Introduction to GenAI

Presented by Deepak Sood

whoami

Senior AI, Data and DevOps Architect @ OpsTree

- M.Tech IIITD CSE 2015-17 Batch
- Software Engineer
- DevOps Engineer
- Data Engineer
- Engineering Lead
- Architect
- Product + Project + Leadership + Strategy + Hiring

Hobbies

- Learning
- Problem Solving
- Note Taking
- https://deepaksood619.github.io/
- http://linkedin.com/in/deepaksood619

Current Levels

Who has heard of the following things -

1. AI
2. ML
3. ChatGPT
4. Using ChatGPT ?
5. Mid Journey / Dall.E
6. LLM
7. Models - Llama, Mixtral
8. Transformers
9. Embeddings
10. Tools - LangChain
11. Vector DBs

Objective

1. What is GenAI
2. What problem does it solve
3. What was before it
4. What is coming up

Technicals

1. Prompt Engineering
2. LLMs
3. Embeddings
4. RAGs
5. Hands-on / Hackathon

Before AI/ML - Simple Rule-Based Programming

Rule-Based Systems: Programs that operate using a predefined set of rules (e.g., "if-else" statements) to make decisions or solve problems.

How it Works:

Fixed Logic: Developers write explicit rules that dictate the program's behavior in specific scenarios.

Deterministic: The outcome is predictable and the same every time for the same input.

Key Characteristics:

Static Rules: Cannot adapt or learn from new data.

Limited Complexity: Only works well for simple, clearly defined tasks.

Challenges with Rule-Based Programming

Scalability Issues:

Inflexibility: Adding new rules or changing existing ones requires manual updates, leading to complex and unmanageable code as the system grows.

Rule Explosion: Large numbers of rules can make the system cumbersome, hard to maintain, and prone to errors.

Lack of Adaptability:

No Learning: Cannot adapt to new situations or learn from past experiences.

Limited Scope: Inefficient for complex problems requiring pattern recognition, prediction, or handling ambiguity.

What are Traditional AI/ML Models?

Artificial Intelligence (AI): Refers to machines simulating human intelligence processes.
Machine Learning (ML): A subset of AI where models learn patterns from data to make predictions or decisions without explicit programming.

Types:

Supervised Learning: Trained on labeled data.
Unsupervised Learning: Identifies patterns in unlabeled data.
Reinforcement Learning: Learns by interacting with the environment and receiving feedback.

Key Characteristics:

Data-Driven: Relies heavily on large datasets.
Task-Specific: Models are designed for specific tasks (e.g., image recognition, language translation).
Feature Engineering: Manual process of selecting relevant features from raw data for better model accuracy.

Challenges with Traditional AI/ML Models

Data Dependency:

Volume Requirement: Needs vast amounts of labeled data, which can be costly and time-consuming to gather.

Quality Issues: Performance is sensitive to data quality; biases and errors in data can degrade model accuracy.

Generalization Limitations:

Task-Specific: Struggles with adapting to new tasks or domains without significant retraining.

Scalability: Expanding the model's scope often requires complex retraining processes.

Development Complexity:

Feature Engineering: Requires expert knowledge to handcraft features, making model development time-intensive.

Model Interpretability: Models can be black boxes, making it difficult to understand and explain decisions

Introduction to GenAI

Generative AI is revolutionizing the way we interact with technology. By leveraging advanced algorithms, we can create content that **mimics** human creativity.

- GenAI focuses on creating new content such as text, images, music or even code.
- History - From rule-based systems to advanced neural networks.
- Key breakthroughs and milestones - GANs and transformer models have significantly advanced GenAI.

LLMs

A large language model is a type of artificial intelligence algorithm that applies neural network techniques with lots of **parameters** to process and understand human languages or text using self-supervised learning techniques. Tasks like text generation, machine translation, summary writing, image generation from texts, machine coding, chat-bots, or Conversational AI are applications of the Large Language Model. Examples of such LLM models are Chat GPT by open AI, Gemini by Google, Llama by Meta, etc.

Working with LLMs - Prompt Engineering

Prompt Engineering: The process of designing and optimizing input prompts to effectively guide AI models, especially large language models (LLMs), in generating accurate, relevant, and contextually appropriate responses.

Key Concepts:

Prompt Structure: Crafting prompts that are clear, concise, and aligned with the desired outcome.

Iterative Refinement: Continuously tweaking and testing prompts to improve the quality and accuracy of AI-generated responses.

Task-Specific Prompts: Tailoring prompts to suit specific tasks like summarization, translation, or generating creative content.

Prompt Engineering – Examples

Example 1: Summarization

Prompt: "Summarize the following article in one paragraph: [Insert article text]."

Example 2: Question Answering

Prompt: "Based on the text provided, answer the following question: What are the key benefits of prompt engineering?"

Example 3: Creative Content Generation

Prompt: "Write a short story about a robot discovering emotions."

Challenges with LLM

- **Hallucinations -** Presenting false information when it does not have the answer.
- Presenting **out-of-date** or **generic** information when the user expects a specific, current response.
- Creating a response from **non-authoritative sources.**
- Creating inaccurate responses due to terminology confusion, wherein different training sources use the same terminology to talk about different things.

Solution - RAG

Retrieval-Augmented Generation (RAG) is the process of **optimizing the output** of a large language model, so it references an **authoritative knowledge base** outside of its training data sources before generating a response. **RAG extends** the already powerful capabilities of LLMs to **specific domains** or an **organization's internal knowledge base**, all without the need to retrain the model. It is a cost-effective approach to improving LLM output so it remains **relevant, accurate, and useful** in various contexts.

Benefits of RAG

Cost-effective implementation

Contextual Relevance

Current information

More developer control

Enhanced user trust

Reducing hallucinations

Practical Applications of RAG

RAG finds applications in various domains and industries, leveraging its ability to combine retrieval-based and generative techniques to enhance text generation and information retrieval.

01.

Question Answering Systems

RAG is particularly valuable in question-answering applications. It can retrieve and generate precise and contextually relevant answers to user queries, making it suitable for virtual assistants, FAQs, and expert systems.

02.

Content Summarization

RAG can be employed to summarize lengthy documents, articles, or reports by selecting the most salient information and generating concise summaries. This is useful for content curation and information digestion.

03.

Information Retrieval

RAG can enhance traditional information retrieval systems by providing more contextually relevant and coherent results. It improves the precision and recall of search engines, making it valuable in research and knowledge management.

Education

RAG can assist in creating educational materials by generating explanations, study guides, and tutorials. It ensures that the content is informative and aligned with the educational context.

Legal

In the legal domain, RAG can be applied to retrieve case law, statutes, and legal opinions. It helps lawyers and legal professionals access relevant legal information efficiently.

Healthcare

RAG can assist healthcare professionals in decision-making by providing up-to-date medical information, research findings, and treatment guidelines. It aids in evidence-based medicine.

Finance

RAG models can generate financial reports, market summaries, and investment recommendations based on real-time data and financial databases, assisting analysts and investors.

Embeddings

Embeddings are numerical representations of data (such as words, images, or documents) in a continuous vector space, where similar items are placed closer together.

Purpose: They capture the semantic meaning and relationships between items, enabling AI models to perform tasks like similarity matching, classification, and clustering.

How Embeddings Work:

Vectorization: Converts complex data into fixed-size vectors of numbers.

Semantic Mapping: Similar data points (e.g., words with similar meanings) are mapped to nearby points in the vector space.

Example: In language models, the words "king" and "queen" are close to each other in the embedding space, reflecting their related meanings.

Embeddings role in RAG

Information Retrieval: Embeddings are used to represent both queries and documents. When a query is made, the system retrieves the most relevant documents by comparing their embeddings.

Contextual Relevance: The retrieved documents provide context to the AI model, which uses this information to generate more accurate and contextually appropriate responses.

Process:

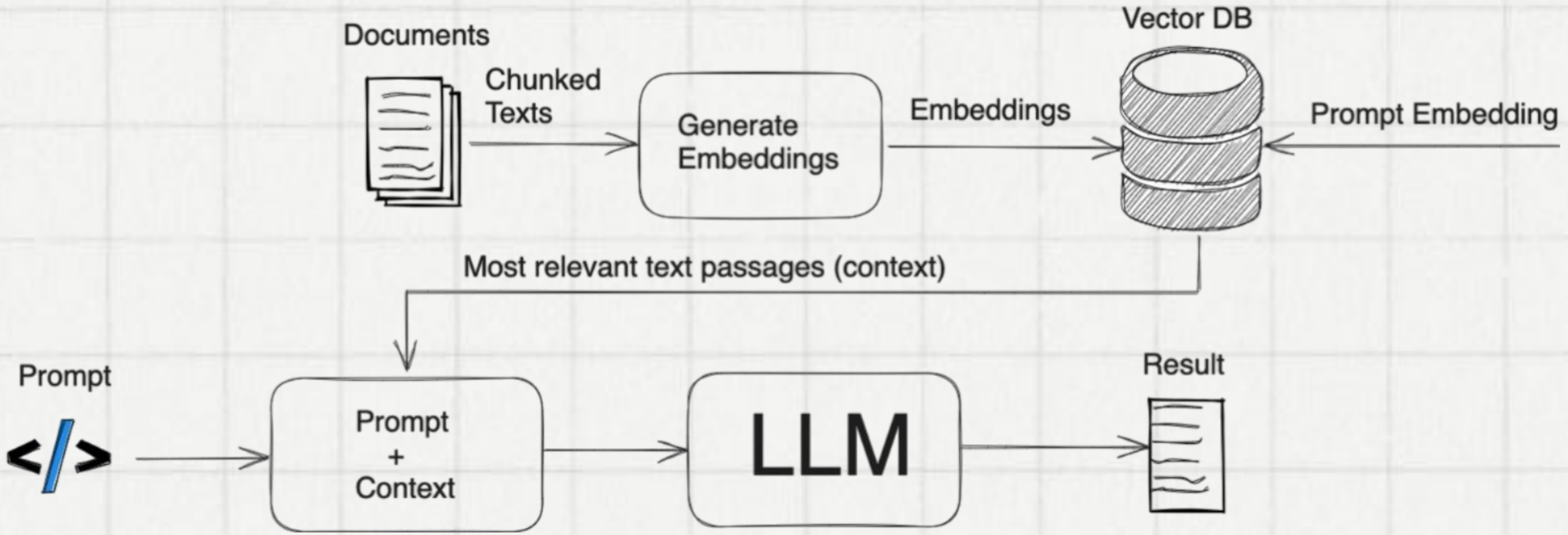
Embedding Generation: Both the user's query and the corpus of documents are converted into embeddings.

Similarity Matching: The system searches for documents with embeddings similar to the query.

Response Generation: The retrieved documents are used to enhance the model's response generation.

Algorithms – Approximate Nearest Neighbor (ANN) Search, Cosine Similarity, Euclidean Distance

RAG Framework



Components of RAG

KnowledgeBase

- APIs, databases, or document repositories.
- Formats like files, database records, or long-form text.
- Embedding language models in a vector database

Retriever

The RAG retriever component is responsible for the initial step of retrieving relevant information from external knowledge sources. It uses retrieval techniques such as keyword-based search, document retrieval, or structured database queries to fetch pertinent data.

Ranker

The RAG ranker component refines the retrieved information by assessing its relevance and importance. It assigns scores or ranks to the retrieved data points, helping prioritize the most relevant ones.

Generator

The RAG generator component is responsible for taking the retrieved and ranked information, along with the user's original query, and generating the final response or output.

Tools for Building a RAG

Text Embeddings

- MTEB - Massive Text Embeddings Benchmark
- FlagEmbedding
- SFR-Embedding-2_R
- AnglE

Vector DBs

- VertexAI
- Pinecone
- Milvus
- Chroma
- Even - Mongo and Postgres

Frameworks

- VertexAI
- LangChain
- LlamaIndex
- Haystack

LLM Models

- Gemini Pro and Flash
- Llama 3.1
- OpenAI
- Mixtral

Best Practices for RAG Implementation

01.

Data Management

Data Management in RAG systems involves storing not just raw text but additional contextual information. This strategy enriches the context available to the model, enhancing its ability to generate relevant and accurate responses.

02.

Embedding Optimization

Embedding Optimization focuses on refining data representations within the model's latent space. This process improves the accuracy of these representations, ensuring they are more aligned with the intended meaning and context.

03.

Advanced Retrieval Techniques

Advanced Retrieval Techniques in RAG include methods like recursive retrieval, hierarchical retrieval (Parent-child relationship), hypothetical questions indexed and summary indexed. These techniques are employed to enhance the model's ability to access and utilize the most relevant information from vast data sets.

Challenges and Future Directions of RAG

Handling Diverse Query Types

RAG models need to be versatile enough to handle a wide range of query types, from straightforward factual questions to more complex, nuanced queries. Adapting the retrieval and generation components to suit this diversity can be challenging.

Balancing Retrieval and Generation

Striking the right balance between retrieval and generation is crucial. Over-relying on retrieval may lead to responses that lack creativity or context, while excessive generation may result in less factual or relevant answers.

Scaling to Large Datasets

As knowledge bases and data sources continue to grow, RAG models must scale efficiently. Handling massive datasets without sacrificing response times and accuracy is a technical challenge.

Evaluation Metrics

Assessing the performance of RAG models can be complex. Traditional metrics may not fully capture the quality of responses, especially when factual accuracy and contextual relevance are critical.

Final reflections

RAG is a game-changer for LLMs, empowering LLMs with access to external knowledge are transforming their capabilities. By leveraging powerful tools like Gemini and Vertex AI, developers and businesses can harness the potential of RAG to build intelligent and insightful AI solutions.

Thanks

DO YOU HAVE ANY QUESTIONS?

https://deepaksood619.github.io
bit.ly/deepnotes

http://linkedin.com/in/deepaksood619
https://bit.ly/deepak-link

Tutorial

Prompt Engineering using Gemini

https://github.com/google-gemini/cookbook

https://bit.ly/gemcook

Prompt Engineering using Gemini

https://bit.ly/gemcook

1. Go to Google AI Studio.
2. Login with your Google account.
3. Create an API key.
4. Use a quickstart for Python, or call the REST API using curl.

https://github.com/google-gemini/cookbook/blob/main/quickstarts/Prompting.ipynb

1. Run in Colab

RAG Walkthrough

https://github.com/GoogleCloudPlatform/generative-ai/blob/main/gemini/qa-ops/building_DIY_multimodal_qa_system_with_mRAG.ipynb

https://github.com/huggingface/cookbook/blob/main/notebooks/en/rag_with_hugging_face_gemma_mongodb.ipynb

Hackathon

https://deepaksood619.github.io/ai/llm/rag-hackathon-questions
https://bit.ly/raghack