

A close-up photograph of a laptop keyboard, focusing on the right side. A circular inset is overlaid on the image, showing a person in a virtual environment, possibly a game or simulation, with a blue and white background. The text "RETRIEVAL-AUGMENTED GENERATION" is written in bold, black, uppercase letters across the bottom left of the image.

RETRIEVAL-AUGMENTED GENERATION



MODULE 1

Introduction to RAG and LLMs



WHAT IS RAG ?

RAG, or Resource Allocation, is a process that involves the distribution of resources (such as time, money, or personnel) to different projects or tasks. It is a critical part of project management, as it ensures that resources are used efficiently and effectively. RAG is often used in the context of project management, where it helps to identify the resources needed for a project and allocate them accordingly. This process is essential for ensuring that a project is completed on time and within budget. RAG is a key component of project management, as it helps to ensure that resources are used efficiently and effectively.

Core Concepts

01

• **Personal knowledge management** is a learning management system with integrated notes, calendar, and social network capabilities. [\[1\]](#)

02

• **Personal organization system** is a software only solution to provide better time management or workflow management. [\[2\]](#)

03

• **Content management system** is a software solution to help manage content and content workflow. [\[3\]](#)

Principles

Investing with purpose in **social** businesses

• ensuring that government companies play a role in achieving social and economic goals

• Encouraging **social** investments without **return**

• Using **market** mechanisms to improve **social** performance
without **loss** of **social** investment and **social** return

Advantages of RAG over Traditional LLM

Aspect	Traditional LLM	RAG
Knowledge Source	Relies on a static, pre-trained model during training, which may become outdated.	Draws from external, up-to-date sources, ensuring information is current.
Customization	Requires retraining the model to incorporate new information, which is costly.	Allows for easy updates to the knowledge base without retraining the model.
Transparency	Often lacks transparency in how information is retrieved and processed.	Can track the source of information, providing better auditability and trust.
Scalability	Scaling up often requires significant retraining and infrastructure changes.	Can scale by adding more external data sources without retraining the model.

Use Cases of RAG

1. **Enhanced Content Creation**
2. **Customer Feedback Analysis**
3. **Market Intelligence**
4. **Personalized Recommendations**
5. **Dialogue Systems and Chatbots**



Key capabilities

Test generation.
Identifying problems and creating test scenarios from scratch.

Understanding.
Understanding and understanding how things work.

Translating.
Converting what someone says into what they really mean.

Evolution of LLMs



ChatGPT

GPT (Generative Pre-trained Transformer)

- Introduced 2018 via GPT-1
- GPT-2 (2019): First public release to showcase the power of pre-trained models, capable of generating text matching human-like quality (e.g., summarizing paragraphs).
- GPT-3 (2020): Introduced the concept of massive scale (175 billion parameters) for a wide range of tasks and domains.



GPT-2

GPT-2

- Released 2019 by OpenAI
- May represent highest yet in AI natural language generation
- Highest rated natural language generation model to date



GPT-3

GPT-3

- Released: 2020 Aug (OpenAI)
- May outperform human-level text, natural language, and image classification
- Released: 175 billion parameters. First ever released for computer-vision tasks



GPT-4

ChatGPT (and GPT-4)

- **OpenAI** (which are the ones who created GPT-4)
- they have **gpt-4** (which is a **commercial model** designed for commercial use) while **GPT-4o** is **giving institutional capabilities** (free and limited)
- **OpenAI** (which are the ones who created GPT-4) are **not** **responsible** for **misinformation** and **disinformation**.



GPT-4o

GPT-4o

- OpenAI's latest AI model
- It's the first AI model to be able to see, hear, and speak in real-time
- It's also the first AI model to be able to see, hear, and speak in real-time



GPT-4o

GPT-4o

- OpenAI's latest AI model
- It's the first AI model to be able to see, hear, and speak in real-time
- It's the first AI model to be able to see, hear, and speak in real-time



GPT-4o mini

GPT-4o Mini

- Introduced: March 2024 alongside GPT-4o by OpenAI.
- Key differentiator from its predecessor is its compact, efficient architecture, offering a fraction of GPT-4o's cost while maintaining high performance across various multimodal inputs.
- OpenAI emphasizes advanced AI research, emphasizing use in education and healthcare to assist and enhance capabilities for clinical diagnosis, and the risks of widespread access to powerful generative models.



BERT (Bidirectional Encoder Representations from Transformers)

- Introduced 2018 by Google
- Now the state-of-the-art for natural language processing (NLP) tasks, including question-answer, text classification, and text summarization
- Employs two separate encoders for each direction, allowing for bidirectional representations of each word in a sentence



T5 [Text-To-Text Transfer Transformer]

- introduction: very big change
- easy to use system: The model can take a text as input and output an arbitrary length as output like a traditional sequence-to-sequence model
- Dataset: Sources available for model such as news, Wikipedia, etc. are used as the training data as a single source of data

Gemini 1.5

Gemini 1.5

- Google's primary video engine
- can process videos up to 1 million seconds with captions or Google Slides and Google for improved efficiency.
- Gemini 1.5 Pro can process 2048x1024px images and 1080p video, which is 10x more than other models. It also has a 128K context window, which is 10x more than other models. It also has a 128K context window, which is 10x more than other models.

Gemini 1.5 Pro

Gemini 1.5 Pro

- Gemini 1.5 Pro is a multimodal model capable of processing and generating content across text, images, audio, and video.
- It has a context window of up to 1 million tokens, allowing it to handle extremely long documents and videos.
- Gemini 1.5 Pro is designed for a wide range of applications, including content creation, research, and customer support.

Current State-of-the-Art

- **Manufacturers and largest supermarket chains** are evaluating increasingly sophisticated, low-cost barcode technologies, such as **radio-frequency identification (RFID)** and **image-based** and **video-based** types.
- **Retail and manufacturers** are trying to improve the way they manage their stock, from top- and bottom processes, including **inventory** tracking and **supply chain** management.
- **Manufacture applications** that are also being used for **various applications** from **production** and **inventory** and **marketing** and **customer relationship** management, **including** tracking the **supply chain** as well as **inventory**.

Types of LLMs

Open-source

Closed-Source

Open-source model :- Llama 3.1 by Meta



Open-source model :- Gamma by Google



Types of LLMs

Open-source

Closed-Source

Types of LLMs



closed-source LLMs

Definition: These models are owned by companies and are usually available for use via APIs. Some developers may release their own and underlying model architecture, but training data remains undisclosed.

Claude by Anthropic

Claude is a powerful language model with a focus on safety and ethical use. Claude simulates user intent and understands harmful requests. It interprets the guidelines as align with human values.

Anthropic Claude believes that when using AI safely, we can help, do no harm, be truthful, and be accountable. Claude is designed to be helpful and harmless in its behavior.

Closed-Source GPT-4.0 mini Model by OpenAI

Capabilities

Fast Prototyping

Ideal for quickly developing and testing AI-powered applications, thanks to its user-friendly interface and tools.



Efficient Reasoning

Excels at complex reasoning tasks and solving challenges, making it suitable for advanced applications.



Seamless Integration

Easily integrates with other systems, making it versatile for various enterprise-level and small business applications.

Closed-Source GPT-4.0 mini Model by OpenAI

URA CANAL

Advanced reasoning
capabilities: high
quality, consistent
answers, handling
complex, open-ended,
creative writing.

Extensive support
for data analysis
and visualizations,
for research and
marketing
insights.

Personalized user
experiences through
learning and
adaptation to user
preferences.

Closed-Source GPT-4 Model by OpenAI

Capabilities



Closed-Source GPT-4 Model by OpenAI

OpenAI

On-Device AI model for personal assistants, offline translation, and privacy-sensitive tasks on smartphones and low-power devices.

Real-Time Translators: Enables live language translation for travel apps, international communications, and multilingual support.

Personalized Content Creation: AI-driven writing, long-form storytelling, and creative brainstorming, providing personalized learning experiences.

Comparison Of Open-Source And Closed -Source Models

Aspect	Open-Source Models	Closed -Source Models
Accessibility	Free and accessible to everyone. No license, individual ownership.	User-friendly with GUIs, but access is restricted and requires payment.
Customizability	Highly customizable with full code access, test code, technical help.	Limited customization through APIs, less flexibility.
Performance	High performance with community contributions, version log, frequent updates.	Generally high performance with vendor optimizations limited to proprietary infrastructure.
Cost	Free software but users bear full infrastructure and maintenance.	Subscription or usage fees, vendor support and maintenance.

The Role of RAG in Enhancing LLM Capabilities



Improving Factual Consistency



Examples of LLM Outputs:

- With GPT-4o, the LLM correctly identifies the correct location of Australia in Australia, but the location of Australia in Australia is not the same as the location of Australia in Australia.
- With GPT-4o, the LLM correctly identifies the correct location of Australia in Australia, but the location of Australia in Australia is not the same as the location of Australia in Australia.

Improving Factual Consistency



How RAG grounds responses in retrieved information

RAG systems use LLMs to synthesize relevant information retrieved from your data for more grounded responses.

Example

1. **Query Generation** The model generates a query from the input.
2. **Information Retrieval** Relevant documents are retrieved from the data source.
3. **Response Generation** The LLM uses the retrieved documents to generate a grounded response.

Reducing Hallucinations



Definition of Hallucination in LLM Context

is the output of LLM, a text generation system, that generates content not grounded in the input data. This can be defined as a text generation system that generates content not grounded in the input data. This can be defined as a text generation system that generates content not grounded in the input data.

Reducing Hallucinations

Techniques RAG Uses to Mitigate False Information

Diversified Querying: Distributed, context-relevant queries to retrieve relevant information.

Post-Processing Checks: Apply checks after retrieval generation to prevent hallucinations, further refining.

Contextual Validation: Cross-checking data from multiple sources to ensure accuracy and consistency.

Information Retrieval: Employ retrieval from more external sources, applying filters to ensure the retrieved information is relevant and accurate.

Enhancing Domain-Specific Knowledge

Adapting General LLMs to Specialized Fields



Enhancing Domain-Specific Knowledge

Case Studies of RAG in Various Industries

Healthcare: Enhances the latest medical research to provide accurate treatment plans and drug recommendations, improving clinical decision-making.

Finance: Pulls data from market reports for ethical investment advice, enhancing risk management and financial forecasting.

Legal: Accesses case law and precedents, improving legal research and document drafting.

Customer Support: Uses a company's knowledge base to deliver accurate answers, reducing customer support inquiries.

Demo

Running a Basic Open Source LLM Locally



MODULE 2

Vector Databases And Embeddings



Understanding Vector Databases

Definition: A vector database stores and queries vector embeddings, numerical representations of data that are typically generated by machine learning models. These embeddings capture semantic meaning and enable efficient similarity search.

Key Concepts



Vector Embeddings: Numerical representations (often dimensional vectors) that encode semantic properties of objects. Composed by models like GPT, LLM, or LLaMA, they represent data in a compact, machine-readable form.



Relationships: Connections between entities in the query space, often represented as:

- **Explicit Relationships:** Direct links between entities.
- **Implicit Relationships:** Indirect links inferred from context.
- **Latent Space Relationships:** Relationships learned in the latent space.



User Queries:

- **Formal/Structured Queries:** Well-defined, structured queries or documents.
- **Informal/Unstructured Queries:** Less structured, often conversational or free-form queries.
- **LLM-based queries and documents as input/output data.**

Core Components of Vector Databases

Querying: Efficiently searching through vectors to retrieve relevant information based on specific criteria.

Scalability: Designed to handle massive amounts of data, scaling to millions or billions of vectors.

Storage: Optimized to store large numbers of high-dimensional embeddings, ensuring fast retrieval and management.

Indexing: Utilizes specialized algorithms like HNSW, IVF, and LSH to enable fast similarity and nearest-neighbor searches.

Role in RAG Systems

Efficient Storage of Document Embeddings

Storage: RAG systems generate embeddings for documents, which are stored in vector databases. These embeddings capture the semantic content of the documents in a format that can be efficiently searched, even with large volumes of data.

Availability: Vector databases are designed for high-performance retrieval, making them ideal for RAG systems that require storing and retrieving document embeddings.

Role in RAG Systems

Fast Similarity Search for Relevant Information Retrieval

Similarity Search: When a query is received, the RAG system generates a query embedding. The vector database performs a fast similarity search, comparing this embedding against the stored document embeddings.

Nearest Neighbor Search: The database identifies the most semantically similar documents or passages to the query, enabling the retrieval of the most relevant information.

Real-Time Performance: The speed and efficiency of vector databases ensure that relevant documents can be retrieved quickly, supporting real-time applications like chatbots, search engines, and recommendation systems.

Types Of Vector Databases

FAISS (Facebook AI Similarity Search)

- An open source library designed by Facebook AI that supports efficient similarity search and clustering of dense vectors.

Waviole

- An open source vector search engine that combines vector embeddings with traditional search techniques.

Alibaba

- A fully managed vector database service designed for fast similarity search, including support for real-time data updates.

Million

- A distributed vector database service designed for large-scale applications, supporting advanced querying capabilities.

Use Cases Beyond RAG



Recommendation Systems: Virtual assistants like Amazon's Alexa use personalized recommendations for products, movies, or music, enhancing user experience.



Image Analysis: Applications like Google Photos use image recognition to categorize photos, identify objects, and suggest similar images.



Anomaly Detection: Banks use machine learning to identify unusual transactions that deviate from the norm, helping to detect fraud or suspicious activity.



Introduction to Embeddings

Definition: Embeddings are dense vector
representations of data (e.g., text, images, audio) that
capture semantic meaning. Generated by machine
learning models, they assign numerical values to
objects in vector space. This enables tasks like
similarity search and recommendation.

How Embeddings Work



Visualizations of word and sentence embeddings can be visualized in a 2D or 3D space (e.g., 2D for word embedding, 3D for sentence embedding), showing how these embeddings maintain semantic relationships meaning.



Embedding Word and Meaning. Embedding transforms data into high-dimensional vectors, where similar data points (e.g., words, sentences) are positioned closer together, reflecting their semantic similarity.

Different Embedding Models

Word2Vec: This popular embedding technique generates words as a context-sensitive embedding, which is useful for capturing semantic relationships. It uses two models: **Continuous Bag-of-Words (CBOW)** and **Skip-gram**, both of which are trained on word co-occurrence data.

Glove: Global Vectors for word representation. This model generates word embeddings by averaging the word co-occurrence matrix. It is a simple and effective model for capturing semantic relationships between words.

BERT: Bidirectional Encoder Representations from Transformers. This model generates word embeddings by considering the entire context of the sentence, allowing for improved understanding of word meanings in different contexts.

GPT Embeddings

Description: GPT models generate embeddings using a transformer architecture, capturing an diverse texts, including rich contextual information.

Advantages:

- **Contextual Understanding:** Captures nuanced meanings and variations.
- **Versatility:** Handles both generated, natural-looking, and structured.
- **Interpretable:** Can be used to explain model behavior for various domains.

Use Cases:

- **Recommendation:** An effective approach for analyzing systems.
- **Content Generation:** Creates coherent and contextually relevant text.
- **Research:** Useful for analyzing complex, high-dimensional data.

Indexing And Similarity Search



Introduction To Indexing and similarity

Indexing is a technique used to store and retrieve data efficiently. It involves creating a list of keywords or terms that are associated with a document or a set of documents. This list is then used to search for specific information. Similarity search is a technique used to find documents that are similar to a given document. It involves comparing the keywords or terms of a document to the keywords or terms of other documents in a database. This comparison is done using a similarity metric, such as cosine similarity or Jaccard index. The results of the search are then ranked based on their similarity to the given document.

Techniques For Efficient Retrieval

FAISS (Facebook AI Similarity Search)



not presented a big leap in similarity search speed with indexing, allowing us to work with trillions of items

Advantage: Easy to use and well supported. One of the best libraries.



HNSW (Hierarchical Navigable Small Worlds)

Uses a multi-layered graph for efficient data search.

Advantage: High accuracy and fast search times, especially in high-dimensional spaces.

Techniques For Efficient Retrieval

Annoy (Approximate Nearest Neighbors Oh Yeah)

“

- **Disadvantages:** A brute force indexing method that stores candidate embeddings for every possible query, which is a less and more inefficient than to find approximate nearest neighbors.
- **Advantage:** Suitable for scenarios where retrieval speed is crucial even at the cost of some accuracy.

”

Speed vs. Accuracy Trade-off

Accuracy

- **HEURIS**. Provides fast (implies accurate) results is available for tasks where precision is critical.
- **DR**. Provides a balanced accuracy with decent accuracy and speed available for a wide range of applications.
- **ResNet**. Provides state-of-the-art speed-accuracy trade-off for tasks where accuracy is critical.



Speed

- **ResNet**. Provides a good balance of speed and accuracy, suitable for tasks where speed is critical.
- **DR**. Provides a good balance of speed and accuracy, suitable for tasks where speed is critical.
- **HEURIS**. Provides a good balance of speed and accuracy, suitable for tasks where speed is critical.

Similarity Metrics



Manhattan Distance

- **Description:** Measures the distance between points, moving only horizontally and vertically.
- **When to Use:** Text analysis and high-dimensional spaces where distance is key.



Euclidean Distance

- **Description:** Measures the straight-line distance between points, considering both direction and magnitude.
- **When to Use:** Clustering and classification in low-dimensional spaces.



Cosine Distance

- **Description:** Computes the angle of vector magnitudes and the cosine of the angle.
- **When to Use:** Large-scale searches and recommendation systems with normalized vectors.

Demo

Setting up a Simple Vector Database



MODULE 3

Building a Basic RAG Pipeline

Building a Basic RAG Pipeline

Overview of the RAG Architecture



Document Loaders And Text Splitting

Types of Document Loaders

PDF Loader

- Fetches documents and splits them into PDFs.
- Use Cases: contracts, reports, legal documents.
- Tools: PyPDF2, pdfplumber.

HTML Loader

- Fetches, parses and extracts content from HTML.
- Use Cases: blog posts, articles, web content.
- Tools: BeautifulSoup, lxml, requests.

CSV Loader

- Fetches, parses and extracts CSV data.
- Use Cases: spreadsheets, tabular data.
- Tools: pandas, csvkit, sqlalchemy.

JSON Loader

- Fetches, parses and loads JSON data.
- Use Cases: API responses, config files, data exchanges.
- Tools: Python's json module, jq, jq-cli.

Text Splitting Techniques

By Paragraph: Splits text at paragraph breaks.
Use Case: Useful for processing text in chunks of paragraphs.

By Sentence: Splits text at sentence boundaries.
Use Case: Useful for processing text in chunks of sentences.

By Word: Splits text at word boundaries.
Use Case: Useful for processing text in chunks of words.

By Character: Splits text at character boundaries.
Use Case: Useful for processing text in chunks of characters.

Embedding Generation And Vector Storage

choosing an embedding model

space reduced models

- Pros: space efficient, fast to train
- Cons: requires fine-tuning and extensive evaluation, reduced accuracy and generalization
- Trade-offs: cannot do long-term or cross-domain embeddings

resource reduced models

- Pros: fast to train
- Cons: requires large amount of data, complex architecture, cannot do long-term
- Trade-offs: cannot do long-term or cross-domain embeddings

advanced models

- Pros: high accuracy, fast to train
- Cons: requires large amount of data, complex architecture, cannot do long-term
- Trade-offs: cannot do long-term or cross-domain embeddings

Embedding Generation And Vector Storage

Integrating with vector databases

Indexing strategies for large document collections

Indexing process is not simple

- navigation (comparing query embeddings with all stored index entries)
- advantages: fast runtime navigation, simple implementation

Approximate Nearest Neighbor (ANN) indexing

- Brute-force, FLAT, Annoy, HNSW
- navigation time depends on search algorithm used
- Advantages: Fast search, multiple partitions of vectors

Query Processing and Augmentation

Query Understanding and Reformulation



Query Processing and Augmentation

Techniques for Expanding Queries



LLM Integration For Generation

Prompt Engineering for Effective Use of Retrieved Context

Input

LLM prompter
incorporates retrieved
information for
input into LLM
generation.

Approach

It is for the
retrieved data to
align with the LLM's
formatting prompt to

Output

LLM model generates
output by taking
retrieved up-to-date
information (e.g.,
latest human news)
prompt retrieved data
document output

LLM Integration For Generation

Balancing Retrieved Information with LLM Knowledge

Searcher: LLMs are
integrated with
external knowledge
bases for
enhancing their
capabilities and
providing relevant
information.

Searcher: LLMs are
integrated with
external knowledge
bases for
enhancing their
capabilities and
providing relevant
information.

Searcher: LLMs are
integrated with
external knowledge
bases for
enhancing their
capabilities and
providing relevant
information.

LangChain Framework



Introduction to LangChain

LangChain is a framework for integrating large language models with external data sources, facilitating retrieval-augmented generation (RAG), a pattern designed for high-quality responses and iterative improvement, simplifying the development and deployment of complex AI applications.

LangChain Framework

Key Features

Modular Design

LangChain offers a modular approach, allowing developers to mix and match components for different use cases, including agents, chains, and vector databases.

Easy Integration

Integrates seamlessly with various LLMs and external services, making it easy to build AI applications.

Integration Capabilities

Seamlessly integrates with various LLMs, vector stores (e.g., Pinecone, ChromaDB), and external services, enabling complex workflows.

Customization

Provides flexible customization options for building unique AI applications, including custom prompts and output parsers.

LangChain Framework

How LangChain Simplifies RAG Implementation



LangChain Framework

LangChain Components



LangChain Framework

LangChain Expression Language (LCEL)

Simple and Powerful Prompt Engineering

Simple

- LCEL is a simple, declarative language for orchestrating LLM applications and integrating various LLMs, LLM providers, prompts, chains, agents, and tools into a single workflow.

Example

```
chain = prompt_template("What is the capital of {country}?") >> llm >> output_parser
```

This example creates a simple chain that takes a country name as input, prompts the LLM, and then returns the capital.

ADVANTAGES

- Simple Syntax: LCEL uses a simple, declarative syntax to define LLM workflows, making it easy to understand and modify.
- Flexible Integration: LCEL integrates with various LLM providers, LLMs, prompts, chains, agents, and tools, allowing for a wide range of use cases.
- Composability: LCEL is designed to be composable, allowing you to build complex workflows by combining simpler components.

LangChain Framework

Benefits of Using LCEL in RAG Pipelines



Streamlined Workflow

- LCEL provides a flexible way to define and chain data processing and retrieval steps, allowing for easy experimentation and iteration.



Improved Readability

- The declarative nature of LCEL makes the flow of data processing and retrieval steps more intuitive and easier to understand, reducing the complexity of the code.



Enhanced Maintainability

- LCEL encourages the use of modular components and clear data flow, making it easier to maintain and update the RAG pipeline as requirements change.



Increased Scalability

- LCEL's modular design makes it easier to integrate and scale different components of the RAG pipeline, allowing for future growth and adaptation.

Demo

Implementing a Simple RAG Pipeline



MODULE 4

Advanced RAG Techniques

Advanced RAG Techniques



Introduction To Advanced RAG Techniques

Advanced RAG Techniques are sophisticated strategies that enhance the performance and accuracy of Retrieval-Augmented Generation systems. These techniques refine the integration of information retrieved and generated models, enabling more accurate and contextually relevant responses. They involve leveraging advanced methods to improve the system's capabilities and reliability in diverse applications.

Key Components Of Advanced RAG Techniques

Multi-Vector Retrieval: Uses multiple vectors for detailed querying representations, improving precision.

Knowledge Graphs: Enhance retrieval by illustrating external relationships between data points, providing context and structure.

Hybrid Retrieval: Combines keyword and vector-based searches for balanced retrieval of relevant content.



Contextual knowledge: Enhances relevance by providing background information and context.

Active Learning: The system learns from user feedback to continuously improve retrieval accuracy.

Reinforced Learning: Enhances the retrieval process by using the improved effectiveness of the system.

Reinforced Learning: Enhances the retrieval process by using the improved effectiveness of the system.

Text Splitting And Chunking Strategies

Importance of Effective Text Splitting

Context Preservation

Proper splitting ensures that the fundamental ideas, meaning, intent, and structure remain coherent and understandable.

Impact on Retrieval Accuracy

Efficient text splitting ensures key details are retrieved without losing important context. Poor splitting can result in incomplete or irrelevant results.

Text Splitting And Chunking Strategies

Balancing Chunk Size with Semantic Coherence



Chunk Size

Smaller chunks tend to preserve context better, but may lose important details. Larger chunks maintain coherence but may include irrelevant info.

Semantic Coherence

Ensure chunk size is large enough to include whole sentences, paragraphs, or sections, if needed, and maintain context.



Different Splitting Techniques

Feature Based Splitting

- Based on the feature value, whether the feature value is used or not.
- Not useful for text data.
- Advantage: It is the most common splitting technique.

Sentence Splitting

- Divides text into sentences using punctuation.
- This is the most common splitting technique.
- Advantage: It is the most common splitting technique.

Paragraph Splitting

- Divides text into paragraphs using punctuation.
- This is the most common splitting technique.
- Advantage: It is the most common splitting technique.

Sentence Splitting

- Divides text into sentences using punctuation.
- This is the most common splitting technique.
- Advantage: It is the most common splitting technique.

Query Transformation And Expansion

Techniques for Improving Retrieval Accuracy



Term Expansion

- Disambiguates cryptic terms with synonyms or related words using thesauri, wordnet or dictionaries.
- Advantages: increases diversity of retrieved objects, results are useful for addressing variations.



Concept Expansion

- Describes the fully related terms or entities to broaden the search results.
- Advantages: increases comprehensiveness by including non-specific or related concepts.



Query Reinterpretation

- Descriptions. Provides instructions for better alignment with search results.
- Advantages: improves precision by matching better, aligning with relevant phrases.

Hypothesis Generation: HyDE Technique

Explanation of Hypothetical Document Embeddings (HyDE)

“

- **Document Embedding:** A numerical representation of a document, often used in information retrieval and recommendation systems.
- **HyDE:** A technique for generating hypothetical document embeddings that are used to improve the performance of information retrieval systems.

”

Hypothesis Generation: HyDE Technique

Implementing HyDE with LangChain

LangChain Overview

- A framework for building applications that integrate language models into the workflow.

Implementation Steps:

1. **Define Embedders**. Use LLMs where text-to-vector embeddings are needed.
2. **Generate Embeddings**. Use LangChain to create these embeddings based on prompt.
3. **Integrate**. Incorporate embeddings into the retrieval system for better search results.

Advantage. HyDE with LangChain enables retrieval by vector representation.

Demo

Comparing retrieval results with
and without query expansion

Re-ranking And Filtering Retrieved Documents

Re-ranking:

re-ranking retrieved results based on additional criteria or importance relevance

- 1. **Contextual Re-ranking:** adjusts relevance based on user context or query history
- 2. **Model-based re-ranking:** uses ML models to predict user interest results by relevance
- 3. **Feedback-based re-ranking:** adjusts rankings using user feedback

Filtering:

filtering results by excluding irrelevant or less quality documents

- 1. **Content-based Filtering:** Filters by content relevance
- 2. **Metadata Filtering:** filters documents by date or author
- 3. **User-defined filtering:** filters results by individual preferences

Using Cross-Encoders For Reranking

Explanation of Cross-Encoders vs. Bi-Encoders



Popular Models: Sentence Transformers, MonoT5

1

Sentence Transformers:

- Description: Used for generating high-quality sentence embeddings, improving similarity tasks.
- Use Case: Fine-tuned for semantic similarity and clustering tasks.

2

MonoT5:

- Description: Employs a single model for both generation and evaluation.
- Use Case: Suitable for text generation and low-resource settings.

3

Benefits:

- Cross-Encoder: Enhanced description understanding for precise matching.
- Bi-Encoders: Efficient and scalable for large-scale systems.

LLM-Based Document Filtering

Using LLMs for Document Relevance

- **Embedding-based LLMs** process documents, generating embeddings for retrieval and relevance scoring, or classification and summarization tasks.
- **Use Case:** Efficiently filter relevant information from large datasets, such as legal or medical documents.

Embeddings & Rule-based Scoring Systems

- **Input Processing:** Feed documents into the LLM.
- **Embedding Generation:** LLM outputs vectors, which are used by scoring models.
- **Threshold & Scoring:** Compare document scores against pre-defined thresholds.
- **Relevance & Ranking:** Sort documents with high scores and relevance.

Demo

- Implementing a reranking pipeline
- Comparing results before and after reranking

Advanced RAG Architectures



Overview

Advanced Retrieval-Augmented Generation (RAG) architectures enhance traditional RAG systems with sophisticated components and methods. They aim to boost accuracy, efficiency, and scalability, making them ideal for various tasks and large-scale applications.

These architectures typically involve:

Multi-Step Retrieval

Coarse-to-Fine Retrieval Strategy

Description: A hierarchical approach where an initial broad search is followed by refined searches that progressively narrow down results, improving efficiency.

Steps:

- **Coarse Retrieval:** Generate a broad set of documents using methods like keyword matching or basic semantics.
- **Fine Retrieval:** Refines the set using advanced techniques like vector search or neural networks for improved precision.

Advantages:

- **Scalability:** Efficiently manages large data volumes by filtering out less relevant documents early.
- **Accuracy:** Combines multiple methods, leading to higher precision in the final results.

Multi-Step Retrieval

Implementing Multi-Step Retrieval with LangChain

LangChain Overview: A framework for building language models, designed for multi-step retrieval.

Implementation Steps:

- 1. Initial Retrieval: Set up initial retrieval process (e.g., using a vector database).
- 2. Iterative Refinement: Iterate with advanced retrieval techniques across multiple steps.
- 3. Final Output: Generate and deliver refined responses.

Advantages:

- 1. Scalability: Easy to adjust steps.
- 2. Efficiency: Optimized and reusable.
- 3. Customization: Further integration of specialized models.

Hybrid Search

Overview: Hybrid search combines vector and keyword retrieval methods to leverage the strengths of both approaches, improving the accuracy and relevance of search results.

Combining Vector and Keyword Retrieval (e.g., Elasticsearch)

- **Indexing:** Documents are indexed using both vector embeddings and keyword metadata.
- **Search:** Queries can combine vector search and keyword search.

Vector Search (e.g., Elasticsearch)

- **Indexing:** Documents are indexed using vector embeddings.
- **Search:** Queries are performed using vector search.

Keyword Search (e.g., Elasticsearch)

- **Indexing:** Documents are indexed using keyword metadata.
- **Search:** Queries are performed using keyword search.
- **Hybrid Search:** Combines vector and keyword search for improved results.

Demo

Building an advanced RAG system with
multi-step retrieval



MODULE 5

Scaling and Deploying RAG Systems

Metrics for Evaluating RAG Performance



Precision

- Definition: Measures the proportion of relevant documents retrieved out of all the retrieved documents.

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

Metrics for Evaluating RAG Performance



Recall

- **Definition:** Measures the proportion of relevant documents that were retrieved out of all the relevant documents available.

- **Formula:**

$$\text{Recall} = \frac{\text{Relevant Documents Retrieved}}{\text{Total Relevant Documents Available}}$$

Metrics for Evaluating RAG Performance



F1-Score

- Combines both Precision and Recall, providing a balanced measure of model performance.

Formula:
$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Metrics for Evaluating RAG Performance



Mean Reciprocal Rank (MRR)

- Defines MRR as a metric for evaluating systems that return ranked lists of results. It measures the inverse of the positional rank of the first relevant result.

- Formula: $MRR = \frac{1}{N} \sum_{i=1}^N \frac{1}{rank_i}$

Metrics for Evaluating RAG Performance

BLEU (Bilingual Evaluation Understudy)

- **Unigram (n=1)** measures how many n-grams (usually n=1 words) in the generated text match those in the original text.
- **Perplexity BLEU** is calculated based on precision over different n-gram sizes, weighted with a lambda penalty for uncalibrated unigrams.
- **Not used** (historically used in machine translation, a metric measure how similar the generated content is to the reference in terms of repeating exact words or phrases). However, it can be helpful when you're looking for exact matches.

Metrics for Evaluating RAG Performance

ROUGE (Recall-Oriented Understudy for Gisting Evaluation)

- 1. **Recall-based Metric:** Measures the overlap between the generated text and the reference text, with a focus on recall (capturing all relevant information). The score ranges from 0 to 1.
- 2. **Formula for ROUGE-n:**
$$\text{ROUGE-n} = \frac{\text{Number of n-grams in generated text overlapping with n-grams in reference text}}{\text{Total number of n-grams in generated text}}$$
- 3. **Limitations:** Does not consider the quality or relevance of the generated text, only the overlap.
- 4. **Use Cases:** Useful for evaluating the model's ability to capture the important aspects of the reference text.

Metrics for Evaluating RAG Performance

METEOR (Metric for Evaluation of Translation with Explicit ORdering)

- METEOR algorithm is designed to measure the quality of machine-generated translations by considering synonyms, stemming, and paraphrasing, along with exact matches. It adjusts for precision and recall.
- Formulas calculate measures not just in grams, but also word order and syntactic structures, requiring a more complex algorithm that needs more resources for processing.
- Use Case: This metric is useful when you're interested in capturing meaning rather than just exact matches. It is often used in machine translation and content quality testing.

Metrics for Evaluating RAG Performance

ReProScore

- **Definition:** ReProScore uses contextual embeddings from GPT-3 to measure the semantic similarity between the generated and reference texts, considering meaning over word overlap.
- **How it Works:** It computes the cosine similarity between embeddings of the words in the generated and reference texts to measure how similar they are in meaning.
- **Advantages:** It captures the underlying meaning and context in text, like syntactic structure and lexical grammar, distribution patterns and statistics, or makes the model and feedback by focusing on semantic similarity.

RAG-specific metrics

Faithfulness



RAG-specific metrics

RELEVANCE

Definition

Relevance measures the quality of the retrieved documents and how well they contribute to answering the user's query. It emphasizes how closely the documents match the user's intent, how helpful they are for processing the final answer.

User Cases

Relevance considerations are crucial for improving the quality of the retrieved documents, as it directly impacts the quality and usefulness of the generated responses.

Introduction to RAGAS Framework

Overview of RAGAS

The RAGAS (Retrieval-Augmented Generation Assessment) framework is designed to evaluate the performance and quality of Retrieval-Augmented Generation (RAG) models. RAG models utilize external knowledge sources, such as documents or databases, to generate responses. RAGAS provides a comprehensive evaluation framework to assess the quality of the generated responses, ensuring that the information retrieved from the external sources is accurate and relevant.

Introduction to RAGAS Framework

Purpose and Key Features

Evaluate Retrieval-Augmented Generation (RAG) systems for relevance, accuracy, and faithfulness.

Key Features:

- Ranking system to assess relevance and granularity
- Consistent retrieval quality and generation accuracy

Introduction to RAGAS Framework

Integration with LangChain



RAGAS evaluation metrics



Answer Relevancy
Measures how relevant the generated answer is to the query.



Context Inclusion
Assesses if the retrieved context aligns with the query.



Factuality
Evaluates if the generated response accurately reflects the retrieved information.



Source Faithfulness
Evaluates the system's ability to include the cited sources in the generated response.

Creating AI-augmented test sets

Techniques for generating diverse test questions



Random generation

Generate questions by randomly sampling from a large pool of pre-generated questions, ensuring diversity through randomization.



Rule-based generation

Use predefined rules and templates to generate questions, ensuring diversity through systematic variations.



AI-driven generation

Use AI models to generate questions, ensuring diversity through prompt engineering and model diversity.

Creating AI-augmented test sets

Ensuring coverage of different question types and difficulties

- **Question Type Distribution:** The test should include various question types, such as multiple choice, short answer, and essay questions.
- **Answer Length Distribution:** The test should include questions with different answer lengths, ensuring that the test covers a range of question types.

- **Difficulty Level Distribution:** The test should include questions of varying difficulty levels, ensuring that the test covers a range of question types.
- **Strategy:** The test should include questions that require different strategies, such as multiple choice, short answer, and essay questions.

- **Answer Length Distribution:** The test should include questions with different answer lengths, ensuring that the test covers a range of question types.
- **Strategy:** The test should include questions that require different strategies, such as multiple choice, short answer, and essay questions.

Optimizing RAG Pipelines

Fine-tuning strategies for embeddings

- **Transfer Embeddings**: Leverage pre-trained models to initialize embeddings, reducing the need for extensive fine-tuning.
- **Adaptive Embedding**: Fine-tune embeddings dynamically based on specific domain or task requirements, improving relevance.
- **Parallel Embedding**: Utilize distributed computing and GPUs to accelerate the fine-tuning process for large datasets.

Optimizing RAG Pipelines

Fine-tuning strategies for embeddings

- **Triplet Learning:** Enhance general embeddings by a specific domain using domain-specific data.
- **Supervised Contrastive Learning:** Encourage the model to pull together similar samples and push apart dissimilar ones.
- **Domain-specific Embedding Refinement:** Refine general embeddings with domain-specific information.

Optimizing RAG Pipelines

Contrastive learning approaches

- **Contrastive learning** (used in Retrieval-Augmented Generation) involves training models to distinguish between relevant and irrelevant information.
- **Relevance-based contrastive learning** (used in Retrieval-Augmented Generation) involves training models to distinguish between relevant and irrelevant information.
- **Similarity-based contrastive learning** (used in Retrieval-Augmented Generation) involves training models to distinguish between relevant and irrelevant information.

Optimizing RAG Pipelines

Contrastive learning approaches

- **InfoNCE** (Information Noise Contrastive Estimation) is a popular choice for contrastive learning. It is used to learn a model that can distinguish between relevant and irrelevant information.
- **InfoNCE** is a loss function that is used to train a model to distinguish between relevant and irrelevant information. It is used to learn a model that can distinguish between relevant and irrelevant information.
- **InfoNCE** is a loss function that is used to train a model to distinguish between relevant and irrelevant information. It is used to learn a model that can distinguish between relevant and irrelevant information.

Optimizing RAG Pipelines

LLM optimization for RAG

System Engineering

LLM optimization is not a one-size-fits-all solution. It requires a deep understanding of the specific use case and the underlying hardware and software constraints.

Techniques

Model Quantization: Reduces the precision of the model's weights, allowing it to run on lower-end hardware. This can significantly reduce memory usage and inference time, but may also lead to a loss of accuracy.

Model Distillation: Involves training a smaller model to mimic the behavior of a larger, more complex model. This can result in a more efficient model that maintains a high level of performance.

Hardware Acceleration: Utilizing specialized hardware like GPUs or TPUs can significantly speed up the inference process. This is particularly important for large-scale applications where performance is critical.

Optimizing RAG Pipelines

LLM optimization for RAG

Specialized Hardware

Use of specialized hardware accelerators like GPUs or TPUs to improve performance of LLM inference, reducing latency and increasing throughput.

Techniques

- **Quantization:** Reducing the precision of LLM weights and activations to lower memory usage and improve inference speed, often achieved through specialized hardware or software libraries.
- **Knowledge Distillation:** Transferring knowledge from a large, complex LLM to a smaller, more efficient model, often achieved through specialized hardware or software libraries.
- **Hybrid Retrieval:** Combining different retrieval methods (e.g., vector search and keyword search) to improve recall and precision, often achieved through specialized hardware or software libraries.

Optimizing RAG Pipelines

Hyperparameter optimization

Embedding Model

- Embedding model is used to convert documents to vectors
- Impact of embedding model on retrieval quality and overall system performance is high
- Embedding model is a key component of the RAG pipeline

Retrieval Model

- Retrieval model is used to find relevant documents from the vector database
- Impact of retrieval model on retrieval quality and overall system performance is high
- Retrieval model is a key component of the RAG pipeline

Generation Model

- Generation model is used to generate the final answer based on the retrieved documents
- Impact of generation model on generation quality and overall system performance is high
- Generation model is a key component of the RAG pipeline

Optimizing RAG Pipelines

Using Bayesian Optimization for Parameter Tuning

Define Objective Function

Set up a function to evaluate the performance of the RAG system based on metrics like accuracy, recall, and F1 score.

Initialize Model Space

Define the range of values for the parameters you want to optimize, such as embedding dimensions, chunk size, and number of retrievers.

Parameter Initialization

Randomly select parameters from the defined ranges to create an initial set of configurations for testing.

Bayesian Optimization Loop

Iteratively evaluate configurations, update the model space based on performance, and select the next configuration to test using Bayesian optimization.

Demo

Evaluating and Optimizing a RAG System



MODULE 6

Scaling and Deploying RAG Systems

Scaling And Deploying RAG Systems

Efficient Indexing Strategies

Approximate Nearest Neighbor (ANN)

Searches efficiently find similar vectors in high-dimensional spaces with methods like HNSW or LSH.

Inverted Indexes: Quickly locate documents by terms, ideal for text-based queries.

Hierarchical Indexing: Use multi-level indexes to speed up retrieval by narrowing down from a coarse index first.

Embedding Compression: Store dense embeddings more efficiently using quantization or sparse matrices.

Scaling And Deploying RAG Systems

Incremental Updating of Vector Stores



Scaling And Deploying RAG Systems

Load Balancing and Caching Techniques

Load Balancing: Distributes incoming traffic across multiple servers to prevent any single server from becoming a bottleneck, ensuring consistent performance and availability.

Cache Invalidation: Ensures that cached data remains accurate by implementing strategies to update or remove stale information, preventing outdated responses.

Database Optimization: Reduces query execution time through techniques like indexing, partitioning, and efficient database schema design, improving overall system responsiveness.

Horizontal Scaling: Increases system capacity by adding more servers to the pool, allowing the system to handle increased load and maintain high availability.

Scaling And Deploying RAG Systems

Asynchronous Processing for Improved Throughput



Message Queue and Pub/Sub for asynchronous request handling to improve scalability and reliability.



Parallel Query Execution with cache or LRU management for relevant documents.



Event-driven Architecture for scaling for asynchronous processing, integrating with event-driven systems.



Rate Limiting to limit the number of requests per user or service to prevent overload and ensure fair use.

Scaling And Deploying RAG Systems

Scaling Retrieval with Distributed Vector Databases

01

Vector Databases: Use distributed databases like Milvus, Redis, or PostgreSQL for high scalability, handling and storing large vector datasets.

02

Sharding: Partition data across multiple servers or nodes to prevent bottlenecks and ensure consistent scaling.

03

Replication: Copy data across multiple nodes to ensure high availability and fault tolerance.

Scaling And Deploying RAG Systems

Parallel Processing of Queries and Document Chunks



Query Parallelization: Distribute queries across multiple processors to reduce latency and handle user interactions.



Document Chunking: Split documents into smaller chunks for parallel processing and efficient retrieval.



Load Balancing: Distribute workload across multiple servers to avoid bottlenecks and optimize resource usage.

Scaling And Deploying RAG Systems



Overview of relevant AWS services (EC2, ECS, Lambda)

AWS EC2, ECS, Lambda, Amazon Cloud

- **Amazon EC2** offers scalable virtual servers for running applications and services.
- **Amazon ECS** provides a managed container service for deploying, managing, and scaling your applications using Docker containers.
- **Amazon Lambda** is a serverless compute service that executes your code in response to events and automatically manages the underlying compute resources.

Scaling And Deploying RAG Systems



Overview of relevant AWS services (EC2, ECS, Lambda)

AWS EC2 (Elastic Compute Service)

1. **Infrastructure as a Service (IaaS)**: Provides virtual machines (EC2 instances) that can be scaled up or down based on demand.
2. **Flexibility and Control**: Offers full control over the operating system and software stack.
3. **Scalability**: Supports both manual scaling and automatic scaling (Auto Scaling) to handle varying workloads.

Scaling And Deploying RAG Systems



Overview of relevant AWS services (EC2, ECS, Lambda)

AWS Lambda

- 1. **Serverless** architecture in response to demand with automatic resource management.
- 2. **Low cost** (paid for event driven tasks and higher light than on-premises)
- 3. **Scaling** automatically grows with incoming requests, handling tasks in parallel

Scaling And Deploying RAG Systems



CI/CD Pipelines for RAG Systems

CI/CD pipelines automate the integration and deployment of code changes, ensuring that the RAG system is always up-to-date and running smoothly. This is crucial for maintaining the accuracy and reliability of the system, especially in a production environment.

Scaling And Deploying RAG Systems

Continuous Integration (CI)



Scaling And Deploying RAG Systems

Continuous Deployment (CD)

Automated Deployment

Use tools like Jenkins, GitHub Actions, or ArgoCD to automate the deployment pipeline.

Rolling Updates

Deploy updates without downtime by gradually replacing old versions with new ones.

Blue/Green Deployment

Implement two identical environments to allow for quick rollback in case of issues.

Scaling And Deploying RAG Systems

Version Control for Documents and Embeddings

Version
Control
Systems



```
graph LR; A((Version Control Systems)) --- B[can manage and manage documents with the help of various tool and resources for updates.]; A --- C[can also manage and manage embeddings with the help of various tool and resources for updates.];
```

can manage and manage documents with the help of various tool and resources for updates.

can also manage and manage embeddings with the help of various tool and resources for updates.

Scaling And Deploying RAG Systems

Version Control for Documents and Embeddings



Scaling And Deploying RAG Systems

Version Control for Documents and Embeddings



Scaling And Deploying RAG Systems

Monitoring and Maintaining Deployed Models



Scaling And Deploying RAG Systems

Setting Up Monitoring Dashboards

Use tools like Prometheus and Grafana to monitor system health and performance. Set up alerts for critical metrics.

Implement logging for system events and errors. Use tools like ELK Stack for log management.

Regular Reviews: Conduct periodic reviews and updates to monitoring tools and dashboards to reflect changes in the system architecture.



Monitor system health metrics like CPU usage, memory usage, and network I/O. Set up alerts for critical thresholds.



Define Metrics: Log metrics relevant to system performance and health. Use tools like Prometheus and Grafana.



Create Dashboards: Design dashboards to visualize key metrics. Use tools like Grafana for dashboard creation.



Scaling And Deploying RAG Systems

Strategies for Keeping the Knowledge Base Current

Batch Refreshes: Run periodic ingestion pipelines to update the knowledge base with new data from various sources.

Version Control: Track data versions for change management and rollback capabilities in case of errors.

Cost Control: Regularly audit and optimize data storage and processing costs.



Scaling And Deploying RAG Systems

When to Update LLMs and Embedding Models

Vector Pruning: Remove outdated vectors to maintain efficiency and relevance.

Re-Indexing: Rebuild or fine-tune models for better vector accuracy.

Versioning: Track updates to the LLMs and the database with change management.



Scaling And Deploying RAG Systems

When to Update LLMs and Embedding Models



Performance Degradation: If accuracy drops or errors increase, update the model with new data or improvements.



Data Drift: Changes in data sources, trends, or behavior over time, requiring the model.



New User Requirements: As user needs evolve, update the model to handle changing and complex.



Model Enhancements: Integrate better algorithms or techniques to improve performance.



User Feedback: Update the model based on user feedback to ensure relevance and satisfaction.

Scaling And Deploying RAG Systems

How to Update LLMs and Embedding Models

Model Training

- **Apprentices:** Develops a working prototype model.
- **Provisional Training:** Fine-tunes the model on a specific dataset to refine and improve.

Incremental

- **Apprentices:** Focuses on incremental learning of new data.
- **Provisional Training:** Fine-tunes the model on the updated dataset to learn new patterns.

Incremental Deployment

- **Apprentices:** Focuses on the incremental deployment of new models.
- **Provisional Training:** Fine-tunes the model on the updated dataset to learn new patterns.

Deployment and Monitoring

- **Apprentices:** Focuses on the deployment of new models.
- **Provisional Training:** Fine-tunes the model on the updated dataset to learn new patterns.

Scaling And Deploying RAG Systems

A/B Testing New Model Versions

Define Objectives

- **Primary:** Identifying key evaluation metrics that accurately reflect model performance
- **Secondary:** Comparing new model version against baseline model across various metrics

Design Experiment

- **Randomized User Allocation:** Dividing users into groups for A/B testing. Ensure each group receives consistent treatment.
- **Control:** Baseline model (previous version) for comparison.

Deploy Models

- **Deployment:** Deploying both models in a controlled environment using feature flags or gradual rollout strategy.

Scaling And Deploying RAG Systems

A/B Testing New Model Versions

Make Decisions

- **Evaluation:** Decide whether to fully deploy, experiment, or prepare for next model.
- **Statistical Analysis:** Use model-based and regression analysis and results.

Analyze Results

- **Performance Metrics:** Analyze accuracy, precision, recall, and F1 score.
- **Statistical Testing:** Determine significance of performance differences.

Collect Data

- **Logging:** Track user interactions, performance, and error feedback for data analysis.

Demo

Deploying a RAG System



MODULE 7

Real-World RAG Applications and Future Trends

Real-World RAG Applications And Future Trends

Implementing RAG for Corporate Knowledge Bases



Real-World RAG Applications And Future Trends

Implementation Steps



Data Integration

- Unified data sources into a single knowledge base



RAG Model Setup

- Fine-tuned LLM and vector embeddings with relevant documents



Search Interface

- Developed an intuitive search interface tailored to the RAG model



Testing & Optimization

- Customized metrics for system performance



Deployment & Training

- Deployed system and trained employees

Real-World RAG Applications And Future Trends

Outcome

Improved Efficiency

Streamlined data retrieval and processing, significantly reducing manual effort and increasing productivity and cost-effectiveness.

Enhanced Accuracy

Reduced human error and improved data integrity, ensuring consistent and reliable information across the organization.

Better Decision-Making

Providing timely and accurate insights, enabling faster and more informed decision-making and strategic planning.

Real-World RAG Applications And Future Trends

CI/Demo: Building an Advanced Enterprise Search System with RAG

YOUNG ZHANG



Scaling And Deploying RAG Systems



Step 1: Data Collection and Preprocessing

- Aggregate data from various internal systems, external databases, documents, management systems, and user feedback.
- Clean and preprocess the data to ensure consistency and quality.

Scaling And Deploying RAG Systems



Step 2: Setting Up The Retrieval System

- Use a vector database like Pinecone or FAISS to store documents, but make them searchable.
- Create embeddings for the documents using a pre-trained transformer model like BERT or GPT.

Scaling And Deploying RAG Systems



Step 3: Fine-Tuning the LLM

1. Select a large language model (LLM) that is suitable for the company's specific needs and goals.
2. Gather the relevant data and documents that the model will be trained on.
3. Fine-tune the model using the gathered data, ensuring that the model understands the company's specific context and requirements.

Scaling And Deploying RAG Systems



Step 4: Integrating RAG

- Connect the Retrieval Layer with the Retrieval System
- Set up a deployment-ready environment and test the retrieval system thoroughly, ensuring that it can generate responses based on the retrieved data.

Scaling And Deploying RAG Systems



Step 5: Building the User Interface

1. Develop a user interface that allows users to interact with the system and view the results of their queries.
2. Implement a search bar and filters to help users find the information they need.
3. Design a clean and intuitive layout that is easy to navigate.
4. Implement a chatbot or assistant to help users with their queries.
5. Implement a feedback mechanism to allow users to provide input on the system's performance.

Scaling And Deploying RAG Systems



Step 4: Testing and Feedback

- Conduct testing with a small group of users to identify any issues or areas for improvement.
- Monitor the design and model performance based on user feedback.

Scaling And Deploying RAG Systems



Step 7: Deployment

- Deploying the system on the company's internal network
- Ensuring it is scalable and can handle the increased query load, as deployment

Scaling And Deploying RAG Systems

Enhancing Customer Support Chatbots with RAG Capabilities



Scaling And Deploying RAG Systems

Enhancing Customer Support Chatbots with RAG Capabilities



Real-World RAG Applications And Future Trends

Content Generation and Summarization Using RAG

- **Real-World Application**
Mainstream large language models (LLMs) with prompts are designed for general-domain, unstructured content.
- **Applications in Content Creation**
RAG generates articles or reports by retrieving relevant data from databases or documents. This is using LLMs to generate content based on retrieved data.
- **Summarization Technology**
RAG retrieves key facts or insights from a large corpus of data for summarizing long-form content, while reducing content.

Real-World RAG Applications And Future Trends

Long-Form Content Summarization Techniques

Abstractive Summarization

Generates new sentences summarizing key points, resulting in more fluid summaries but requiring tuning.

Hybrid Approaches

Combines extractive and abstractive methods for accurate snippets and well-structured summaries.

Extractive Summarization

Selects key sentences from the text, preserving original wording but may miss context.

Use of RAG

RAG enhances summarization by retrieving relevant references and context, improving the quality and coherence of the results.

Real-World RAG Applications And Future Trends



Coordinating Multiple LLMs for Complex Tasks

Multi-agent systems in the context of Retrieval-Augmented Generation (RAG) involve using multiple Large Language Models (LLMs) to collaboratively solve complex tasks. These systems are designed to leverage the strengths of different models or specialized agents to achieve more effective and accurate responses.

Coordinating Multiple LLMs For Complex Tasks



Implementing a Simple Multi-Agent RAG System



Agent Definition

Assign roles to multiple agents, such as information discoverers, summarizers, and generators.



Communication Protocol

Set up how agents and external tools using individuals like message passing or a centralized coordinator.



Retrieval Mechanism

Enable agents to access external data via a Retrieval-Augmented Generation (RAG) framework.



Coordination Strategy

Design a rule-based or ML-driven system to manage the sequence of agent actions.



Evaluation and Optimization

Analyze performance through metrics like accuracy, and response time, then fine-tune agents for better results.

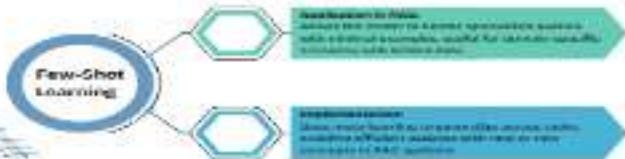
Continuous Learning And Adaptive Retrieval In RAG Systems

Techniques for Dynamically Updating Knowledge Base.



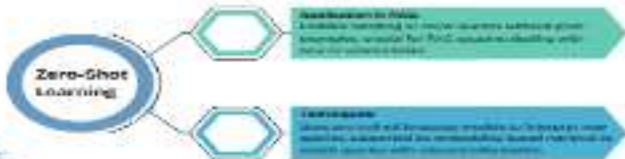
Continuous Learning And Adaptive Retrieval In RAG Systems

Exploration of Few-Shot and Zero-Shot Learning in RAG



Continuous Learning And Adaptive Retrieval In RAG Systems

Exploration of Few-Shot and Zero-Shot Learning in RAG



Continuous Learning And Adaptive Retrieval In RAG Systems

Exploration of Few-Shot and Zero-Shot Learning In RAG

Adaptive Retrieval in Few-Shot and Zero-Shot Scenarios:

- The retrieval component of a RAG system can be adapted to better support few-shot and zero-shot learning by using (or adapting) techniques that are robust to variations in the data. For example, embedding-based retrieval methods, which represent both queries and documents in a shared latent space, allow the system to retrieve relevant information even when dealing with previously unseen queries.

Ethical Considerations And Responsible AI

Bias Mitigation in IFAC Systems

Identifying and Addressing Bias

- Conduct regular audits to assess bias in the system, identifying areas where the model may be exhibiting bias or unfairness.
- **Assessment**
 - Bias assessment: Regularly evaluate the system for bias across different groups, such as gender, age, and ethnicity.
 - Transparency: Explain the model's decision-making process to stakeholders, highlighting areas of potential bias.
 - Bias mitigation: Use techniques like data augmentation, model regularization, and fairness constraints to reduce bias.
 - Monitoring: Continuously monitor the system's performance across different groups to detect and address bias.

Implementing Bias Mitigation Strategies in IFAC Systems

- **Assessment**: Regularly evaluate the system for bias across different groups, such as gender, age, and ethnicity.
- **Transparency**: Explain the model's decision-making process to stakeholders, highlighting areas of potential bias.
- **Bias mitigation**: Use techniques like data augmentation, model regularization, and fairness constraints to reduce bias.
- **Monitoring**: Continuously monitor the system's performance across different groups to detect and address bias.

Ethical Considerations And Responsible AI

Ensuring Transparency and Explainability

Challenges for Developing Explainable AI

- **Model Complexity:** Deep learning models are often highly complex, making it difficult to interpret the internal workings of the model.
- **Black-box Nature:** Many AI models, particularly those based on neural networks, operate as "black boxes," meaning their internal logic and decision-making process are not transparent or understandable.

Developing More Explainable AI Models

- **Interpretable Algorithms:** Research is ongoing to develop algorithms that are inherently more interpretable, such as using decision trees or rule-based systems.
- **Explainability-Oriented Frameworks:** Frameworks like SHAP (Shapley values) and LIME (Local Interpretable Model-agnostic Explanations) provide methods for explaining the output of complex models.

Discussion And Brainstorming Session

Identifying members and add members in the discussion community

- Identify members who are
not members of
social media networks or
etc.
- Identify members who
are not in the community
and are not members
of the community
and are not members
of the community
and are not members
of the community

Identifying who is not a member of the community

- Identify members who
are not members of the
community and are not
members of the community
and are not members
of the community
- Identify members who
are not members of the
community and are not
members of the community
and are not members
of the community

Identifying who is not a member of the community

- Identify members who
are not members of the
community and are not
members of the community
and are not members
of the community
- Identify members who
are not members of the
community and are not
members of the community
and are not members
of the community

**THANK
YOU!**

