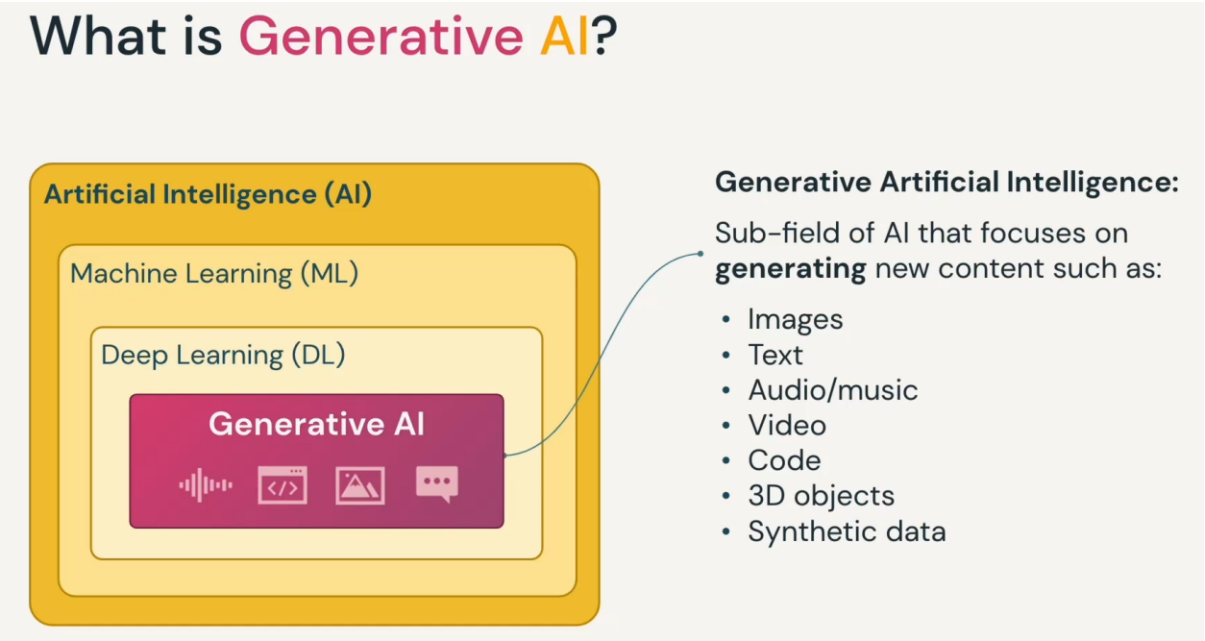

Chatbot Arena Major Models Board

Model	Provider	Major Fund Contributors
Claude	Anthropic	AWS, Google
GPT	OpenAI	Microsoft
Gemini	Google DeepMind	Alphabet
Grok	Elon Musk's XAI	X, Tesla, SpaceX
LLaMA	Meta (Formerly Facebook)	Meta
Mistral	French AI Startup	Lightspeed Venture Partners
Cohere	Cohere Inc.	Coatue Management, Nvidia

Summary Table of LLM Types

Category	Type	Description	Examples
Based on Usage	Open-Source	Publicly available, free to use and modify	BERT, GPT-Neo, Hugging Face Models
	Proprietary	Owned models, access typically requires a license	GPT-3, Claude, Bard
Based on Training Methods	Pre-trained	Trained on large datasets for general representation. Next word generation	BERT, T5, DBRX Base

	Instruction-Based	Designed to follow user instructions and answer question	InstructGPT, ChatGPT, DBRX Instruct
Based on Algorithms	Encoder-Only	Focuses on understanding input text	BERT, RoBERTa
	Decoder-Only	Generates text based on previous tokens	GPT-2, GPT-3
	Encoder-Decoder	Processes input and generates output	T5, BART

Comparison between Foundational Models and Fine-Tuned Models in tabular format:

Aspect	**Foundational Model**	**Fine-Tuned Model**
Definition	A large, pre-trained model built on vast amounts of data for general purposes.	A model that has been further trained on specific data for a specialized task.
Training Data	Trained on broad, diverse datasets (e.g., web text, books, articles).	Trained on a specific dataset related to a particular domain or task.
Use Case	General-purpose; can be applied to various downstream tasks without modification.	Optimized for specific tasks (e.g., sentiment analysis, customer support).
Adaptability	High adaptability to different tasks, but with varying performance.	Highly specialized to perform well on a particular task or domain.
Performance	Moderate performance on domain-specific tasks without adaptation.	High performance on domain-specific tasks due to task-specific training.
Training Effort	Requires extensive computational resources and time to train from scratch.	Requires less computational resources for fine-tuning (compared to training from scratch).
Customization	Cannot perform specific tasks optimally without further fine-tuning.	Custom-tailored to solve specific problems or tasks.
Model Size	Typically, very large in terms of parameters (e.g., GPT-3, BERT).	Smaller model size possible after fine-tuning for a specific task.
Examples	GPT, BERT, T5, DALL-E.	GPT fine-tuned for customer support, BERT fine-tuned for sentiment analysis.
Generalization	Strong generalization across different domains.	Better at domain-specific or task-specific generalization.
Training Complexity	Complex, requires vast amounts of data and extensive resources.	Simpler training process after foundational model training.

Inference Time	Potentially slower due to larger model size.	Can be faster if optimized or distilled for specific tasks.

Conclusion:

- Foundational models are broad and versatile, serving as the base model for a wide range of tasks.
- Fine-tuned models are task-specific versions of foundational models, offering higher accuracy for particular use cases.

Key Prompt Engineering input parameters:

Parameter	**Description**	**Range/Value**	**Effect**
Temperature	Controls the randomness of the output. Higher values result in more random responses, while lower values make it focused.	0.0 to 1.0	Low temperature (e.g., 0.2) produces deterministic, focused responses, while high (e.g., 0.8) results in more diversity.
Top-p (Nucleus Sampling)	Limits sampling to the smallest set of tokens whose cumulative probability exceeds p.	0.0 to 1.0	Lower values (e.g., 0.2) result in highly focused output, while higher values (e.g., 0.9) allow for more diverse choices.
Top-k	Considers only the top k most probable next tokens, where k is a user-defined number.	Integer (e.g., 1, 5, 50)	Smaller values (e.g., 1 or 5) make output deterministic, while larger values (e.g., 50) increase diversity.
Max Tokens	The maximum number of tokens (words or characters) to be generated in a response.	Integer (e.g., 50, 1000)	Controls the length of the output, where a higher number results in longer responses.
Presence Penalty	Encourages the model to talk about new topics by penalizing repetition of already used words.	-2.0 to 2.0	Higher values (e.g., 2.0) penalize frequent repetition, leading to more diverse topics.
Frequency Penalty	Reduces the likelihood of repeating the same token within the same response.	-2.0 to 2.0	Higher values decrease repetition within the text, promoting novelty.
Stop Sequences	A sequence of tokens that, if generated, will end the generation process.	User-defined strings (e.g., [“END”])	Defines where the model should stop generating more tokens.

Logit Bias	A bias applied to specific tokens, making them more or less likely to appear in the generated text.	Integer (e.g., positive or negative values)	Adjusts the probability of certain words appearing in the response, useful for specific token manipulation.

These parameters allow for fine-tuning the behavior of Generative AI models, helping to control the creativity, focus, and determinism of the outputs.

Vector Databases

Vector databases are specialized systems designed to store, index, and query high-dimensional vector embeddings, which are numerical representations of data (such as text, images, or audio). They enable efficient **vector similarity search** by comparing these embeddings, making them crucial for applications involving similarity, clustering, and ranking tasks.

Use Cases of Vector Databases

1. **RAG (Retrieval-Augmented Generation)**:
    - Supports **Generative AI** by retrieving relevant documents or information based on query embeddings to generate or augment the content.
2. **Recommendation Engines**:
    - Provides personalized recommendations by comparing user profiles and item embeddings to suggest similar items based on past preferences.
3. **Similarity Search**:
    - Used to find items (e.g., documents, products) that are most similar to a given query in terms of semantics or features. Commonly used in e-commerce and document retrieval systems.

Vector Similarity

- **Distance Metrics**:
    1. **Euclidean Distance**:
        - Measures the straight-line distance between two points in a vector space.
        - Example: Used in clustering and anomaly detection.
    2. **Manhattan Distance**:
        - Measures the distance between two points along axes at right angles (sum of absolute differences).
        - Example: Used in grid-like data (e.g., image pixels).

- **Similarity Metrics**:
    1. **Cosine Similarity**:
        - Measures the cosine of the angle between two vectors. Ranges from -1 (opposite) to 1 (similar).
        - Example: Common in document and text similarity search.

Vector Search Strategies

- **KNN (K-Nearest Neighbors)**:
    - Searches for the closest vectors to a given query by comparing distances in the vector space.
- **ANN (Approximate Nearest Neighbor)**:
    - Speeds up KNN searches by approximating nearest neighbors, trading off some accuracy for faster retrieval times.
    - **Tree-Based (Annoy by Spotify)**:
        - Builds multiple trees that partition the vector space, allowing fast approximate searches.
    - **Proximity Graphs (HNSE)**:
        - Constructs graphs where vectors are nodes connected to their nearest neighbors, enabling efficient search traversal.
    - **Clustering (FISS by Facebook)**:
        - Clusters similar vectors together, reducing the search space and improving efficiency.
    - **Hashing (LSH)**:
        - Uses hash functions to map similar vectors to the same "buckets," simplifying search.
    - **Vector Compression**:
        - Reduces the size of vector embeddings to speed up searches without sacrificing too much accuracy.
        - **SCaNN by Google**: Leverages quantization and clustering for fast search.
        - **Product Quantization (PQ)**: Compresses vectors into smaller chunks, useful for large-scale vector search.

Pros and Cons of Vector Databases

Pros	Cons

Efficient high-dimensional vector search	May require specialized hardware for large datasets
Supports similarity search, ranking, and clustering	Approximate methods may trade accuracy for performance
Enables semantic understanding and complex queries	Requires fine-tuning of indexing and search parameters

---

Reranker in Retrieval

A **reranker** is used after an initial search to reorder the retrieved items based on more sophisticated criteria or models, enhancing the relevance of the results.

- **Benefits**:
  - Improves the quality of search results by applying a deeper analysis of candidates.
  - Can use more computationally expensive techniques after narrowing down the initial set.
- **Challenges**:
  - Increased computational cost compared to the initial search.
  - Requires careful balance between reranking depth and real-time performance.

Chunking strategies used in **Generative AI** for document or data extraction, focusing on how data can be split into manageable chunks and embedded for storage in a vector store.

Chunking Strategy	Description	Advantages	Use Case
Context-Aware Chunking	Chunk by sentence, paragraph, section, or punctuation to preserve semantic context	- Retains context and meaning - Suitable for long documents	Ideal for use cases where the context of sentences/paragraphs needs to be preserved. Examples: article summarization, FAQ systems
Fixed-Size Chunking	Divide the document into chunks of a fixed size based on the number of tokens (e.g., 512 tokens)	- Ensures each chunk fits within model constraints - Simple implementation	Good for models with strict token limits like GPT, which require chunking to avoid overflow.
Chunk by Sentence	Split the document into individual sentences	- Provides fine-grained control - Suitable for brief text	Useful when short, specific answers are required. Example: chatbots or sentence classification

Chunk by Paragraph	Break the document into paragraphs	- Maintains the logical flow - Balances between detail and size	Ideal for structured documents like legal contracts or research papers where paragraphs are coherent units
Chunk Overlap	Overlapping between chunks to preserve context between consecutive chunks	- Reduces loss of meaning at chunk boundaries - Ensures smoother transitions between chunks	Used when handling larger context windows is essential (e.g., when summarizing or answering questions about long texts)
Window Summarization	Create a rolling window to summarize chunks by overlapping a few tokens	- Allows continuous analysis - Reduces context loss	Useful for tasks requiring long-range dependencies and continuous input. Example: video transcription summarization
Summarization with Metadata	Summarize chunks and include metadata such as timestamps, headings, or references	- Improves retrieval accuracy - Adds contextual clues for better query matching	Ideal for research databases, document storage systems where metadata provides critical search and retrieval insights. Example: legal case summaries

Use Cases:

1. **Document Summarization**:
    - **Context-aware chunking** is useful when summarizing large articles, ensuring that sentences or paragraphs are chunked in meaningful units for better embedding.
2. **Question Answering Systems**:
    - **Fixed-size chunking** allows precise control over how much data is fed into the model, optimizing performance for QA systems by limiting the input size.
3. **Text Classification**:
    - **Chunking by sentence** is ideal for systems where each sentence's meaning and classification is needed, such as spam detection or sentiment analysis.
4. **Long-form Content Retrieval**:
    - **Chunk Overlap** is effective in handling content like books or long articles where context from previous chunks is essential for accurate retrieval.
5. **Window Summarization**:
    - Especially useful in summarizing continuous streams of data (e.g., video transcriptions, meeting recordings).

6. **Metadata-Based Summarization**:
    - Suitable for professional systems like legal databases, where metadata such as authorship, section headers, and timestamps help improve document retrieval quality.

Notes:

- **Chunking is crucial** when working with large documents and LLMs because of token limits and performance considerations.
- The chunking strategy should align with the **model constraints** (e.g., token limits, memory) and the **use case goal** (e.g., retrieval, summarization, QA).

Retrieval-Augmented Generation (RAG)

RAG is a hybrid architecture that combines **retrieval-based** methods with **generative models** to enhance the quality, accuracy, and relevance of generated text. It is widely used in applications such as question answering, summarization, and chatbots, where retrieving relevant information from a knowledge base enhances the generative process.

Main Concepts of RAG Workflow:

1. **Index & Embed**:
    - **Index**: The process of creating an index of the knowledge base, documents, or external data that the model will retrieve from.
    - **Embed**: Convert chunks of text into **vector embeddings** using an embedding model (e.g., BERT, GPT). Each chunk is transformed into a numerical representation that can be searched using similarity metrics.

Purpose: Provides a structured, searchable format for external data that the model will retrieve during generation.

2. **Vector Store**:
    - A **database** that stores vector embeddings of indexed content. This is where all the documents or knowledge chunks are stored as vectors. Examples include vector databases like FAISS, Pinecone, Weaviate, etc.

Purpose: Efficiently store and retrieve vector embeddings for fast and accurate similarity search.

3. **Retrieval**:
    - Given a query, the model retrieves the most relevant embeddings from the vector store based on similarity metrics like **cosine similarity** or **Euclidean distance**.

Purpose: To find and bring relevant chunks of information that can inform or guide the generative model.

4. **Filtering & Reranking**:
    - **Filtering**: Removing extraneous, irrelevant, or noisy results after retrieval to ensure higher quality data for generation.

- **Reranking**: Reordering the retrieved results to prioritize the most relevant or contextually important information.

Purpose: Refine retrieved results to ensure the generative model uses the best information available, improving response accuracy.

5. **Prompt Augmentation**:
    - The retrieved context is appended or embedded into the input prompt provided to the generative model. This gives the generative model additional context or knowledge to improve the quality of the generated output.

Purpose: Strengthens the generative model's response by incorporating external knowledge retrieved from the vector store.

6. **Generation**:
    - The generative model (e.g., GPT, BERT, T5) processes the augmented prompt (query + retrieved context) and produces the final response or output.

Purpose: Generate a natural language output that answers the query or task using the combined information from the input and retrieved context.

Benefits of RAG Architecture:

- **Improved Accuracy**: Combines generative power with external data retrieval, leading to more accurate responses, especially for knowledge-intensive tasks.
- **Dynamic Knowledge Updates**: Unlike traditional models that rely on static training data, RAG can fetch the latest information from a live knowledge base.
- **Efficient Handling of Long Documents**: By retrieving only the most relevant chunks, RAG allows for efficient interaction with long documents or large datasets.
- **Reduced Hallucination**: Incorporating external knowledge retrieval minimizes hallucination, where generative models might invent false facts.
- **Customizability**: You can modify the retrieval mechanism, use different knowledge bases, or adjust the generative model to fit specific business use cases.

Why is Data Preparation Important for RAG?

1. **Poor Quality Model Output**:
    - If the knowledge base or documents used for retrieval contain **noisy or irrelevant data**, the generative output will also degrade in quality.
    - High-quality data must be curated to ensure retrieval enhances rather than hampers the model.

2. **"Lost in the Middle"**: ("Needle in haystack test")
    - This refers to a scenario where important information is missed during retrieval because of poor chunking or document structure.
    - Effective **chunking strategies** are essential to ensure relevant context is available to the model.
3. **Inefficient Retrieval**:
    - If the vector store is poorly optimized or if the chunking is inconsistent, it could lead to slow or inaccurate retrieval of knowledge, affecting the performance of the RAG pipeline.
    - Efficient retrieval is key for real-time or near-real-time applications.
4. **Exposing Sensitive Data**:
    - Poor data preparation can lead to unintended leakage of sensitive or confidential information.
    - Ensuring **data governance** and privacy are critical when using external data sources for retrieval.
5. **Wrong Embedding Model**:
    - If the wrong embedding model is used for document encoding (e.g., one not trained on the domain-specific data), the retrieved documents may be irrelevant to the query.
    - Choosing an embedding model trained on domain-relevant data improves retrieval accuracy.

Additional Important Details:

- **Context Precision**: Ensuring that the retrieved information is highly relevant to the query.
- **Context Relevance**: The degree to which the retrieved context helps inform the generated response.
- **Context Recall**: Ensures that important parts of the document are retrieved.
- **Answer Faithfulness**: Ensures the generated output accurately reflects the retrieved information, reducing hallucination.
- **Answer Relevance**: The relevance of the generated response to the original query.

Incorporating these metrics during evaluation helps assess the quality and effectiveness of RAG systems.

RAG vs Traditional Models:

- Traditional models generate responses purely based on their internal knowledge.

- RAG models enhance their response by dynamically retrieving relevant external information, offering more grounded and up-to-date outputs.

This approach is highly valuable for tasks like **customer support**, **document summarization**, and **dynamic question answering**, where up-to-date, accurate, and specific responses are crucial.

RAG (Retrieval-Augmented Generation) evaluation metrics assess the quality of a model's output based on how well it retrieves and generates responses in context. These metrics are often grouped into categories like query relevance, context relevance, response accuracy, and faithfulness. Here's a breakdown of the key metrics:

RAG Evaluation Metrics

Metric	Definition	Evaluation Focus
Context Precision	Measures the proportion of retrieved documents (contexts) that are relevant to the query.	**Retrieval Accuracy**
Context Recall	Measures the proportion of relevant documents (contexts) retrieved out of all possible relevant documents.	**Retrieval Coverage**
Context Relevance	Measures how relevant the retrieved context is to the query. Typically evaluated by human judgment or ranking algorithms.	**Context Quality**
Answer Relevance	Measures the quality of the generated response relative to the query and retrieved context. It assesses whether the response is relevant to the user's question.	**Generated Response Quality**
Context Faithfulness	Assesses whether the generated response is accurate and grounded in the retrieved context (i.e., not hallucinating or generating unsupported claims).	**Truthfulness & Faithfulness**
Answer Fluency	Evaluates how well the generated response is written or communicated, focusing on grammatical correctness and smoothness.	**Language Quality**
Ground Truth Accuracy	Compares the generated response to a ground truth or reference answer, evaluating how closely they match.	**Overall Response Accuracy**
Answer Informativeness	Measures whether the response provides sufficient information to answer the query.	**Depth of Information**
Semantic Overlap	Measures how semantically similar the generated response is to the ground truth, using metrics like cosine similarity or BLEU.	**Semantic Similarity**
Context Coverage	Assesses whether the response utilizes all the relevant information present in the context, avoiding omissions.	**Comprehensive Utilization**

Faithfulness to Query	Assesses if the response generated aligns faithfully to the original intent of the query.	**Intent Preservation**
Hallucination Rate	Measures the proportion of information in the generated response that does not correspond to the retrieved context or the ground truth.	**Error Minimization**
Ground Truth Recall	Evaluates the proportion of key points from the ground truth captured by the generated response.	**Recall of Key Information**
Contextual Relevance	Measures the relationship between retrieved context and the response, ensuring the response is based on context.	**Context-Response Coherence**
Answer Correctness	Evaluates the factual correctness of the response, especially in domains requiring precise answers (e.g., medical or legal).	**Correctness**

Detailed Descriptions

- **Context Precision**: Measures how many of the retrieved documents are truly relevant to the query. High precision means that most retrieved contexts are relevant and useful.
- **Context Recall**: Measures how many relevant contexts were retrieved out of all possible relevant contexts. High recall ensures that the retrieval system didn't miss important contexts.
- **Context Relevance**: Evaluates whether the retrieved context is useful and makes sense given the query. Low relevance means the retrieved documents do not help generate a good answer.
- **Context Faithfulness**: Ensures that the generated answer is grounded in the provided context, meaning it doesn’t "hallucinate" details not found in the context. Faithfulness is crucial for factual correctness.
- **Answer Relevance**: Measures how well the generated response answers the query. Even if the response is correct, it should also address the query’s specific needs.
- **Answer Fluency**: Looks at how natural and grammatically correct the generated response is, ensuring it’s coherent and easy to understand.
- **Ground Truth Accuracy**: Compares the generated response with the expected (ground truth) answer. This could use metrics like BLEU, ROUGE, or semantic similarity.
- **Answer Informativeness**: Assesses whether the response is informative enough to satisfy the query’s intent, rather than providing vague or incomplete answers.
- **Semantic Overlap**: Focuses on how semantically similar the generated response is to the ground truth, beyond simple word-matching (using cosine similarity or word embeddings).
- **Faithfulness to Query**: Ensures the response doesn't drift from the original intent of the query. A response can be factually correct but irrelevant to the query.

- **Hallucination Rate**: Important in RAG applications to measure how often the model generates unsupported or false claims. Lower hallucination rates are critical for trustworthiness.
- **Ground Truth Recall**: Looks at how many key points from the ground truth are included in the generated answer, focusing on completeness.
- **Contextual Relevance**: Ensures the generated response is not only correct but also based on the retrieved context, thus showing coherence between retrieval and generation stages.
- **Answer Correctness**: Looks at whether the answer is factually accurate, especially in knowledge-intensive applications where wrong information can have serious consequences.

Additional Metrics in Context of RAG:

- **BLEU (Bilingual Evaluation Understudy Score)**: Measures the similarity of generated text to a reference text using n-gram overlap. Useful for evaluating the closeness of generated answers to human-provided ground truths.
- **ROUGE (Recall-Oriented Understudy for Gisting Evaluation)**: Focuses on recall and how much of the reference text's content (n-grams, word sequences) is captured by the generated response.

These metrics help evaluate the **retrieval** phase (retrieving relevant documents) and the **generation** phase (producing coherent, relevant, and accurate responses) in RAG systems, ensuring overall high-quality AI responses.

Summary of Use Cases:

Metric	Primary Use	Common Tasks	Evaluation Focus
BLEU	**Language Translation**, Text Generation	Machine translation, dialogue systems	Precision (n-gram match)
ROUGE	**Text Summarization**, Document Generation	Text summarization, machine translation	Recall (content capture)

BLEU vs. ROUGE:

- **BLEU** is better suited for tasks where **word-for-word precision** is crucial, such as machine translation.
- **ROUGE** is more useful for tasks like **summarization**, where it's important to capture the **overall meaning** and important information rather than exact word matches.

Prompt Safety & Guardrails (e.g., Llama Guard)

- **Prompt Safety**: Ensuring user inputs (prompts) do not generate harmful, biased, or unsafe responses.
- **Guardrails**: Techniques or systems that restrict the behavior of AI models, ensuring that their outputs are aligned with ethical standards and safety requirements.

- **Llama Guard**: A framework or approach for adding guardrails to large language models (LLMs), specifically LLaMA-based models.
- **Use Cases**: Preventing harmful outputs, detecting bias, and enforcing constraints on content generation.
- **Methods**:
  - Pre-filtering prompts.
  - Post-filtering responses.
  - Setting rules or constraints for outputs.

AI Security Risks

- **Adversarial Attacks**: Input manipulation to cause models to make incorrect predictions.
- **Model Inversion Attacks**: Attackers can infer training data by querying the model multiple times.
- **Data Poisoning**: Maliciously crafted data used during training to manipulate model outputs.
- **Output Manipulation**: Generating harmful content (e.g., biased, toxic, or inappropriate outputs).
- **Privacy Leakage**: Models unintentionally memorizing and revealing sensitive information from the training data.

Base Foundation Model Metrics

- **Loss**:
  - A measure of how well a model's predictions match the true values during training.
  - Commonly used loss functions for LLMs include **Cross-Entropy Loss**.
  - **Limitation**: It's an average metric and may not indicate specific failures or biases in generated content.
- **Perplexity**:
  - A metric used to evaluate language models by measuring how well a model predicts a test set.
  - Lower perplexity indicates better performance in predicting sequences of words.
  - **Limitation**: It doesn't measure the quality, coherence, or safety of generated text.
- **Toxicity**:
  - Measures how often a model generates harmful or offensive content.
  - Can be evaluated using pre-trained classifiers like **Perspective API** or **OpenAI's Moderation Tools**.

    - **Limitation**: Difficulty in defining universally accepted thresholds for toxic content. Can miss subtle forms of bias.

Limitations of These Metrics

- **Loss** doesn't capture ethical considerations like bias or safety in outputs.
- **Perplexity** evaluates how well a model predicts text, but doesn't measure the correctness or appropriateness of the responses.
- **Toxicity** detection tools may not capture nuanced or context-sensitive toxic outputs.
- Metrics alone cannot fully assess whether a model is producing fair, useful, or safe outputs in real-world applications.

Addressing Evaluation Limitations (LLM-as-a-Judge)

- LLMs alone cannot be solely trusted to evaluate themselves due to inherent biases and limitations.
- Human judgment is often required for final assessment.

LLM-as-a-Judge Basics

- In this approach, the LLM itself is used to evaluate the quality of generated responses.
- The model is prompted to judge the relevance, coherence, and factual correctness of outputs.
- **Limitations**:
    - LLMs can reinforce their own biases.
    - Lack of external validation from real-world users or evaluators.

LLM-as-a-Judge Best Practices

- **Diversity in Evaluation**: Use multiple LLMs for evaluations to reduce bias.
- **Human-in-the-loop**: Include human reviewers to validate LLM-based judgments.
- **Adversarial Testing**: Use adversarial prompts and queries to test the robustness of the model's evaluations.
- **Grounding Models**: Incorporate factual grounding and ensure the LLM has access to accurate information to enhance reliability.
- **Ethical Guidelines**: Embed ethical constraints in the evaluation process to avoid biased or harmful judgments.

Conclusion:

Prompt safety, evaluation of AI security risks, and effective use of metrics (Loss, Perplexity, and Toxicity) are essential to ensuring high-quality outputs from LLMs. However, no single metric can fully capture the complexities of language generation, which is why advanced frameworks like **LLM-as-a-Judge** and guardrails such as **Llama Guard** are crucial to handling these challenges.

In Databricks, batch inference refers to the process of making predictions on a batch of data using a pre-trained machine learning model. You can use SQL via ai_query() to perform batch inference in Databricks, which is part of their integration with machine learning models and SQL interfaces.

Example: Batch Inference using SQL in Databricks

Let's assume you have trained a machine learning model (e.g., a regression or classification model) and registered it in the Databricks Model Registry. You can perform batch inference by calling this model in SQL via the ai_query() function.

Here's a basic template on how to perform batch inference using SQL:

1. **Train and Register Model**: Ensure that your model is trained, saved, and registered in Databricks. Suppose it's registered under model_name.
2. **Batch Inference using SQL**: You can perform batch inference on a dataset using SQL with the ai_query() function to call the model.

*SELECT *,*

ai_query('model_name',

named_struct('feature1', feature1_column, 'feature2', feature2_column, 'feature3', feature3_column)) AS prediction

FROM your_input_data_table;

Key Points:

- **Model Registration**: Make sure the model is registered in the Databricks Model Registry and is available to be called by ai_query().
- **Input Structure**: The named_struct should match the expected input feature names of the model.
- **Inference Result**: The result of the inference is added as a new column in your SQL query, alongside the original data.

This way, you can easily scale predictions across large datasets using SQL in Databricks!

AI Agent Overview

What is an Agent?

An agent in AI is an autonomous entity capable of perceiving its environment through sensors and acting upon it to achieve specific goals. Agents can make decisions based on data input, user requests, or predefined goals and adapt their behavior to optimize performance over time.

How Does an Agent Decide Which Actions to Take?

Agents decide actions using algorithms or policies that map perceived inputs to actions. The decision-making process can involve rule-based systems, reinforcement learning, planning, or

reasoning. Agents analyze the environment and then calculate the best action to achieve their objectives, either reactively or through long-term planning.

Agent Reasoning

Agent reasoning refers to the internal processes through which an agent evaluates possible actions and makes decisions. It can involve simple heuristics, logic-based systems, or complex machine learning models. Reasoning systems are used to:

- Interpret observations or data from the environment.
- Infer potential outcomes of actions.
- Make predictions based on past interactions or predefined rules.

Agent Design Patterns

Design patterns define commonly used frameworks for building AI agents.

- **ReAct**: Combines reasoning (thinking) and acting. Agents use a loop where they evaluate options, reason, and act accordingly. ReAct can be useful in dynamic environments where the agent needs to adapt constantly.
- **Tool Use/Function Calling**: AI agents can use external tools or call functions to extend their capabilities. This allows agents to query databases, interact with APIs, or perform calculations in real-time. OpenAI's function-calling feature is a good example.
- **Planning**: Planning involves agents reasoning over a set of future actions. AI agents with planning capabilities can develop strategies or sequences of actions to reach long-term goals, such as in robotics and logistics.
- **Multi-Agent Collaboration**: In multi-agent systems, agents collaborate and communicate with each other to achieve shared goals. This pattern is useful for distributed systems or complex problem-solving scenarios like game simulations and automated trading.

Tools to Build Agents

- **LangChain Agents**: A powerful tool for building language model-based agents. LangChain allows developers to create complex agents that can perform multi-step tasks, interact with APIs, and leverage various data sources.
- **AutoGPT**: An open-source tool that enables agents to be self-directed. Agents can autonomously plan, execute tasks, and interact with their environment, making it useful for advanced automation scenarios.
- **OpenAI Function Calling**: This allows GPT models to interact with external tools or APIs by calling functions. It expands the agent's capabilities by enabling it to gather real-time information or execute operations not natively supported by the model.
- **AutoGen**: A framework designed to automate the generation and optimization of AI models, including agents. It can manage various aspects of model tuning and deployment, simplifying the creation of agents.

- **Transformers Agents**: Built on the Transformers architecture, these agents can handle tasks ranging from natural language processing to multimodal interactions, combining text, images, and more.

Multi-Model Applications and Architecture

Multi-model applications involve using more than one model or agent to solve complex tasks. These applications benefit from the combined strengths of different models, such as using a language model for text interpretation and an image model for visual data.

In **Multi-Model Architecture**, different models are integrated, allowing for specialized task handling. For example, an AI system might use a combination of GPT for text analysis, vision models for image recognition, and reinforcement learning agents for action planning.

Agent Design in Databricks Using Chatbricks

In Databricks, **Chatbricks** can be used to design and build AI agents tailored for specific use cases like data analysis, automation, or interactive workflows. By integrating the power of Databricks' big data platform with agent frameworks, businesses can create agents that not only respond to queries but can also execute actions like running data pipelines or generating reports.

In the context of **Databricks** and **machine learning (ML)** workflows, "drift" typically refers to **model drift** or **data drift**. These are concepts that occur when there is a change in data patterns, distributions, or model behavior over time, potentially causing the performance of a machine learning model to degrade. Drift detection and mitigation are important aspects of maintaining and monitoring deployed ML models.

Types of Drift:

1. **Data Drift**:
    - Data drift occurs when the distribution of input data changes over time compared to the data the model was trained on. This can happen due to evolving real-world conditions, external factors, or shifts in how data is collected.
    - **Example**: A model trained on customer transaction data may start to perform poorly if new customer behaviors or trends emerge (e.g., different purchasing habits after a holiday season).
2. **Concept Drift**:
    - Concept drift happens when the relationship between the input data and the target variable (the concept the model is predicting) changes. This could lead to the model making inaccurate predictions as the underlying data generating process has shifted.
    - **Example**: A fraud detection model may start failing if fraudsters adapt their methods and the patterns of fraud behavior change.
3. **Model Drift**:

- Model drift refers to the phenomenon where the performance of the deployed model deteriorates over time, even though the input data might not have changed significantly. This could be due to changes in external conditions that influence model outcomes.
- **Example**: In a recommendation system, a model may start recommending outdated products as customer preferences shift.

Drift in Databricks

In **Databricks**, drift can be monitored, detected, and addressed using various features:

1. **Monitoring with MLflow**:
   - Databricks integrates tightly with **MLflow**, which can be used to track model performance and log predictions and metrics over time. This makes it easier to detect when model or data drift occurs.
   - MLflow allows tracking of key metrics and model performance in real-time, which can be compared over different periods to identify drift.
2. **Data Quality Monitoring**:
   - In Databricks, you can implement **data quality checks** using tools like **Delta Live Tables** or **Databricks SQL** to monitor incoming data for changes in distribution or quality. By setting alerts or dashboards to monitor data pipelines, drift can be detected early.
3. **Model Retraining Pipelines**:
   - With Databricks' orchestration tools like **Jobs**, **Delta Lake**, and **MLflow**, you can build automatic pipelines to retrain your model if data drift is detected.
   - **AutoML** and **CI/CD pipelines** can be used to continuously deploy and retrain models, helping to mitigate drift when new data arrives.
4. **Drift Detection Techniques**:
   - Techniques like **statistical tests** (e.g., KS test, Chi-square test), monitoring **model residuals**, or analyzing **prediction intervals** can be used in Databricks notebooks to identify drift in a model's performance or in the incoming data.
   - **Feature drift detection** can be done by comparing historical feature distributions with the current data using visualization or statistical tests in Databricks notebooks.

Example of Managing Drift in Databricks:

1. **Tracking Performance Metrics with MLflow**: After deploying a model in Databricks, you can use **MLflow** to log key performance metrics (e.g., accuracy, AUC-ROC, precision, etc.) at regular intervals. If the metrics degrade over time, it may indicate model drift.
   - **Step**: Set up monitoring in Databricks to log predictions and actual outcomes over time and compare the metrics to the model's original performance.

2. **Data Drift Detection**: You can use statistical tests or machine learning approaches to detect when the input data distribution has changed. This could involve:
    - Regularly computing feature statistics (mean, variance, distribution shape).
    - Setting up monitoring dashboards to visualize shifts in data distribution over time.
3. **Automated Retraining Pipeline**: Using Databricks **Jobs** and **Delta Lake**, you can set up an automated pipeline that triggers model retraining when drift is detected. The pipeline can monitor incoming data, detect drift, and retrain the model if necessary, ensuring that the model stays up to date.

Conclusion:

In Databricks, **drift** refers to shifts in the data or model behavior that can affect the performance of machine learning models. Databricks provides tools like **MLflow**, **Delta Lake**, and orchestration features that can help monitor, detect, and address both **data drift** and **concept drift** through model monitoring, data quality checks, and automated model retraining pipelines.

Various Model and their usage

- **Audio Models**: Whisper, DeepSpeech, Wav2Vec 2.0, and Tacotron are widely used for tasks involving speech recognition, transcription, and synthesis.
- **Vision Models**: YOLO, ResNet, ViT, and StyleGAN dominate in object detection, classification, and image generation tasks.
- **Language Models**: GPT, BERT, and T5 focus on text generation, understanding, and summarization.
- **Multi-modal Models**: CLIP, DALL·E, and Stable Diffusion work across both **text** and **vision** domains, handling tasks such as image generation from text or aligning images and text.

Databricks and Anthropic Models

- **Claude** models, developed by Anthropic, focus on **conversational AI** with an emphasis on safety and steering language generation.
- **Databricks Dolly** is fine-tuned for **enterprise applications**, leveraging Databricks' cloud platform to provide business use cases for AI.
- **Databricks Lakehouse AI** offers models specifically designed for **enterprise-level AI** and **machine learning**, integrated with the **Lakehouse** architecture for handling large-scale data.

Here's a list of models and frameworks designed for reading and extracting tabular data from **PDFs**, **images**, or **scanned documents**. These models utilize a combination of **OCR (Optical Character Recognition)** and **deep learning** techniques for parsing structured data like tables.

Model/Framework	Primary Usage	Domain
TabNet	Interpretable deep learning model for tabular data	**Tabular Data**
Camelot	Extracting tables from PDFs	**PDF/Table Extraction**
pdfplumber	Parsing and extracting tables and text from PDFs	**PDF/Table Extraction**
Tesseract OCR	OCR for extracting text and simple tables from images/PDFs	**OCR for Images & PDFs**
PaddleOCR	OCR for table and text extraction, supports multi-language	**OCR for Images & PDFs**
TableNet	Extracting tabular data from document images	**Table Detection in Images**
Adobe PDF Extract API	Extracting structured data including tables from PDFs	**PDF/Table Extraction**
PyMuPDF (Fitz)	Extracting content (text, tables) from PDF documents	**PDF Parsing**
Tabula	Extracting tables from PDFs into CSV/Excel	**PDF/Table Extraction**
Keras-OCR	OCR for detecting and extracting text and tables from images	**OCR for Images**
LayoutLM	Pre-trained model for reading and extracting structured data from scanned documents	**Document Understanding/OCR**
TrOCR (Transformer OCR)	OCR model based on Transformer architecture for extracting text and tables	**OCR for Documents**

Amazon Textract	Automated text and table extraction from documents	**OCR for PDFs & Images**
Google Cloud Vision API	OCR with table detection capabilities for scanned images	**OCR for Images & PDFs**

Key Components of Modern AI Reasoning and Retrieval Libraries

LangChain

LangChain is a framework for building applications that leverage large language models (LLMs). It enables multi-stage reasoning and workflows by connecting components like prompts, retrievers, and tools.

- Prompt
- Chain
- Retriever
- Tool

LlamaIndex

LlamaIndex (formerly GPT Index) is a toolkit designed for indexing and querying large datasets using LLMs. It helps in efficiently storing and retrieving information with customized models and prompts.

- Models
- Prompts
- Indexing and Storing
- Querying
- Agents

Haystack

Haystack is an open-source framework for building search systems and question-answering pipelines. It enables the integration of components like retrievers, generators, and document stores to create powerful AI-driven search applications.

- Generators
- Retrievers
- Document Stores
- Pipelines

DSPy

DSPy is a framework that allows structured interactions with LLMs and AI components. It focuses on programmatic interfaces, facilitating workflows through modular components like signatures and automatic compilers.

- **Signature**
- **Teleprompters**
- **Automatic Compiler**

Study Use Cases

1. On-bench Employee Allocation System:
Design a RAG system to identify on-bench employees available for project allocation. Utilize a vector-based retrieval system to search through employee profiles, skill sets, and availability. Implement a reindexing mechanism to ensure real-time updates when employees' statuses change and apply a chunking strategy to handle large employee databases efficiently.

2. HR Document Search Automation System:
Develop a RAG-based HR automation system to track and search organization-level documents, such as policies, performance reports, and employee guidelines (**Word or PDF or Excel documents**). Use vector embeddings for document retrieval and manage embeddings to ensure documents are easily accessible. Implement a chunking strategy to divide long documents into smaller sections, improving the retrieval process and search performance.

3. Customer Support Chatbot:
Design a RAG-based system that automates customer support by retrieving product manuals, troubleshooting guides, and FAQs. Use vector embeddings to efficiently manage and rank similar queries, ensuring faster response times for users.

4. E-commerce Recommendation System:
Implement a LangChain agent that dynamically retrieves product information and reviews to provide personalized recommendations. Incorporate a reindexing mechanism to ensure real-time updates on inventory and new product embeddings, and utilize chunking to handle large product descriptions, improving retrieval performance for long-form data.

5. Legal Document Analysis Tool:
Create a chain tool that retrieves relevant case law, legal precedents, and statutes by managing embeddings of large legal text corpora. Use a chunking strategy to break down complex legal documents into smaller sections for more granular retrieval and enhanced search performance, allowing lawyers to efficiently draft legal arguments.

6. Academic Research Assistant:
Develop an agent application that retrieves academic papers and research articles using a vector-based retrieval system. Incorporate reindexing for newly published research and apply a chunking strategy to manage long academic texts, enabling more precise literature reviews and fast access to relevant material.

7. Financial Market Analysis Agent:
Build a Agentic system that retrieves real-time financial data and historical market trends. Integrate a **financial data API** (such as Alpha Vantage or Yahoo Finance) to fetch up-to-date stock prices, market

indices, and economic indicators. Use vector embeddings to optimize retrieval performance and manage large-scale time-series data efficiently. Additionally, incorporate a **calculation function** to analyze market trends and generate insights, such as moving averages or volatility calculations, enabling traders to make informed investment strategies. Integrate a **WikipediaRetriever tool** to provide context on market news and economic events that could impact stock performance.

8. Supply Chain Management System:
Design a LangChain system that retrieves inventory data and supplier information using vector embeddings for accurate matching of product details. Incorporate an agent architecture to automatically reindex stock levels and logistic routes, optimizing the entire supply chain workflow.

9. Clinical Decision Support System:
Implement an agent-based application that retrieves patient data, medical history, and research papers using an efficient vector-based retrieval system. Utilize embedding management and chunking to handle long medical records, ensuring healthcare professionals have access to the most relevant and up-to-date information for treatment decisions.

Connect on: LinkedIn Website

Generative AI Certification: Guide Notes

To register for Generative AI workshop, please click here