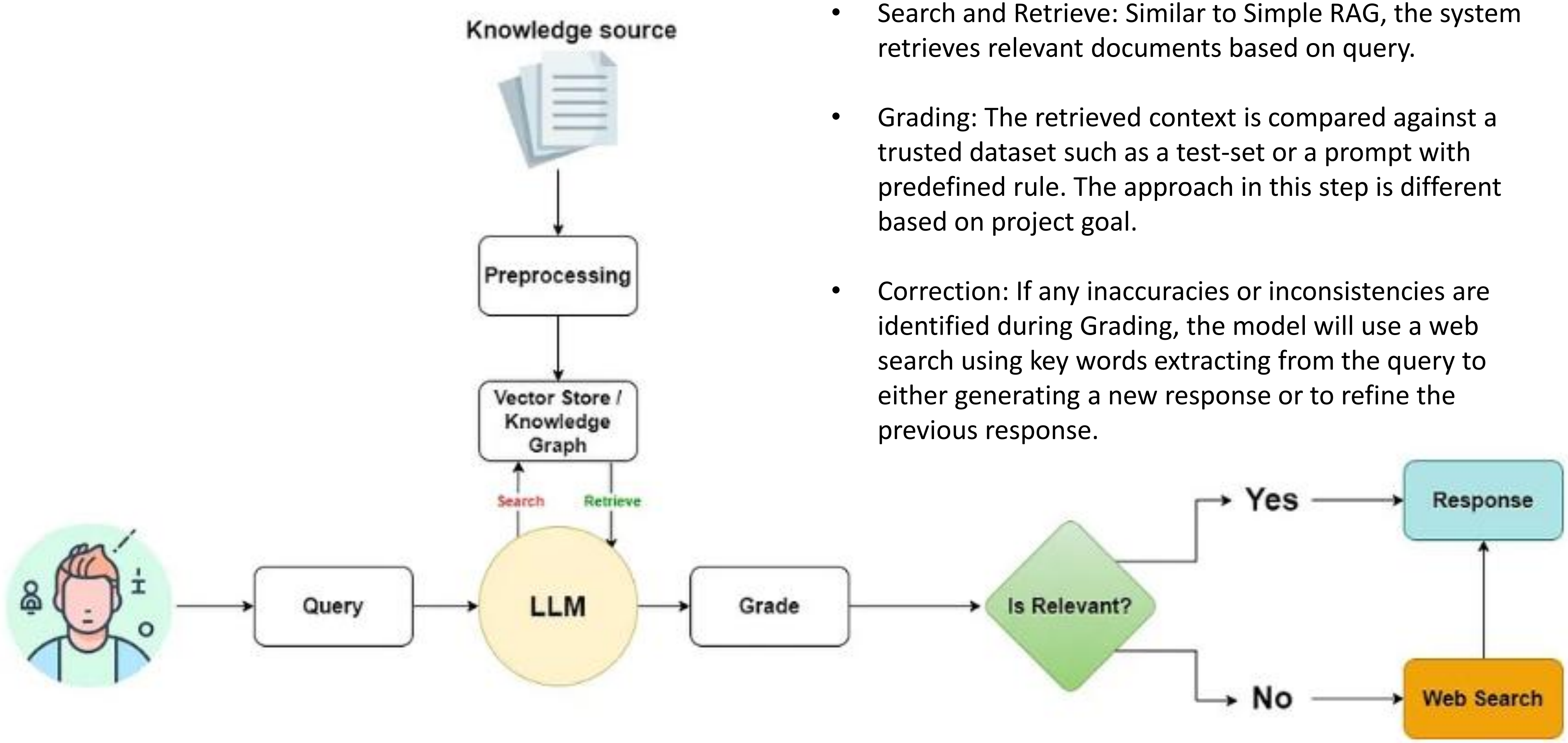
Knowledge source
Preprocessing
Vector Store / Knowledge Graph
Search
Retrieve
Query
LLM
Response

Corrective RAG

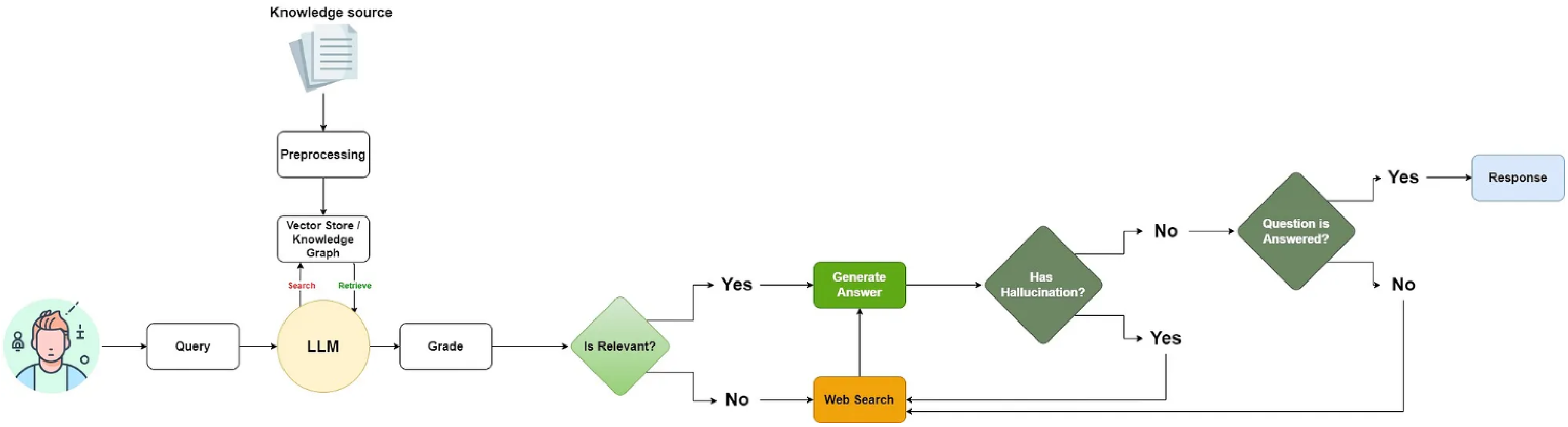
In Corrective RAG, the system not only retrieves and generates responses but also validates and corrects them.

Here's how the process works:

- Search and Retrieve: Similar to Simple RAG, the system retrieves relevant documents based on query.
- Grading: The retrieved context is compared against a trusted dataset such as a test-set or a prompt with predefined rule. The approach in this step is different based on project goal.
- Correction: If any inaccuracies or inconsistencies are identified during Grading, the model will use a web search using key words extracting from the query to either generating a new response or to refine the previous response.



Self RAG



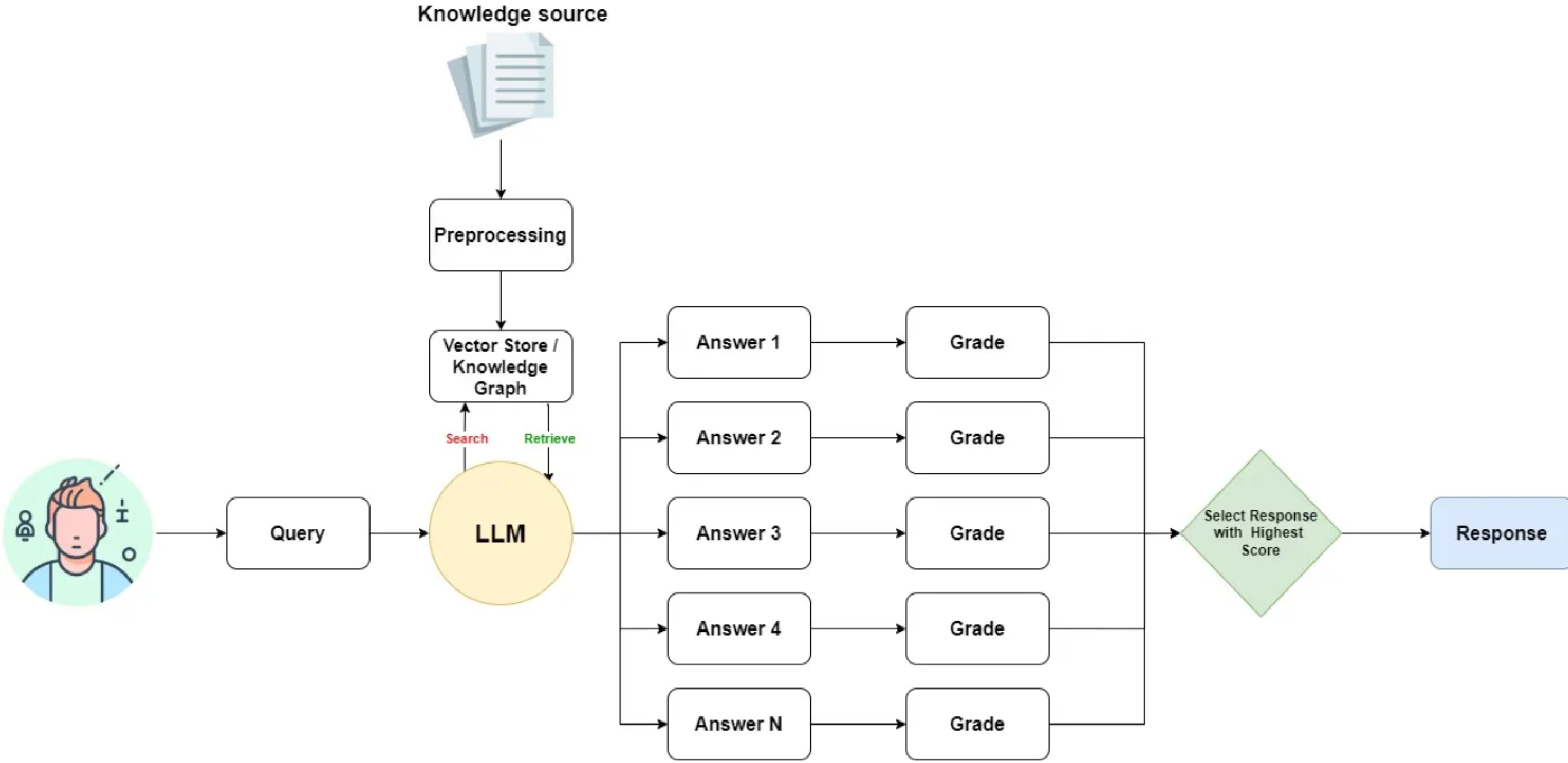
Knowledge source
Preprocessing
Vector Store / Knowledge Graph
Search
Retrieve
Query
LLM
Grade
Is Relevant?
Yes
No
Generate Answer
Web Search
Has Hallucination?
No
Yes
Question is Answered?
Yes
No
Response

SELF RAG

Self RAG improve the quality of RAG results by self-reflection or self-critique.

- Search and Retrieve: The model starts by retrieving relevant information and generating responses based on the input query.
- Grading: To grade or reflect on documents, LLM will critique each answer to know if it is relevant to the query or not. If the document is not relevant then it will use an external source, if it is relevant, it will check the hallucination and accuracy.
- Hallucination: Hallucination node check if the answer supported by document. Sometimes, AI models "hallucinate," meaning they generate answers that sound correct but aren't actually supported by any real data or documents. The hallucination node prevents this by making sure the model's response is backed by the documents it found, ensuring the answer is accurate and reliable. Answer Question: The answer question node check if generated answer, answer the question. It looks at the generated response and checks if it is relevant and complete in answering the original question. If it doesn't, the model can improve or adjust the answer to ensure it's accurate.
- Output: With each iteration, the model produces more accurate and contextually relevant responses. The number of iteration depends on the project scale and available processing power.

Speculative RAG



Speculative RAG

Speculative RAG is an approach where multiple responses are generated for a given query, leveraging a retrieval model to supply relevant information. These responses are then evaluated through a Grading system to choose the most accurate and contextually appropriate one. This method helps handle ambiguity or situations where a query may have multiple interpretations.

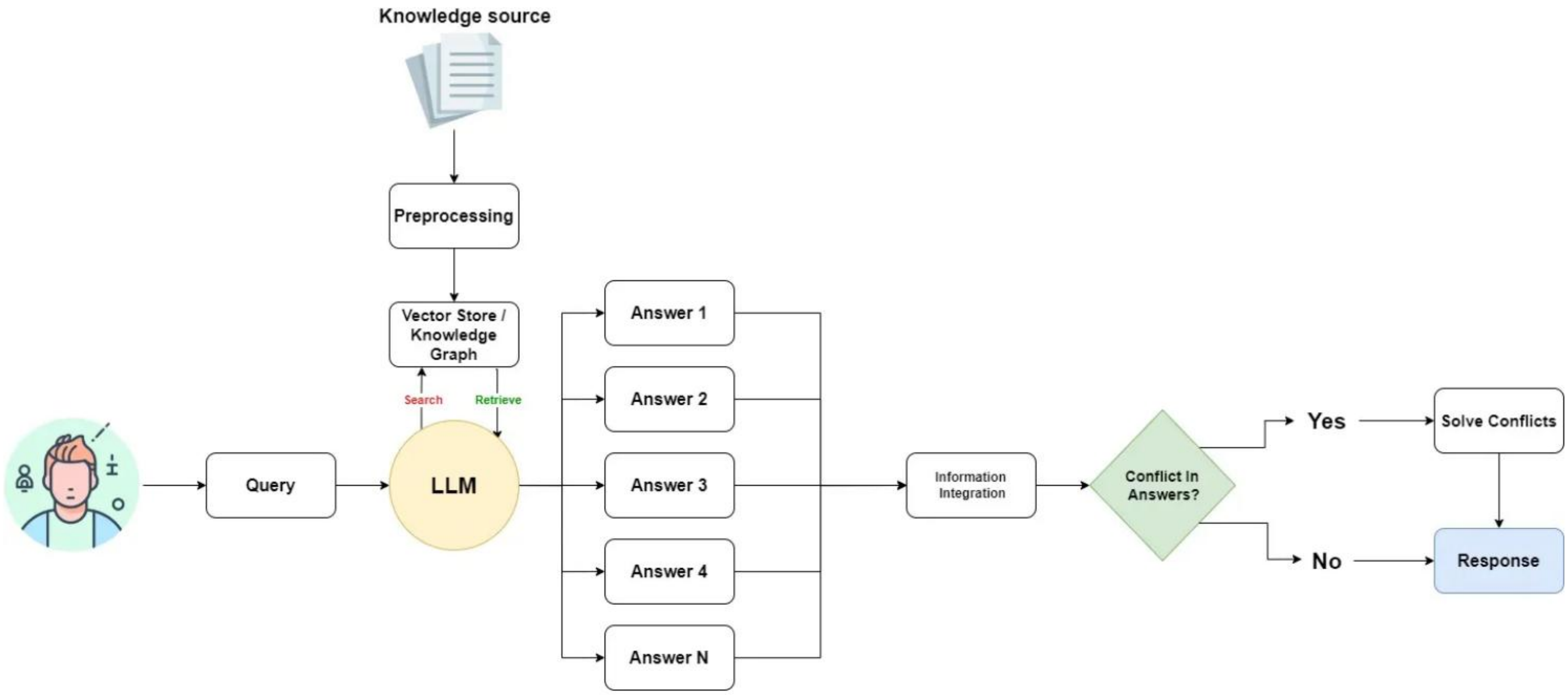
Search and Retrieve: As in Simple RAG, the system retrieves several documents relevant to the query.

Speculation: LLM creates multiple speculative responses from the retrieved documents, exploring various possible outputs instead of just one.

Grading: A grading mechanism evaluates and scores each response based on criteria such as relevance, coherence, completeness, and factual accuracy. This can involve comparing responses with more retrieved documents or using scoring models. Similar to corrective RAG, this step depends on objective and domain of the project.

Selection and Response: The model ranks the responses and chooses the highest-scoring one as the final output.

Fusion RAG



Knowledge source
Preprocessing
Vector Store / Knowledge Graph
Search
Retrieve
Query
LLM
Answer 1
Answer 2
Answer 3
Answer 4
Answer N
Information Integration
Conflict In Answers?
Yes
No
Solve Conflicts
Response

Fusion RAG

Fusion RAG combines information from multiple retrieved sources to create a well-rounded response.

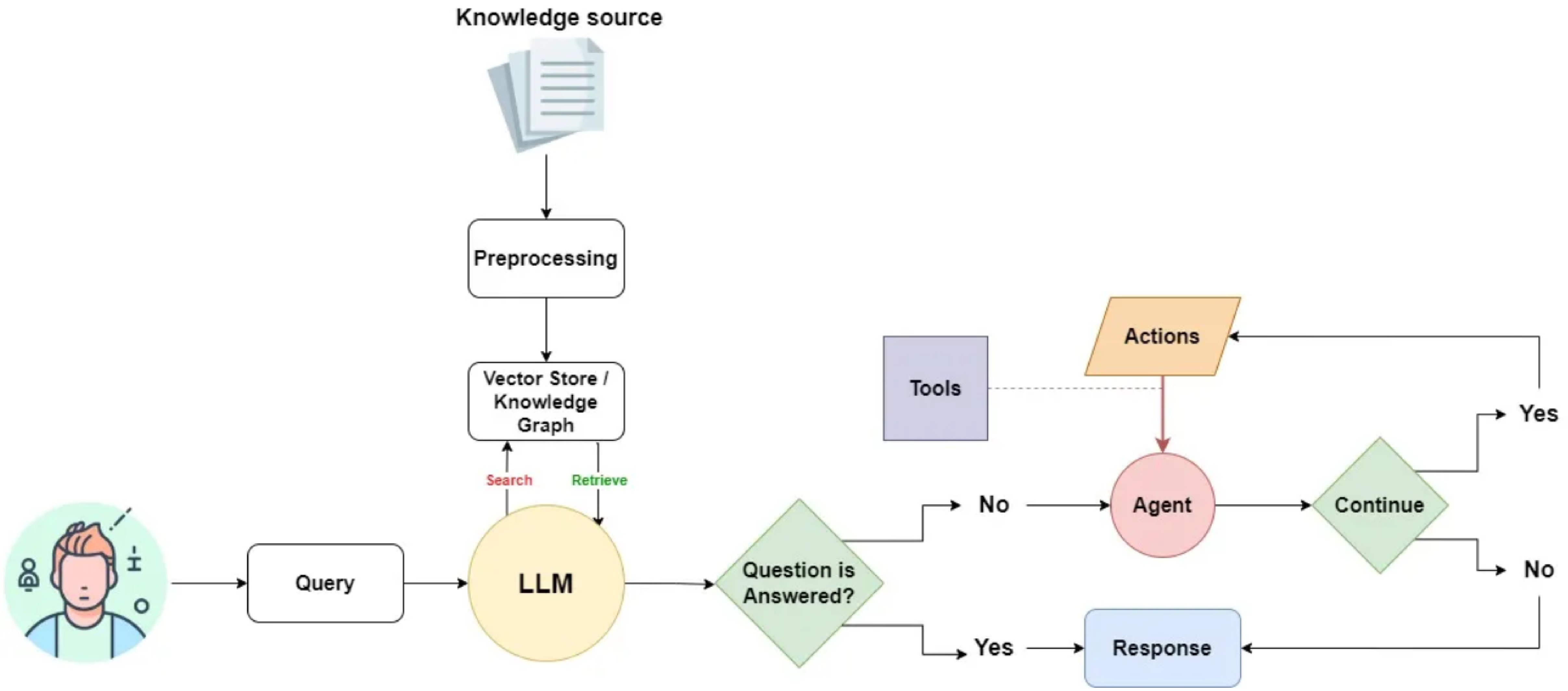
Here's how it works:

Search and Retrieve Diverse Documents: The system retrieves multiple relevant documents, ensuring they represent various perspectives or address different aspects of the query. Each document can be considered one answer to the query.

Information Integration: LLM not only combines documents that are consistent across multiple sources but also takes into account various viewpoints or angles from the different documents and aims to present a response that fairly represents these differing perspectives. Then the model generates a coherent, unified response by combining relevant information from all the retrieved documents, presenting a balanced view based on the evidence.

Conflict Resolution: When there are conflicts, the model resolves them using additional context or predefined rules to ensure consistency in the final answer.

Agentic RAG



Knowledge source
Preprocessing
Vector Store / Knowledge Graph
Search
Retrieve
Query
LLM
Question is Answered?
No
Yes
Tools
Actions
Agent
Continue
Yes
No
Response

RAG vs Fine-tuning

Conceptual Differences

•**Prompt Engineering**:

- Uses the LLM as-is, with **no external data** or model adaptation.
- Lowest effort and infrastructure.

•**RAG**:

- Feeds external, task-relevant knowledge at runtime (like giving a textbook).
- Ideal for **dynamic, information-rich tasks**.

•**Fine-Tuning (FT)**:

- Modifies internal model weights (like training a student over time).
- Best for learning **structure, tone, and style** or task-specific behaviors.

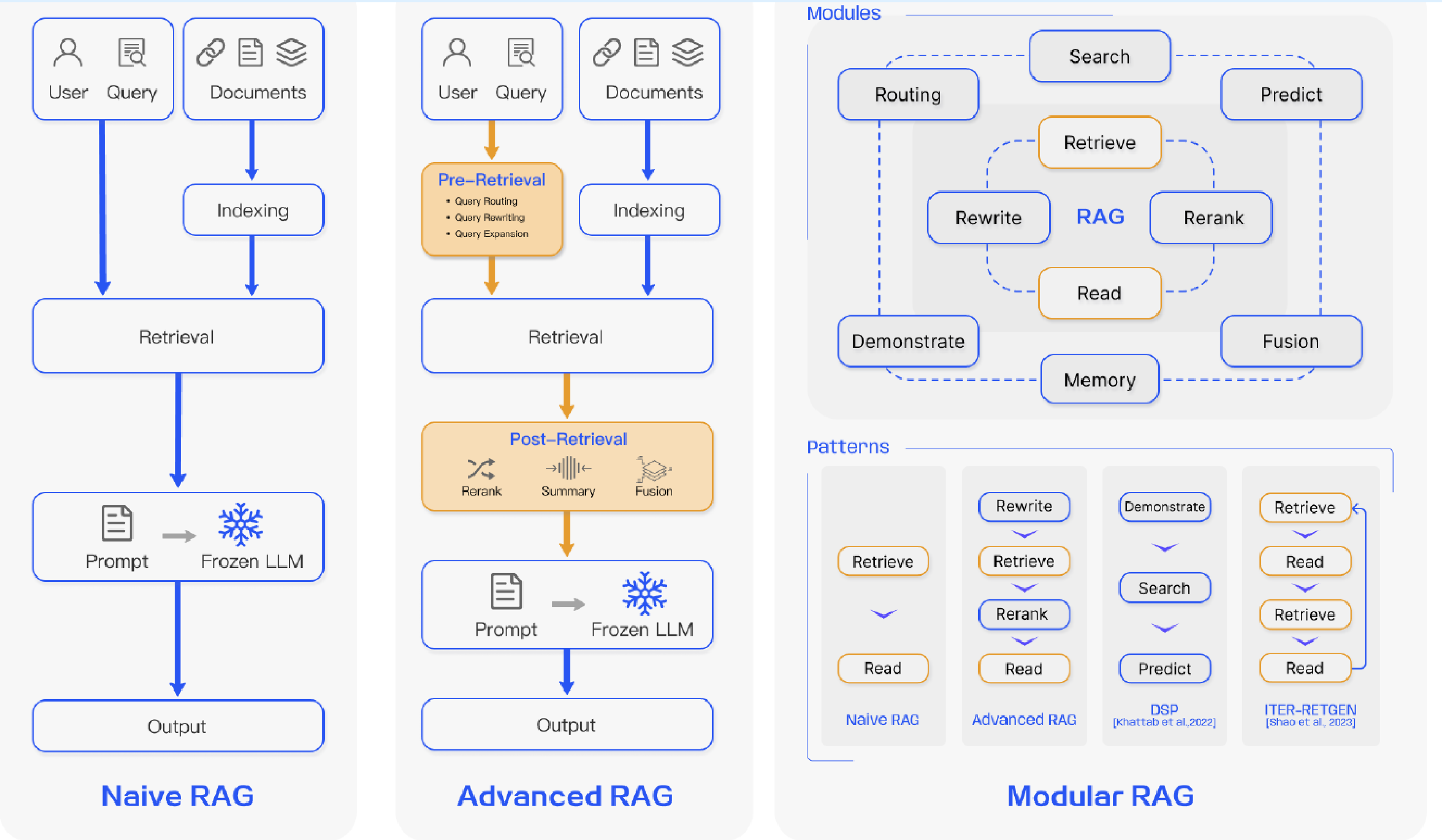
RAG:

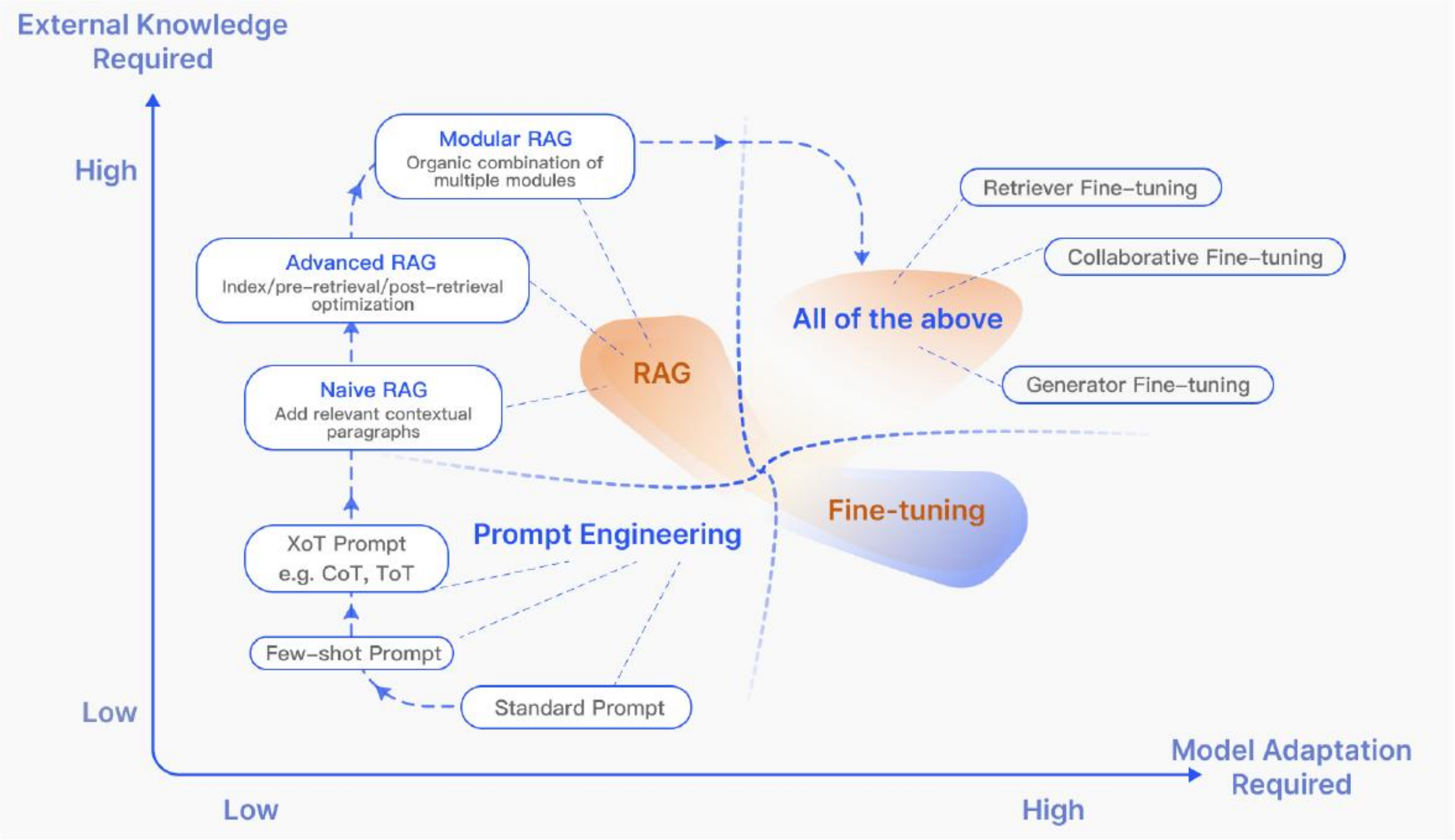
- ✅ Pros:
  - Real-time updates using **external knowledge** (no retraining).
  - Highly **interpretable**: can show sources for answers.
  - **Better performance** on both known and novel knowledge tasks (vs. unsupervised FT).
- ❌ Cons:
  - **Higher latency** due to retrieval + generation.
  - Needs careful **data curation and ethical handling** (e.g., private info).
  - May rely too much on retrieval and not synthesize.

🔧 **Fine-Tuning:**

- ✅ Pros:
  - Deep **customization of model behavior and style**.
  - Reduces hallucination for **known training domains**.
- ❌ Cons:
  - **Requires retraining** for updates (static).
  - **High compute cost** and dataset preparation burden.
  - Poor generalization to **unseen factual data**.

RAG and FT Can be combined iteratively for:Better factual accuracy. Customized tone/behavior. Dynamic knowledge injection + model adaptation.


User
Query
Documents
Indexing
Retrieval
Prompt
Frozen LLM
Output
Naive RAG
User
Query
Documents
Pre-Retrieval
• Query Routing
• Query Rewriting
• Query Expansion
Indexing
Retrieval
Post-Retrieval
Rerank
Summary
Fusion
Prompt
Frozen LLM
Output
Advanced RAG
Modules
Search
Routing
Predict
Retrieve
Rewrite
RAG
Rerank
Read
Demonstrate
Fusion
Memory
Patterns
Retrieve
Read
Naive RAG
Rewrite
Retrieve
Rerank
Read
Advanced RAG
Demonstrate
Search
Predict
DSP
[Khattab et al.,2022]
Retrieve
Read
Retrieve
Read
ITER-RETGEN
[Shao et al., 2023]
Modular RAG

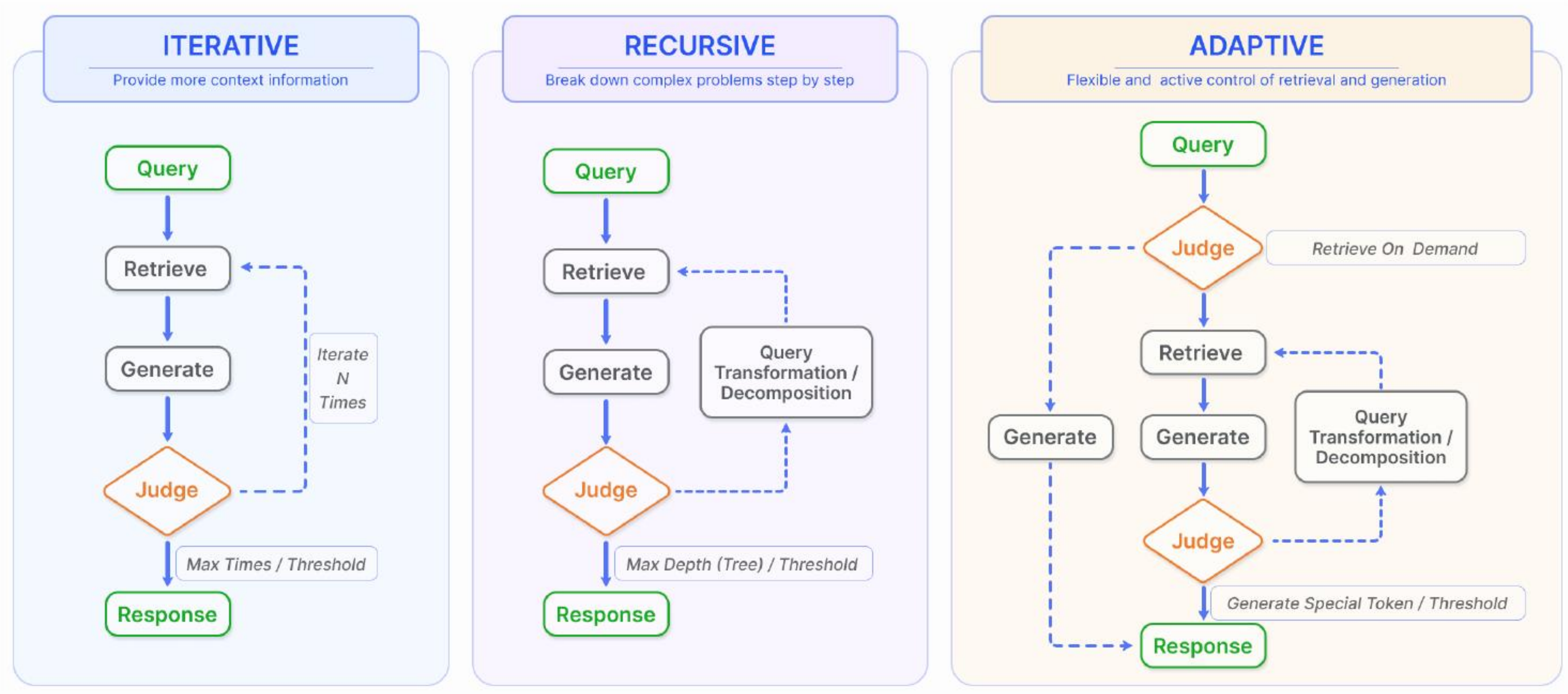


Ref: ArXiv Retrieval-Augmented Generation for Large Language Models: A Survey

Method	External Knowledge	Model Adaptation	Best For
Prompting	❌ Low	❌ Low	Quick testing, reasoning
Naive RAG	✅ Medium	❌ Low	Document QA
Advanced RAG	✅ High	⚠️ Medium	Domain-specific assistants
Fine-tuning	❌ Low	✅ High	Structured output, tone control
Modular RAG	✅ High	✅ Medium/High	Custom hybrid pipelines

- Generation
- Retrieval to expand??

Three types of retrieval augmentation processes



Three types of retrieval augmentation processes

1. **Iterative Retrieval**

- Alternates between retrieval and generation.
- Refines context at each step based on previous outputs.
- Yields more targeted and enriched context over time.

2. **Recursive Retrieval**

- Breaks down complex queries into sub-queries.
- Refines and solves sub-problems iteratively.
- Useful for multi-hop reasoning or layered tasks

.

3. **Adaptive Retrieval**

- RAG system decides dynamically:
- Whether retrieval is needed.
- When to stop retrieving or generating.
- Uses LLM-generated control tokens to manage flow.

Agentic RAG

Agentic RAG involves an AI system operating autonomously with a specific goal, using a retrieval process to make decisions and guide its actions.

Key Workflow Steps:

- **Query Input**
  User provides a clear objective or complex query (e.g., explain, recommend, solve).
- **Search & Retrieve**
  Retrieve from preprocessed knowledge base (e.g., vector store, KG).
- **Initial Answer Check**
  Model self-evaluates:
  ✓ If sufficient → return final response
  ✗ If not → trigger agent actions

🔹 **Agent Action Planning**

- Agent invokes tools/actions (e.g., web search, database queries) based on LLM reasoning (e.g., Chain-of-Thought prompts).

- 🔹 **Iterative Refinement**
  Continually evaluates progress and adjusts retrieval or strategies in real time.
- 🔹 **Final Response**

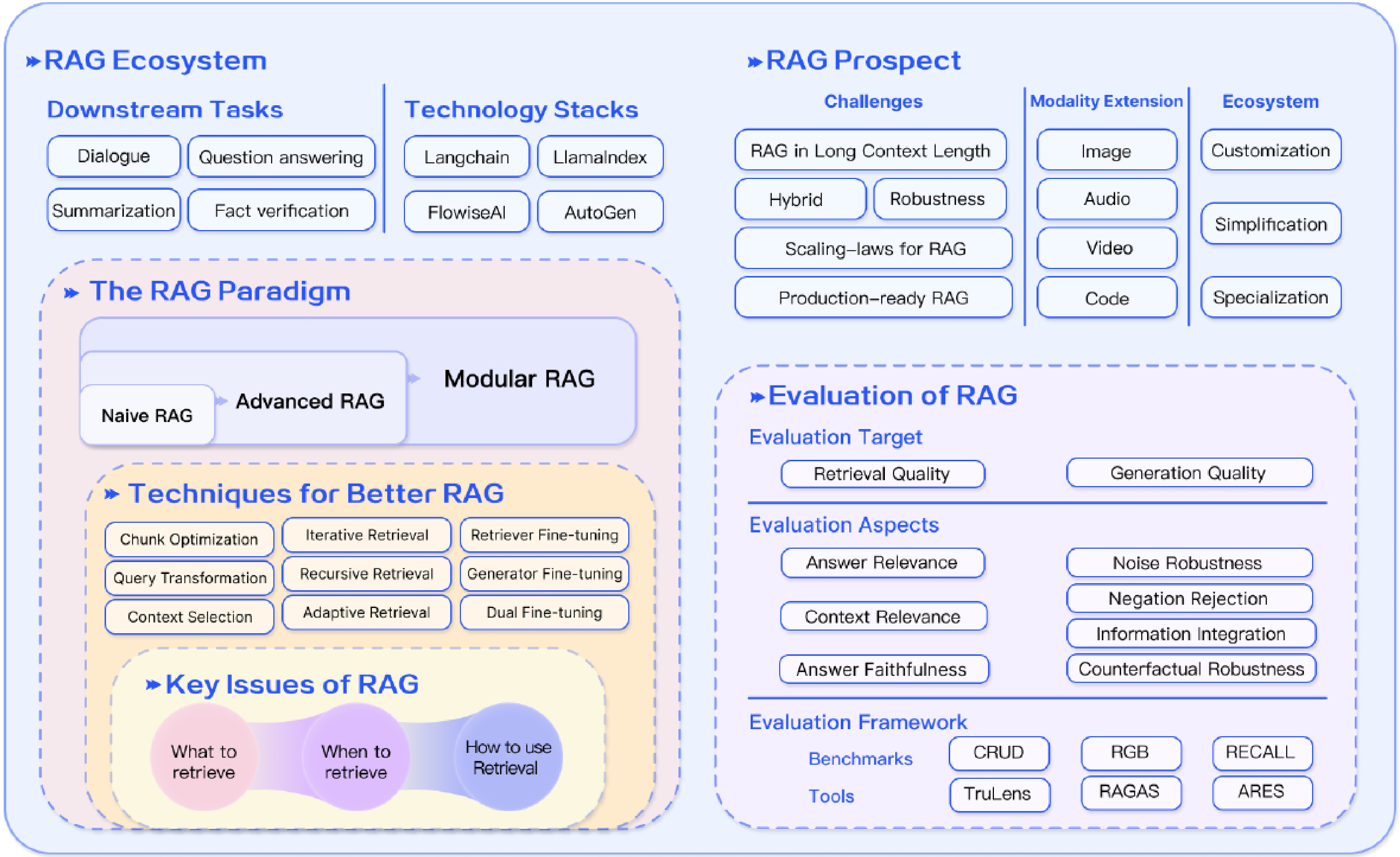
Confirms goal completion and delivers a refined, task-specific output.

SUMMARY OF METRICS APPLICABLE FOR EVALUATION ASPECTS OF RAG

	Context Relevance	Faithfulness	Answer Relevance	Noise Robustness	Negative Rejection	Information Integration	Counterfactual Robustness
Accuracy	✓	✓	✓	✓	✓	✓	✓
EM					✓		
Recall	✓						
Precision	✓			✓			
R-Rate							✓
Cosine Similarity			✓				
Hit Rate	✓						
MRR	✓						
NDCG	✓						
BLEU	✓	✓	✓				
ROUGE/ROUGE-L	✓	✓	✓				

Evaluation Framework	Evaluation Targets	Evaluation Aspects	Quantitative Metrics
RGB†	Retrieval Quality Generation Quality	Noise Robustness Negative Rejection Information Integration Counterfactual Robustness	Accuracy EM Accuracy Accuracy
RECALL†	Generation Quality	Counterfactual Robustness	R-Rate (Reappearance Rate)
RAGAS‡	Retrieval Quality Generation Quality	Context Relevance Faithfulness Answer Relevance	* * Cosine Similarity
ARES‡	Retrieval Quality Generation Quality	Context Relevance Faithfulness Answer Relevance	Accuracy Accuracy Accuracy
TruLens‡	Retrieval Quality Generation Quality	Context Relevance Faithfulness Answer Relevance	* * *
CRUD†	Retrieval Quality Generation Quality	Creative Generation Knowledge-intensive QA Error Correction Summarization	BLEU ROUGE-L BertScore RAGQuestEval

Summary of RAG System



LLM Evaluation Metrics

The field of Artificial Intelligence (AI) has witnessed a paradigm shift with the emergence of Large Language Models(LLMs). These models, trained on massive datasets of text and code, exhibit impressive abilities in generating human-like text, translating languages, writing different kinds of creative content, and answering your questions in an informative way. The capabilities of LLMs have led to their integration into various applications, ranging from content creation and customer service to health care and finance.

Critical gaps persist in LLM evaluation:

- **Hallucinations**: 28% of outputs contain plausible but incorrect information [7]. Mitigation strategies include retrieval-augmented generation (RAG) [23] and prompt engineering [12].
- **Inconsistency**: Outputs vary in 36.4% of repeated queries [4]. Entropy-based stability metrics are proposed to address this [24].
- **Bias**: Financial models exhibit Western-market preferences [2], while medical models lack contextual depth for non-English populations [1].

Contemporary evaluation frameworks can be broadly categorized into three paradigms:

1. **Reference-based evaluation**: Compares model outputs against predefined ground truth [29]
2. **Model-based evaluation**: Uses secondary models to assess quality [30]
3. **Human evaluation**: Incorporates subjective quality assessments [31]

LLM Leaderboards

LLM leaderboards are **curated platforms** that track and compare the performance of different **large language models** across **standard benchmarks** (like MMLU, TruthfulQA, GSM8K, HumanEval, etc.). They use either:

- **Automated metrics** (e.g., accuracy, BLEU, pass@k), or
- **LLM-as-a-Judge** (e.g., GPT-4 judging outputs)

These platforms help:

- Track model progress over time
- Standardize comparisons
- Surface strengths/weaknesses of models
- Assist researchers and users in **choosing or tuning** models

If you're building or fine-tuning an LLM, consider:

- HuggingFace's eval-harness
- OpenAI's evals framework
- LM Eval (used by EleutherAI)
- TruLens + RAGAS for faithfulness + RAG performance
- FastChat + MT-Bench codebase for simulated arena-style evaluations

LLM Leaderboards

Leaderboard	Hosted By	Features	Use Case
Open LLM Leaderboard	HuggingFace + EleutherAI	Tests models on **MMLU, HellaSwag, ARC, TruthfulQA**; includes **self-submission**	Track general reasoning ability
Chatbot Arena	LMSYS (Vicuna team)	Human + GPT-4 judged **battle-style chat** comparisons	Dialogue-level evaluation, helpfulness
HELM (Holistic Evaluation of LMs)	Stanford CRFM	Multi-dimensional eval (accuracy, fairness, robustness, calibration)	Deep audit-like evaluation
MT-Bench	LMSYS	GPT-4-judged **multi-turn dialogue** test	Conversational ability and instruction following
AlpacaEval	Tatsu Lab	Single-turn instruction evaluation with GPT-4 as judge	Helpfulness and clarity
Big-Bench	Google DeepMind et al.	200+ tasks covering math, reasoning, common sense	Academic testing
CodeBench / HumanEval	OpenAI, MBPP, others	pass@k on coding tasks	Code generation models

LLM Evaluation Metrics

- **Accuracy and Completeness**
- **Bias and Fairness**
- **Domain-Specific Metrics**
- **Accuracy and Correctness**
  - Exact Match
  - Accuracy
  - Precision
  - Recall
  - F1 Score
  - Accuracy in Specific Domain
- **Fluency and Coherence**
  - Perplexity
  - BLEU
  - ROUGE
- **Relevance and Informativeness**
  - Relevance Score
  - Informativeness Score
- **Safety and Robustness**
  - Toxicity Score
  - Bias Detection Metrics
  - Robustness Metrics
- **Efficiency**
  - Inference Speed
  - Memory Usage
  - Computational Cost

LLM Evaluation : Accuracy and Completeness

Accuracy

•**Definition**: How correctly the generated response matches the expected output.

•**Methods**:

- **Likert scale (1–6)**: Human raters score outputs on a scale (e.g., 1 = totally wrong, 6 = completely correct).
- **Binary factuality scoring**: Answers are marked as factually correct (1) or incorrect (0).
- **Automated metrics**: Azure AI and other platforms offer built-in NLP pipelines that:
  - Detect factual inconsistencies
  - Highlight hallucinations or unsupported claims

Completeness

•**Definition**: Whether the generated response fully addresses **all aspects** of the user prompt or question.

•Often assessed with:

- **Granular human ratings**
- **Span overlap** in multi-answer QA datasets (e.g., "How many valid points were covered?")
- **Information recall metrics** (in summarization or multi-hop QA)

AlpacaEval and **MT-Bench** are two **LLM evaluation benchmarks** that popularized the use of **LLM-as-a-Judge**, especially in the **era of chat-based instruction-tuned models**.

LLM Evaluation : Bias and Fairness

Bias Detection

•**Demographic skew**: Checking if outputs disproportionately reflect or favor specific groups (e.g., race, gender, religion).

•Tools & datasets:

- **StereoSet**: Measures bias by scoring stereotype, anti-stereotype, and unrelated completions.
- **CrowS-Pairs**: Contains matched sentence pairs to assess racial/gender/age biases.

Fairness Auditing

•**ROI-based fairness audits**: Evaluate bias through **Return on Investment (ROI)** perspective:

- Are certain demographic groups consistently getting worse outcomes?
- Used in applications like education, healthcare, and hiring models.

LLM Evaluation- Accuracy and Correctness

Exact Match :

Measures the percentage of generated outputs that exactly match the ground truth.

Accuracy in Specific Domains:

For domain-specific applications, accuracy is measured against domain-specific benchmarks. For example, in finance, accuracy can be evaluated using financial analysis tasks and in law, using bar exam questions

Accuracy:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

Precision:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Recall:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

F1-Score:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

LLM Evaluation : Fluency and coherence

Fluency and coherence are **subjective qualities** of natural language generation. They reflect how well-structured, grammatically correct, and logically connected the output is.

Perplexity

•**Definition**: A measure of how well a language model predicts a sequence of words.
•**Formula**:

$$\text{Perplexity}(P) = 2^{-\frac{1}{N}\sum_{i-1}^{N}\log_2 P(w_i|w_1,\ldots,w_{i-1})}$$

•**Interpretation**: Lower perplexity means the model is more confident in its predictions—hence better fluency.
•**Use Case**: Model training evaluation, especially for autoregressive models (GPT-like).
•**Limitation**: Requires access to model internals and cannot assess correctness or meaning.

BLEU (Bilingual Evaluation Understudy)

•**Definition**: Measures overlap between generated text and a reference using **n-gram precision**.
•**How it Works**:

- Compares n-grams (1–4 typically) in output vs. reference.
- Applies a brevity penalty to avoid very short outputs scoring high.

•**Use Case**: Machine Translation, sometimes summarization or QA.
•**Limitation**: Ignores word meaning, synonyms, and word order variation.

LLM Evaluation : Fluency and coherence

Fluency and coherence are **subjective qualities** of natural language generation. They reflect how well-structured, grammatically correct, and logically connected the output is.

ROUGE (Recall-Oriented Understudy for Gisting Evaluation)

- **Definition**: Measures overlap between generated and reference text using recall-oriented statistics.
- **Variants**:
  - **ROUGE-1**: unigram overlap
  - **ROUGE-2**: bigram overlap
  - **ROUGE-L**: longest common subsequence (captures sentence-level fluency)
- **Use Case**: **Text summarization** tasks.
- **Limitation**: Still focuses on surface-level matching and doesn't capture semantic fidelity.

BLEU and ROUGE are increasingly being **replaced or complemented by semantic metrics** like:

- **BERTScore**: Measures embedding similarity between reference and output
- **BLEURT / COMET**: Learned metrics trained to match human judgments
- **GPT-based Scorers**: Ask GPT-4 to rate fluency/coherence directly

LLM Evaluation -Relevance and Informativeness

Relevance Score

- **Purpose**: Measures how closely the output answers the **intended question** or **follows the prompt**.
- **How it's computed**:
  - Often via **semantic similarity** (e.g., cosine similarity of embeddings).
  - Can also be human-evaluated using Likert scales.
- **Tools**: Used in frameworks like **RAGAS**, **ARES**, and **TruLens**.
- **Limitations**: Cannot detect hallucination or factual correctness—only surface relevance.

Informativeness Score

- **Purpose**: Measures whether the model **adds meaningful, novel, or non-trivial content** to its response.
- **How it's computed**:
  - Usually via **human judgment**, but recent efforts use **trained LLM judges**.
  - Could also involve scoring unique entities, facts, or answer length adjusted for relevance.
- **Use case**: Summarization, long-form QA, and creative generation.
- **Note**: A response can be relevant but uninformative (e.g., parroting back the prompt).

LLM Evaluation : Safety and Robustness

Toxicity Score

- **Purpose**: Measures **offensiveness**, **hate speech**, or **abusive language**.
- **How it's computed**:
  - **Perspective API**, **RealToxicityPrompts**, and custom **toxicity classifiers**.
- **Metric Output**: Often a score between 0 and 1; higher means more toxic.
- **Limitations**: May be overly sensitive or fail to detect subtle harms (e.g., sarcasm).

Bias Detection Metrics

- **Purpose**: Detect **unfair treatment or stereotype perpetuation**.
- **Types of Bias**:
  - **Gender bias** (e.g., assuming "doctor" is male)
  - **Racial or religious stereotypes**
  - **Socioeconomic bias**
- **Popular Tools**:
  - **StereoSet**: evaluates stereotypical completions.
  - **CrowS-Pairs**: sentence-pair comparisons for bias.
- **Modern Approaches**: Use **LLM self-diagnosis** and **embedding space analysis**.

LLM Evaluation : Safety and Robustness

Robustness Metrics

Purpose: Measure **how stable the model is to adversarial or noisy inputs**.

Examples:

- **Spelling errors**
- **Negation flips**
- **Paraphrase attacks**

Use cases:

- Industrial applications (finance, healthcare)
- Adversarial testing suites (e.g., BIG-Bench hard examples)

Metric Output: Drop in performance (accuracy, BLEU, etc.) under perturbations.

Namah Shivaya