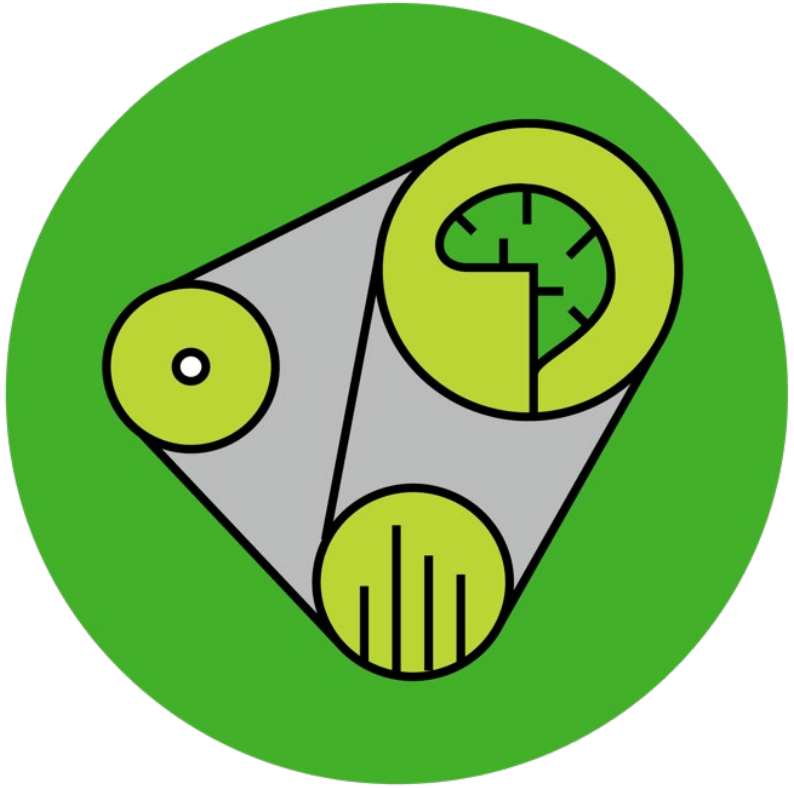
Deloitte.



AI Guild | GenAI Practicum

Retrieval Augmented Generation

Week Agenda

Topic	Content
01 Introduction	• Intro to RAG • Embeddings
02 RAG	• Vectors, Chunking • Retrieval, Generation
03 Vector Databases	• Retrieval • Generation
04 Build	• Put it all together to build a RAG App

Section 1 - RAG

Learning objectives

By the end of session, you should be able to

01 Understand and articulate the process of augmenting LLMs with custom data, including the creation of embedding databases and embedding frameworks.

02 Describe the RAG architecture, its components, and the technical steps involved in its implementation, including data chunking and vector conversion.

03 Explain the principles and techniques of chunking, embeddings, and vector databases in RAG, and their roles in enhancing retrieval and generation processes.

04 Implement a basic RAG system, including the design of effective prompts, and demonstrate its application in a real-world scenario, such as developing a document assistant.

Introduction to Augmenting LLMs

Large Language Models

- Trained on massive text data.
- Understands language with meaning and context.

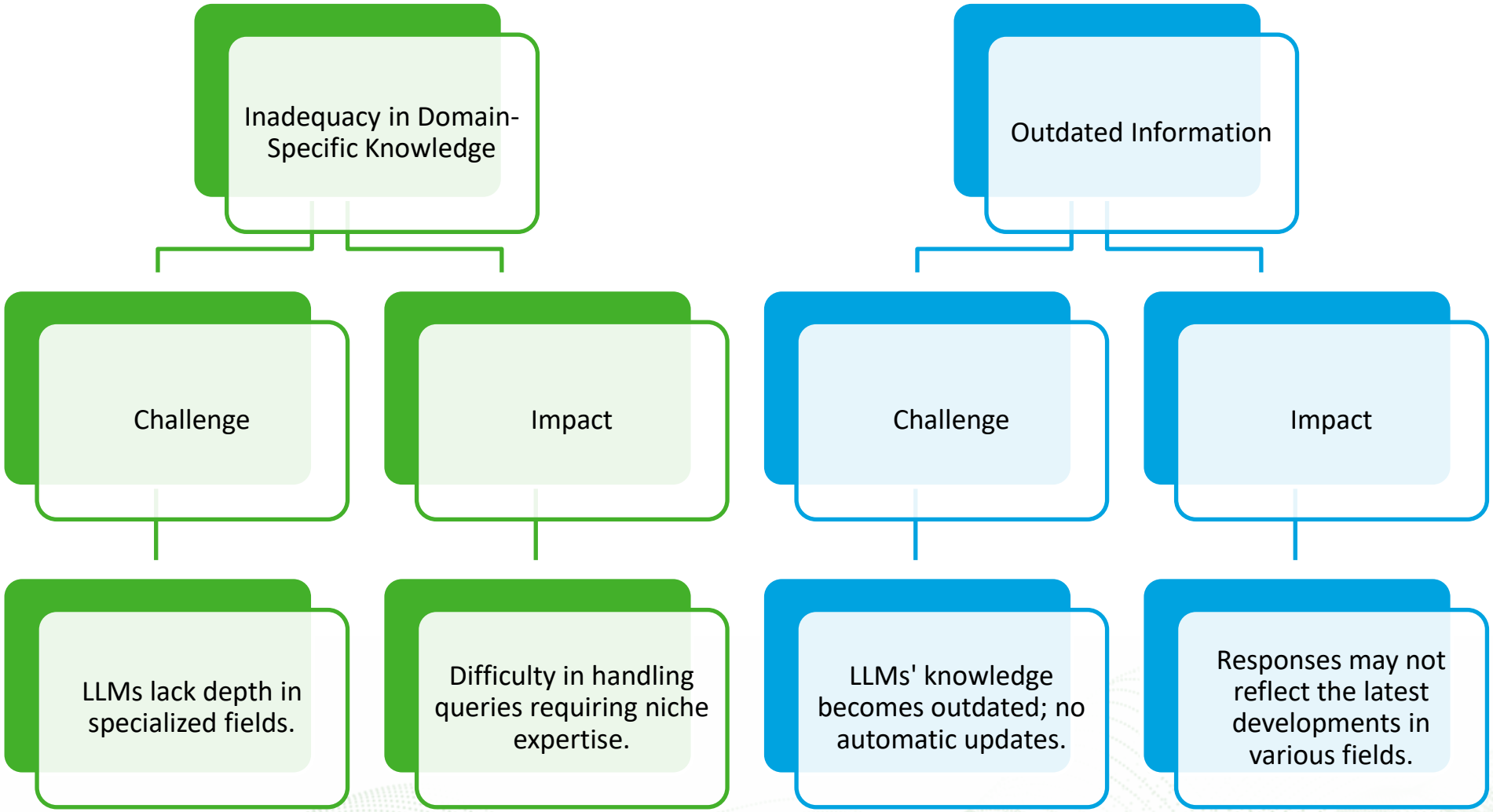
Why Fine-Tune LLMs?

- For domain-specific tasks.
- Recognize niche terminologies.
- Example: GPT-3 for NLU tasks, BERT for question answering.

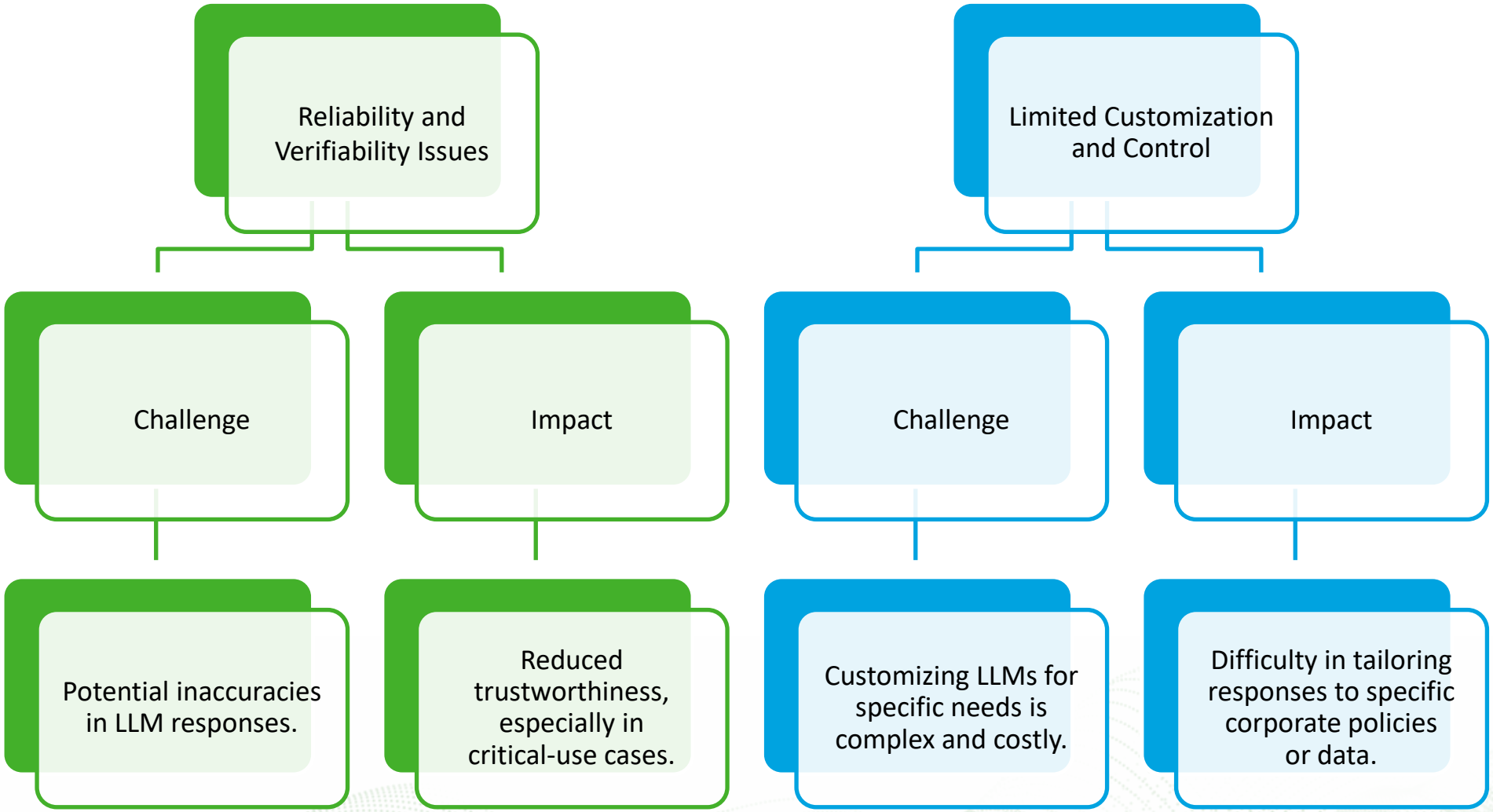
Techniques for Augmentation and Contextual Responses

Semantic Capture and Embedding Databases

Why do we need Retrieval Augmented Generation?



Why do we need Retrieval Augmented Generation?



What is RAG? - Defining the Framework

1. **RAG:** An AI Framework that retrieves facts from an external knowledge base to ground LLMs on accurate and current information, enhancing user insights into the LLM's generative process

2. **Enhancement of LLMs:** Augments capabilities by retrieving facts from external sources.

3. **Bridging Knowledge Gaps:** Connects generalized LLM knowledge with specific, real-time information.

What is RAG? - Significance in Generative AI

Improving

Improving Accuracy and Relevance

- Access to Current Data:
- Tailored Responses:

Building

Building Trust and Reducing Misinformation

- Source Citation:
- Decreased Errors:

What is RAG? - Practical Implications and Adaptability

Real-time Data Integration: Equips LLMs with capabilities to access various up-to-date data sources.

Dynamic Information Access

Adaptability: Allows for greater customization to industry-specific requirements.

Customization for Specific Needs

The Evolution of RAG

Early Foundations and the Machine Learning Shift

Early Rule-Based Systems

- Initial NLP approaches based on fixed rule sets.
- Limited ability in natural language understanding and generation.

Shift to Machine Learning

- Integration of machine learning for dynamic language processing.
- Enhanced capability to learn from data, surpassing rule-based limitations.

The Evolution of RAG

The Rise of Transformer Models

Impact of Transformer Models

- Introduction of BERT and GPT models revolutionizing NLP.
- Unprecedented depth in language processing and generation.

Limitations of Transformers

- Challenges in accessing up-to-date, domain-specific knowledge.
- Inability to dynamically incorporate external information.
- Instructions specific fine-tuning and inferences.

The Evolution of RAG

The Advent of RAG

Need for Dynamic Knowledge

- Increasing demand for real-time, specific-domain knowledge integration.
- Recognition of limitations in static, pre-trained models.

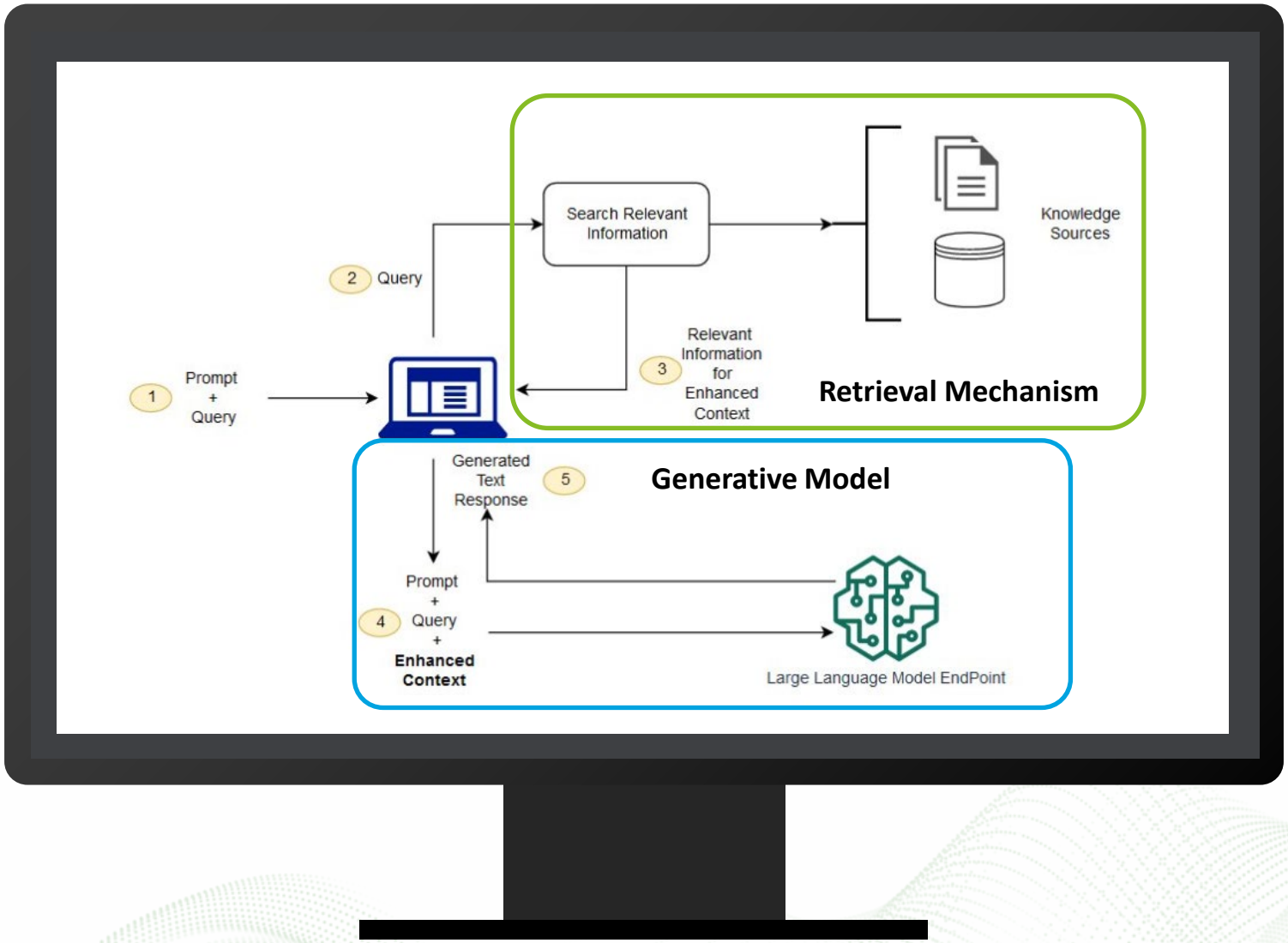
Development of RAG

- Conception of RAG to augment LLMs with external data sources.
- Enhanced accuracy, relevance, and applicability in AI responses.

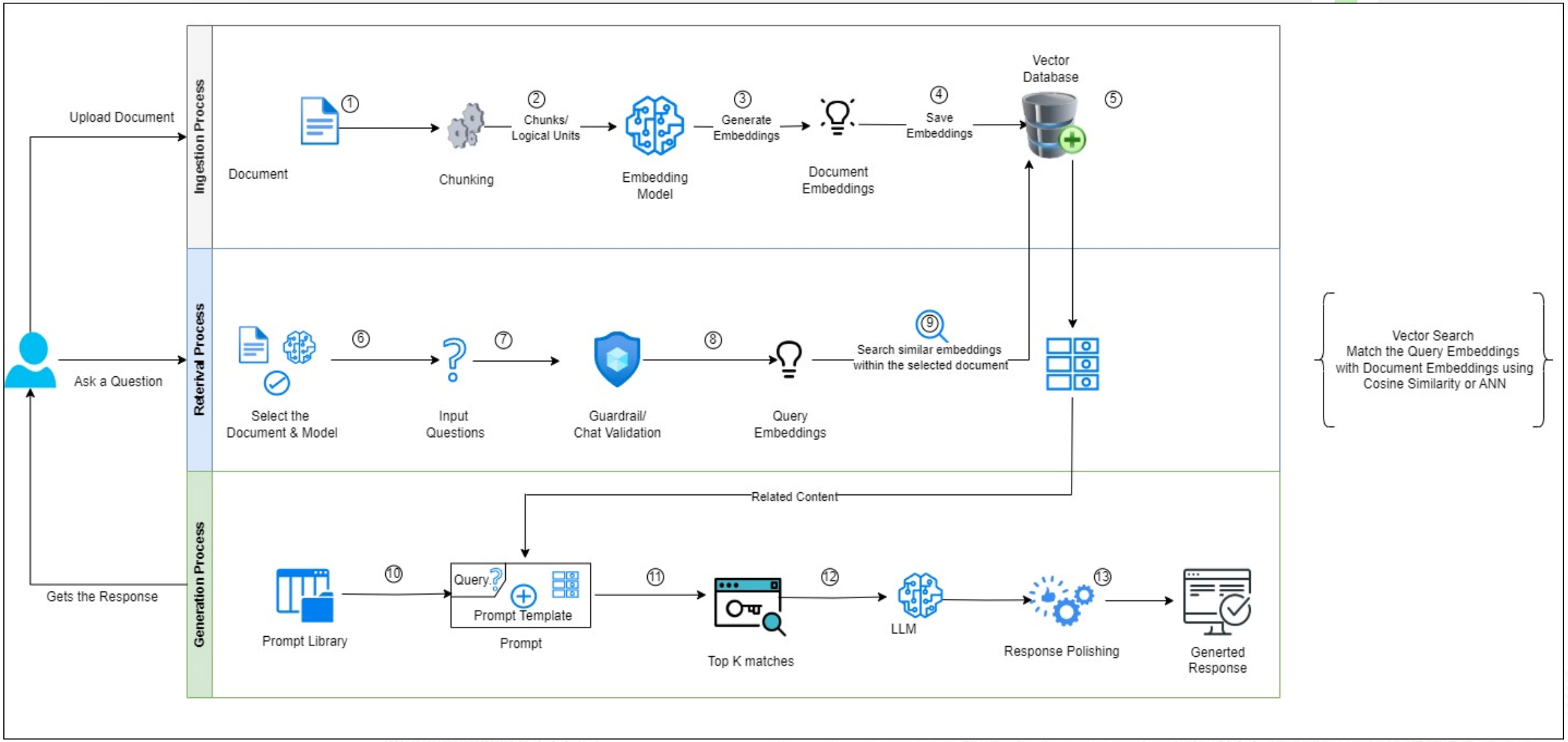
Technological and Research Milestones

- Advancements in machine learning algorithms and computational power.
- Ongoing research and collaborations fueling RAG's evolution.

Basic Architecture of RAG



General Flow of a RAG Application



Basic Architecture of RAG

Overview

Combining Two Core Components

Retrieval Mechanism:

Sources relevant information from external databases.

Generative Model:

Uses a pre-trained language model to generate responses.

Basic Architecture of RAG

The Retrieval Mechanism

Indexing and Query Capabilities:

Organizes and retrieves data relevant to user queries.

Integration with Language Models:

Feeds relevant external information to the generative model.

Basic Architecture of RAG

The Generative Model

Data Synthesis:

Merges retrieved data with the model's internal knowledge.

Crafting Responses:

Generates contextually accurate responses incorporating external information.

What is RAG? - Practical Implications and Adaptability

Security and Reliability:

Vital for handling sensitive data.

Scalability and Efficiency:

Key for managing large data volumes and providing timely responses.

RAG Architecture Pattern - Technical Steps in RAG Implementation

1

Data Chunking
Efficient segmenting for quick retrieval

2

Text-to-Vector Conversion
Transforming text into embeddings for retrieval model use

3

Linking Data and Embeddings
Ensuring relevance and accuracy in information retrieval

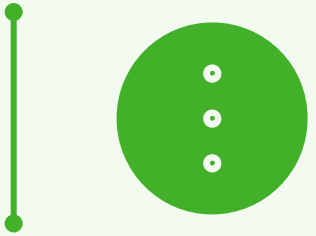
RAG Architecture Pattern – Technical Steps in RAG Implementation

1

Role as Gatekeepers
Search and find relevant information in large data sets

2

Implementation Techniques
Use of vector embeddings, vector search, and document indexing (BM25, TF-IDF)



3

Importance
Introduce external knowledge for context-rich language generation

Components of RAG - Generative Models

1

Function as 'Creative Writers'
Synthesize retrieved information into coherent text.

2

Capabilities
Built on LLMs; ensure grammatical and semantic correctness

3

Contribution
Transform raw data into a narrative structure for easy understanding

Components of RAG – Interaction Between Components

Synergistic Functionality

Retrieval models provide factual content; generative models compose coherent language

Collaborative Output

Generates contextually relevant and information-enriched responses for advanced NLP applications

Understanding Chunking in RAG

Chunking

- Breaking down text into smaller segments.

Importance of Chunking

- Enhances accuracy in matching queries with content.
- Reduces noise in the retrieval process.

Strategies for Chunking

Fixed-size Chunking

- Dividing text into chunks of a specific character count

Recursive Chunking

- Splitting text by character count until chunks are sufficiently small

Lang Chain's ParentDocumentRetriever

- Maintaining links to original documents for detailed retrieval

Introduction to Embeddings in RAG

Embeddings transform textual data into dense vector representations using transformer decoders.

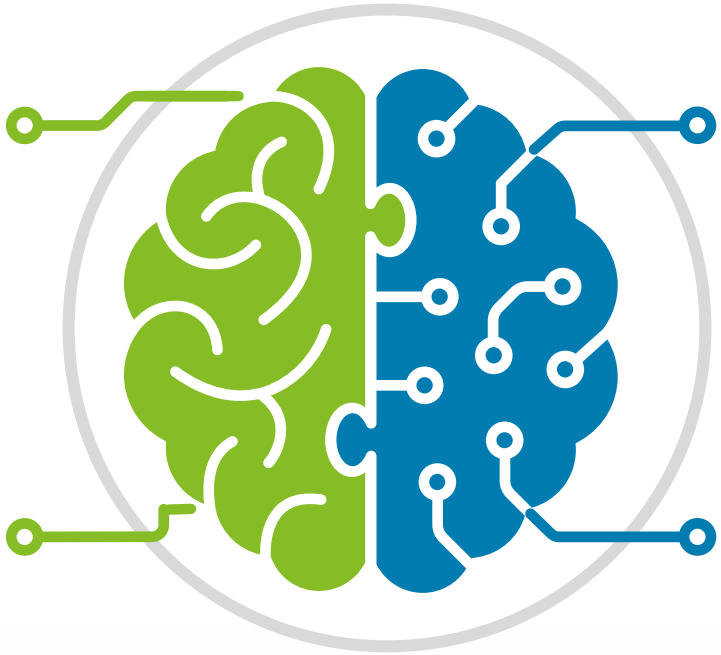
Facilitate semantic understanding of text by the system; optimize query-passage similarity.

Essential for encoding queries and passages independently, enhancing text retrieval efficiency.

Embedding Process in RAG

Breaking down large data sets into manageable pieces.

Converting text chunks into numerical embeddings.



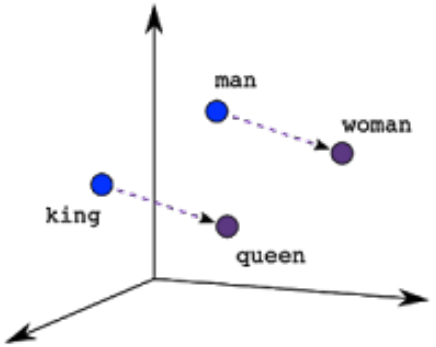
Embeddings stored with metadata for efficient access and reference.

Using embeddings to match user queries with relevant document chunks in vector databases.

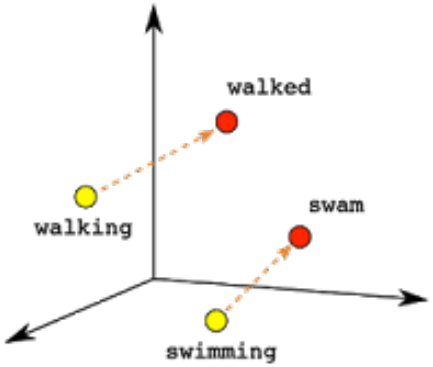
Embeddings Visualization

Input text needs to be converted into a format that the model can understand and process. That is where embedding comes in: it **transforms the text into a numerical representation** that the model can work with.

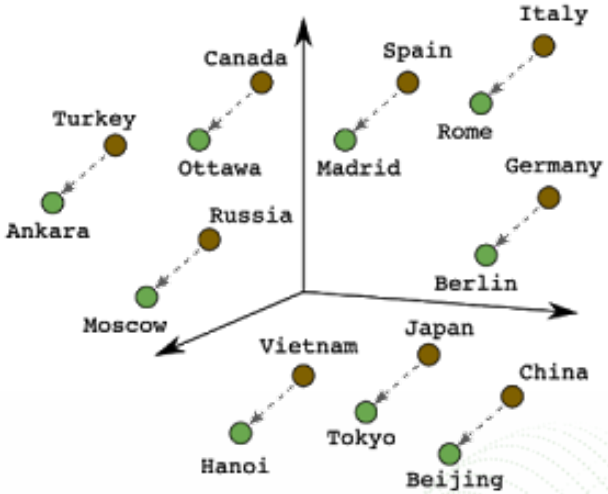
The numerical representation is in a large vector space – typically at least 768 dimensions, with the latest models using 3,072 and more. Let's imagine just three simplistic dimensions:



Male-Female



Verb Tense



Country-Capital

Note: These examples show single words for clarity. Actual embeddings for RAG represent the entire chunk of text

Graphics source: https://cloud.google.com/blog/topics/developers-practitioners/meet-ais-multitool-vector-embeddings

Utilization in RAG

Embedding user queries to retrieve relevant data chunks.

Combining retrieved data with queries to generate contextually relevant LLM responses.

Advanced Techniques in Embeddings

- Enhancing model adaptability to new queries with limited examples
- Embedding training sets, leveraging question similarities for model enhancement
- Utilizing tools like **Qdrant** for embedding and fine-tuning processes.

Introduction to Vector Databases

Vector databases use vector representations in a multi-dimensional space for data storage and processing

Efficient handling of large, complex datasets including unstructured data like text, graphics, and audio

Essential for managing the increasing complexity of modern data, particularly in AI applications

Data Representation and Indexing

Translates data into numerical vectors, each dimension representing a unique data feature

Use of spatial indexes and nearest neighbor search algorithms for rapid identification of similar vectors

Image search engines utilizing vector embeddings for similarity searches

Enhanced Contextual Relevance

Stores and retrieves contextually related data, improving the relevance of generated content in RAG

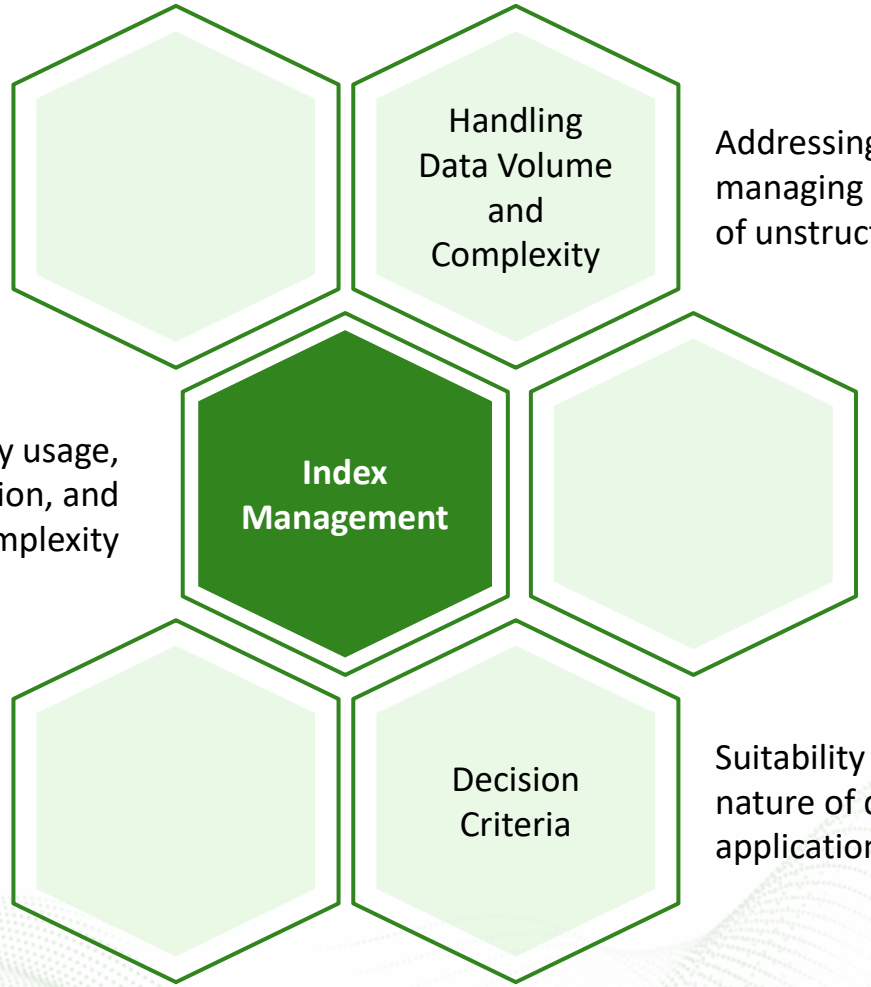
Efficient Information Retrieval

Optimized for quick access to data, crucial for applications requiring timely responses

Accurate Semantic Understanding

Leverages NLP for semantic search, enhancing the quality of interactions in RAG applications

Challenges and Future Directions

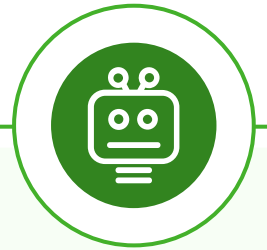


Addressing challenges in managing growing amounts of unstructured data

Balancing memory usage, index construction, and update complexity

Suitability based on the nature of data and application requirements

Introduction to Similarity Search



Augmenting LLMs with an information retrieval system for enhanced grounding data.

Uses vector embeddings for nuanced, context-aware matching.

More effective for ambiguous or interpretive content.

Retrieval Process and Techniques in RAG

Document Retrieval Process

Query conversion, embedding, and similarity matching

Threshold Setting

Customizing the number of returned documents

Key Technique – k-NN Search

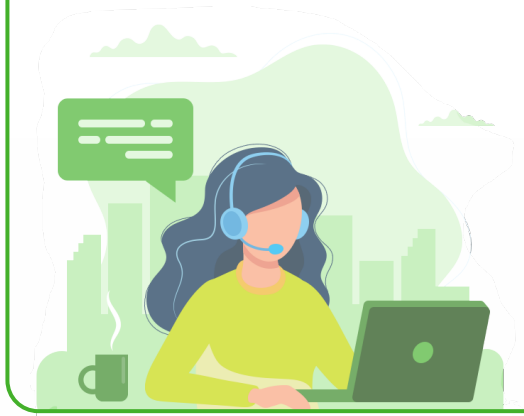
Utilized for efficient similarity matching in large datasets

Understanding Prompts and Context Sizing in RAG

Prompts

The initial input or query that initiates the retrieval and generation process.

Context includes the retrieved documents or data that provide background information, enhancing the relevancy and accuracy of generated responses.



Context Sizing

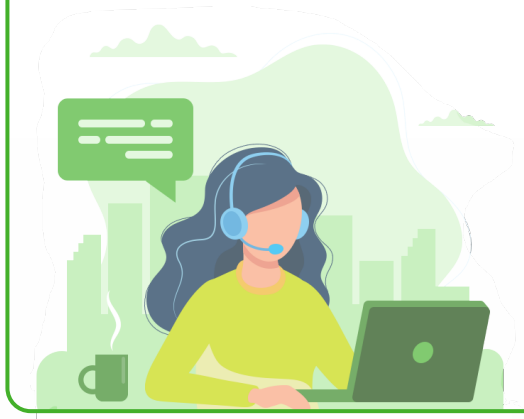
- The process of determining the optimal amount of contextual information to be retrieved and used alongside the prompt.
- Balances between thoroughness of information and efficiency of processing.

Essential for enabling RAG to adaptively retrieve and generate information for complex, knowledge-intensive tasks.

Components of Prompt and Context Sizing

Optimizing Prompt Length

Balancing document retrieval number and context length for effective retrieval



Data Preparation and Chunking

Translating and splitting text into chunks, with smaller chunks for granular matches

Advanced Chunking Techniques

Utilizing methods like **LangChain's** *ParentDocumentRetriever* and **LlamaIndex's** *SentenceWindowNodeParser.*

Embedding Strategies and Performance Trade-offs

Embedding Strategies for Context Representation

Using OpenAI embeddings for general tasks and domain-specific embeddings for specialized fields

Addressing Complex Queries

Limitations of embeddings in capturing complex or ambiguous queries

Performance Trade-offs

Balancing between response speed and accuracy, tailoring embedding strategies for specific application needs

Section 2- Vector Databases

Learning objectives

By the end of session, you should be able to

01 Understand Vector Databases, Libraries, Vector Indexes, and Type of vector indexes

02 Describe the Types of Vector Libraries – including the postgres vector database

03 Explain the Types of Vector Databases, cloud and local options, use cases
Small POC- local option (chroma DB), scaling (hosted services)

04 Optimization of Vector Database, impact of embedding dimensions (tips and tricks)

Need of Vector Databases

Unstructured Data is 80% of data or more

Vector Databases are really good with unstructured data

Examples of Unstructured Data include text, images, videos, audio, etc.

Retrieval Augmented Generation (RAG)
Expand LLMs' knowledge by incorporating external data sources into LLMs and your AI applications.

Recommender System
Match user behavior or content features with other similar ones to make effective recommendations.

Text/ Semantic Search
Search for semantically similar texts across vast amounts of natural language documents.

Image Similarity Search
Identify and search for visually similar images or objects from a vast collection of image libraries.

Video Similarity Search
Search for similar videos, scenes, or objects from extensive collections of video libraries.

Audio Similarity Search
Find similar audios in large datasets for tasks like genre classification or speech recognition

Molecular Similarity Search
Search for similar substructures, superstructures, and other structures for a specific molecule.

Anomaly Detection
Detect data points, events, and observations that deviate significantly from the usual pattern

Multimodal Similarity Search
Search over multiple types of data simultaneously, e.g. text and images

Vector Databases

Vector databases are specialized databases designed for handling vector embeddings, which are used in various applications such as machine learning models, particularly in similarity search and recommendation systems. These databases store, manage, and facilitate efficient querying of high-dimensional vectors.

Vector databases can be categorized based on their deployment environments and architectures:

Local Vector Database

- **Example: Chroma DB**
- Use Case: Suitable for small-scale proof of concept (POC) projects or when data privacy and control are paramount. They run on local servers or personal computers.

Cloud-based Vector Database

- **Example: AWS OpenSearch (formerly Elasticsearch), Google Cloud's Vertex AI Matching Engine**
- Use Case: Ideal for scaling operations as they provide robust infrastructure, managed services, and scalability options

Vector Indexes

Vector indexes are data structures designed to speed up data retrieval in vector databases. They are crucial for efficient similarity search operations.

Flat Indexes

- Description: A simple yet exhaustive search approach where each query vector is compared against all stored vectors.
- Use Case: Suitable for small datasets where brute-force search can be performed quickly.

Hierarchical Navigable Small World (HNSW) Indexes

- Description: An advanced index that uses a graph-based approach to quickly traverse closer points in the vector space.
- Use Case: Effective for large-scale and high-dimensional data, offering a good balance between accuracy and speed.

Optimization of Vector Databases

Choosing the Right Embedding Dimensions

Higher dimensions can capture more information but increase computational complexity and storage requirements. It's crucial to find a balance based on the specific use case.

Indexing Strategies

Selecting the appropriate indexing method (e.g., flat, HNSW) based on the dataset size and query speed requirements.

Batching and Parallel Processing

Processing queries in batches and using parallel computing resources can significantly speed up operations.

Hardware Acceleration

Utilizing GPUs or TPUs for operations involving high-dimensional vector calculations can drastically reduce latency.

Section 3 – RAG in the Real World

RAG in the Real World

Be prepared to have a lot of moving pieces in your solution. Be prepared for the tweaking and tuning needed over the life of the solution.

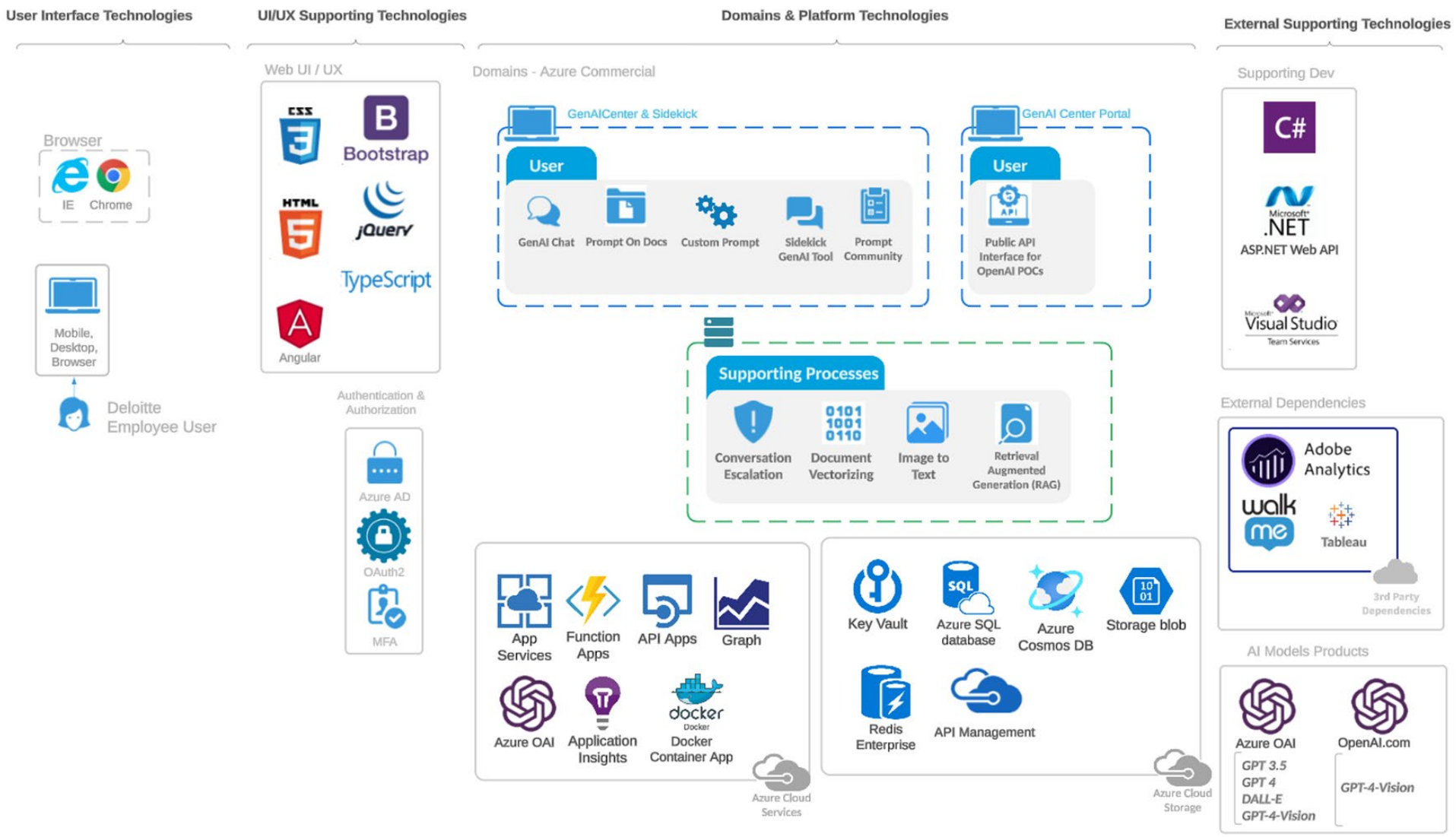
Simple RAG Use Case

- A department wants to put all of its collected knowledge in a RAG solution. This will consist of PDFs, Word docs, Excel sheets, PowerPoint decks, and more. Some of this will be usable, some won't. It all needs to be curated.

An Advanced RAG Use Case

- A client organization wants to put all of its collected knowledge in a RAG solution. This will have all the issues inherent in the Simple case, just at scale.
- Complexity
- Security
- Cost
- Multidisciplinary
- Sys Admins
- Cloud Services

Sidekick Architecture



Section 4 – LAB

Lab: Create a small RAG system

Jupyter notebook

Knowledge base

Q & A

Session Takeaways

Share key Takeaways

01 LLMs can be augmented with custom data using RAG

02 RAG comprises chunking, vectorization, embeddings, storage, retrieval, and generation

03 Vector databases, and the libraries that access them, are integral to a successful RAG implementation

04 Local LLM / RAG can be a great way to create a POC, using approved technologies

05 Production LLM/RAG applications should be in the cloud again using approved technologies

Appendix - References

References

Lewis, P., et al. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks."

Devlin, J., et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding."

Radford, A., et al. "Language Models are Unsupervised Multitask Learners."

- https://www.promptingguide.ai/techniques/rag
- https://www.pinecone.io/learn/retrieval-augmented-generation/
- https://deci.ai/blog/retrieval-augmented-generation-using-langchain/
- https://www.youtube.com/watch?v=Q-uEhJMu3ak
- https://nanonets.com/blog/retrieval-augmented-generation/
- https://microsoft.github.io/autogen/blog/2023/10/18/RetrieveChat/
- https://www.anyscale.com/blog/a-comprehensive-guide-for-building-rag-based-llm-applications-part-1

Deloitte.