Contents

Fundamentals

- LLM: Model trained on massive datasets to achieve advanced language processing capabilities
- Foundation Models: Large ML model trained on vast amount of data & fine-tuned for more specific language understanding and generation tasks

Components

- Encoder: Converts text input into tokens (embeddings)
- Decoder: Converts generated output tokens back into meaningful words
- Transformer: Train the token embeddings

Databricks AI

- GenAI (Custom models, model serving, RAG)
- End-to-end AI (MLOps with MLFlow, AutoML, Monitoring, Governance)
- Databricks + MosaicML:
  - Rapid democratization of model capabilities
  - Making GenAI models work for enterprises
  - Unifying AI and data stack
  - Advantages: Customize models, secure environment, competitive

Legal and Ethical Considerations

- Prompt Injection: Inserting a specific instruction or prompt within the input text to manipulate the normal behavior of LLMs
- Prompt Engineering: Designing and crafting effective prompts or instructions for generating desired outputs from an LLM

1 Solution Development

From Prompt Engineering to RAG

- Prompt: Input or query given to a LLM to elicit a specific response
- Prompt engineering: Practice of designing and refining prompts to optimize the responses generated by an AI model
- Prompt components: Instruction, context, input/question, output type/format
- Prompt techniques:
  - Zero-shot: Prompt that generates text or performs a task without providing any examples or additional training specified to that task
  - Few-shot: Prompt provides with a few input-output examples to guide the model for generating the desired output
- Prompt chaining: Multiple tasks are linked together, with the output of one prompt serving as the input for the next → allows for more complex tasks to be broken down into manageable steps
- Chain-of-thought (CoT) prompting: Enhances the reasoning capabilities of LLMs by guiding them to articulate their though processes step-by-step

Tips and tricks

- Different models may require different prompts
- Provide examples and clues to guide model's response generation
- Different use cases may require different prompts
- Use delimiters to distinguish between instruction and context
- Ask the model to return structured output
- Ask the model not to hallucinate, not to assume, not to rush

RAG

- Passing context as model inputs improves factual recall
- Retrieve data/documents relevant to a question/task, provide them as context to augment the prompts to an LLM to improve generation
- Components:
  - Index & embed: Embedding model creates vector representations of the documents and the user query
  - Vector store: Specialized to store unstructured data indexed by vectors
  - Retrieval: Search stored vectors using similarity search to retrieve relevant information
  - Filtering & Reranking: Process of selecting or ranking retrieved documents before passing as context
  - Prompt augmentation: Prompt engineering workflow to enhance context via injection of data retrieved from the vector store
  - Generation
- Flow:
  - Documents are embedded
  - User asks question and LLM converts query to embeddings
  - Similarity search for embedded documents and embedded user query
  - Similar documents are passed as context to LLM generation model

  - Model augments and generates completion
- Databricks:
  - DLT
  - Mosaic AI Model Serving (embeddings, foundation models, custom models)
  - Mosaic AI Vector Search
  - Unity Catalog's Model registry
- Benefits:
  - Up-to-date and accurate responses,
  - Reducing inaccurate responses or hallucinations
  - Domain-specific contextualization
  - Efficiency and cost-effectiveness

Preparing Data for RAG Solution

- Issues:
  - Poor quality model output
  - Lost in the middle paradigm
  - Inefficient retrieval
  - Wring embedding model
- Data prep process:
  - Ingestion → Data storage → Chunking → Persist in vector store
  - Delta Lake for storing data and tables in the Databricks DI Platform
  - Unity Catalog for governance, discovery, access control for RAG applications
  - Document extraction: Split documents into chunks, embed chunks with a model, store them in a vector store
- Chunking data:
  - Context-aware chunking: By sentence/paragraph/section, leverage special punctuation, include metadata/tags/titles
  - Fixed-size chunking: Divide by a specific number of tokens, simple and computationally cheap method
  - Chunking overlap defines the amount of overlap between consecutive chunks ensuring that no contextual information is lost
  - Window summarization: Each chunk includes a windowed summary of previous chunks
- Data preparation in Databricks:
  - Ingestion: Tables and Volumes → Files & metadata
  - Document processing (parsing, cleaning, chunking, featurization): Workflows, DLT, Notebooks → Chunks & features
  - Embedding: Workflows, DLT, Notebooks → Chunks, vectors & features
  - Storage: Delta Tables → Automatic Sync
  - Vector DB: Vector Search

Vector Search

- Vector DB:
  - Store and retrieve high dimensional vectors such as embeddings
  - In RAG architecture, contextual information is stored in vectors
  - Specialized and fully-fledged DB for unstructured data

  - Speed up query search for closest vector
  - Use cases: RAG, recommendation engines, similarity search
- Vector similarity:
  - Distance metrics: L2 (Euclidian), Manhattan distance (L1)
  - Similarity metrics: Cosine similarity
- Vector search strategies: KNN, Approximate NN (ANN), Hierarchical Navigable Small Words (HNSW)
- Libraries create vector indices, plugins provide architectural enhancements
- Reranking:
  - Prioritize documents most relevant to user's query
  - Initial retrieval → Not all documents are equally important
  - Reranker: Reorder documents based on the relevance scores → Place most relevant documents at the top of the list
  - Adjusts initial ranking of retrieved documents to enhance precision and relevance
  - Supports deeper semantic understanding
  - Benefits: Improve accuracy, reduce hallucinations
  - Challenges: LLM must be called repeatedly, increasing cost and latency, adds complexity to RAG pipeline
- Mosaic AI Vector Search:
  - Stores vectors and metadata, Integrated with Lakehouse, supports Access Control Level (ACL) using Unity Catalog
  - Methods:
    * Delta Sync API with managed embeddings: automatic sync, fully managed embeddings
    * Delta Sync API with self-managed embeddings: automatic sync, self-managed embeddings
    * Direct access CRUD API: manual sync via API, self-managed embeddings
- Set up:
  - Create a Vector Search Endpoint: Compute resource
  - Create a Model Serving Endpoint: Foundation Models APIs, external or custom models
  - Create a Vector Search Index: Created and auto-synced from Delta Table, indexes appear in and are governed by Unity Catalog

Assembling and Evaluating a RAG Application

- RAG Application Workflow: Development → Expert/User testing → Offline evaluation → Production
- MLFlow
- Evaluating RAG Pipeline:
  - Components to evaluate: Chunking, embedding model, vector store (retrieval, reranker), generator
  - Context precision: Signal-to-noise ratio for retrieved context, based on query and context
  - Context relevancy: Measure relevancy of retrieved context, based on both the query and the context
  - Context recall: Measures the extent to which all relevant entities and information are

retrieved and mentioned in the context provided, based on ground truth and context
  - Faithfulness: Measures the factual accuracy of generated answer in relation to provided context, based on responses and retrieved context
  - Answer relevancy: Assess how applicable the generated response is to the user's query, based on the alignment of the response with the user's intent or query specifics
  - Answer correctness: Measures accuracy of generated answer when compared to the ground truth, based on ground truth and response, encompasses both semantic and factual similarity with ground truth
- MLFlow evaluation:
  - Batch comparisons
  - Rapid and scalable experimentation
  - Cost effective

2 Application Development

Foundations of Compound AI Systems

- Compound AI System: AI System that has multiple interacting components
- Systems: Prompt engineering, RAG, Agent-based chain, orchestration chain etc.
- Prompts have multiple intents → Consists of one to many tasks, e.g., translate, summarize, analyze sentiment, and classify
- Designing Compound AI Systems:
  - Approach: Analysis, design, development, production, monitoring
  - Identify intents, tools, build the chain

Building Multi-Stage Reasoning Chains

- Map concepts with technical terms
- Composition frameworks help to manage multi-stage reasoning systems
- LangChain main components:
  - Prompt: Structured text to communicate a specific task/query to an LLM
  - Chain: Sequence of automated actions or components
  - Retriever: Interface returning relevant documents
  - Tool: Functionality/resource that an agent can activate to perform a specific task
- LlamaIndex: Framework that enhances capabilities of LLMs by structuring and indexing data
- Haystack: Python framework for building applications with LLMs, focusing on document retrieval, text generation, and summarization
- Databricks products for Building Multi-stage Reasoning Systems:
  - Foundation Model API → Access and query open GenAI models
  - DBRX:
    - DBRX Base: pre-trained model, functions like smart auto-complete, useful for further fine-tuning
    - DBRX Instruct: fine-tuned model, answers questions and follows instructions
  - Vector Search:
    - Stores vector representation of data and metadata
    - Integrated with Lakehouse
    - API for real-time similarity search

Agents and Cognitive Architectures

- Agent:
  - Application that can execute complex tasks by using a LM to define a sequence of actions to take
  - Compound AI systems: Hard-coded calls to external tools and services
  - Agents replace hard-coded sequence of actions with a query-dependent sequences chosen dynamically by LLMs
- Non-agentic workflow: LLM generates an answer based on actions defined
- Agentic workflow: Agent does research, writes first draft, another model checks the draft (less determined)
- Workflow:

  - Process tasks
  - Collect data
  - Data analysis
  - Output generation
- Agent Reasoning: Cognitive process by which AI agents analyze information, draw logical conclusions, and make decisions autonomously
- Design patterns:
  - ReAct (Reason + Act):
    * Enables models to verbal reasoning traces and actions
    * Main states:
      · Thought: Reflect on the problem given and previous actions taken
      · Act: Choose correct tool and input format to use
      · Observe: Evaluate the result of the action and generate next thought
  - Tool use / function calling: Research tools, document retrieval, coding, image
  - Planning: Agents must be able to dynamically adjust their goals and plans based on changing conditions
  - Multi-agent collaboration
- Multi-Modal AI: Models with inputs/outputs that include data types beyond text (image, audio, video)

3 Application Evaluation and Governance

Securing and Governing GenAI Applications

- Data and AI Security Framework (DASF) Databricks as Security:
  - Algorithm: Model Serving, Lakehouse Monitoring
  - Evaluation: MLflow, Lakehouse Monitoring
  - Model Management: MLflow, Unity Catalog
  - Catalog: Unity Catalog
  - Operations: Asset Bundles, CLI, Secrets
  - Platform: Cloud architecture, Serverless
- Key Security Tooling: Unity Catalog, Mosaic AI

Evaluation Techniques

- Metrics:
  - Human feedback, LLM-as-a-judge
  - Perplexity $\rightarrow$ Low Perplexity = High confidence
  - Toxicity $\rightarrow$ How harmful is the model?
- Benchmarking: Compare models against standard evaluation datasets
- LLM-as-a-judge

End-to-End Application Evaluation

4 Application Deployment and Monitoring