Harness Proprietary Data with Foundational Models and RAG

by Marian Veteanu

Use Cases

Chat system that can answer questions from your internal knowledge base

Legal document search and analysis

Idea Generation using patent and research data to inspire new product ideas.

More Use cases: Part I

Sales and Marketing Insights

- **Personalized Content**: Generate tailored emails or product descriptions using customer data.
- **Market Research**: Summarize competitor data and market trends to support decision-making.

Employee Onboarding and Training

- **Quick Knowledge Retrieval**: Enable new hires to ask questions and get relevant answers from company resources.
- **Customized Training**: Generate personalized training programs based on job roles.

IT and Technical Support Automation

- **Faster Issue Resolution**: Retrieve past incident logs and guides for resolving tech issues.
- **Developer Support**: Provide real-time code snippets and best practices from internal repositories.

Supply Chain Optimization

- **Inventory Forecasting**: Generate forecasts based on real-time and historical data.
- **Vendor Risk Management**: Pull performance data from various sources to assess vendor risks.

Human Resources (HR)

- **Resume Screening**: Automatically retrieve and match candidate profiles with job descriptions.
- **Sentiment Analysis**: Summarize employee feedback for actionable insights.

Healthcare and Life Sciences

- **Clinical Trial Research**: Retrieve relevant studies and trials to accelerate research.
- **Diagnosis Support**: Provide doctors with real-time access to the latest medical research.

More Use cases: Part II

Contract and Legal Document Analysis

- **Contract Summaries**: Automatically extract key clauses and risks from contracts.
- **Legal Research**: Retrieve relevant case law and precedents for faster legal analysis.

Customer Data Insights

- **Customer 360 View**: Integrate and retrieve customer data from multiple sources for a comprehensive view.
- **Feedback Analysis**: Summarize feedback from various channels for insights into product improvement.

Financial Services and Risk Analysis

- **Fraud Detection**: Retrieve transaction patterns and risk data to identify fraud cases.
- **Financial Analysis**: Generate real-time financial reports and investment recommendations.

Product Development and Innovation

- **Idea Generation**: Pull data from patents and research to inspire new product ideas.
- **R&D Collaboration**: Retrieve internal and external research to support innovation efforts.

Knowledge Management

- **Unified Search**: Retrieve context-aware information across fragmented enterprise systems.
- **Document Summaries**: Generate concise summaries of long reports or documents.

Retail and E-Commerce

- **Product Recommendations**: Suggest personalized products based on browsing and sales data.
- **Dynamic Pricing**: Retrieve competitor and market data to optimize pricing strategies.

About Foundational Models

What Are Foundational Models?

- Large-scale models trained on vast amounts of diverse data.
- Capable of performing multiple tasks without task-specific training.
- Examples: GPT (OpenAI), Gemini (Google), Llama, and more.

Key Characteristics

- **Pretrained**: Built on extensive datasets, allowing generalization across different domains.
- **Multimodal Capabilities**: Some models handle text, images, and other forms of input.
- **Transfer Learning**: Fine-tuning these models for specific use cases with minimal data.

Benefits for Enterprises

- **Scalability**: Handle large, complex datasets efficiently.
- **Versatility**: Can be applied across various domains, from customer support to research.
- **Cost-Effective**: Leverage pre-existing knowledge without needing to train from scratch.

Large Language Models (LLMs) are a type of foundational model specifically designed to understand and generate human language.

Your First Encounter with an LLM

Many people’s first experience with a Large Language Model (LLM) is through chatbots or virtual assistants like ChatGPT, Google Assistant, Siri, or Alexa.

These assistants (powered by LLMs) are trained on a vast and diverse range of text data, sourced primarily from publicly available information such as books, websites, articles, forums, and other written content.

The models don’t have access to proprietary databases or private information unless explicitly provided during a conversation.

It is important to note that ChatGPT's training data has a cutoff point (September 2021 for older models), meaning it doesn’t have access to real-time information or events that occurred after this period.

Why Base LLMs Are Not Enough?

Outdated Knowledge

Problem: LLMs are trained on data up to a specific point in time, meaning they do not have access to the latest information post-training.
Cause: Once an LLM is trained, it retains a fixed snapshot of the world's knowledge. Updates require full retraining or fine-tuning, which is resource-intensive.
Impact: This can be a critical limitation in fast-moving industries or environments requiring real-time information (e.g., financial services, legal, R&D).

Retraining Challenges

Problem: Retraining or fine-tuning LLMs to incorporate new knowledge or adapt them to specific business needs is expensive and time-consuming.
Cause: Large-scale training of LLMs involves vast computational resources, and fine-tuning can require additional datasets and infrastructure.
Impact: For many businesses, retraining LLMs regularly is not practical or cost-effective, limiting their adaptability to specific or proprietary business needs.

Hallucinations

Problem: Base LLMs often generate false or misleading information when they don't have the answer.
Cause: LLMs are probabilistic and rely on patterns from their training data. When uncertain, they may "hallucinate" plausible-sounding but incorrect responses.
Impact: Can lead to unreliable insights, especially in critical business contexts.

Limited Context Windows

Problem: LLMs have a limited "memory" or context window, meaning they can only process a certain amount of text at once.
Cause: Models like GPT-4 can handle about 8,000 tokens (~6,000 words) in a single query, but enterprise data often spans far larger documents, making it hard to retrieve all necessary context in one go.
Impact: This limits the ability to effectively process large documents (e.g., legal contracts, scientific papers) or complex queries requiring multiple sources of data.

Introducing RAG (Retrieval-Augmented Generation)

What is RAG?

Retrieval-Augmented Generation (RAG) combines the power of Large Language Models (LLMs) with real-time information retrieval from external sources.

LLMs generate text-based outputs, while RAG enhances them by retrieving relevant data from external knowledge bases, vector databases, or proprietary datasets.

Why RAG Matters for Enterprises?

Reduces Hallucinations: By incorporating real-time, factual information from business knowledge bases, RAG helps mitigate the risk of LLMs generating false or incorrect responses.

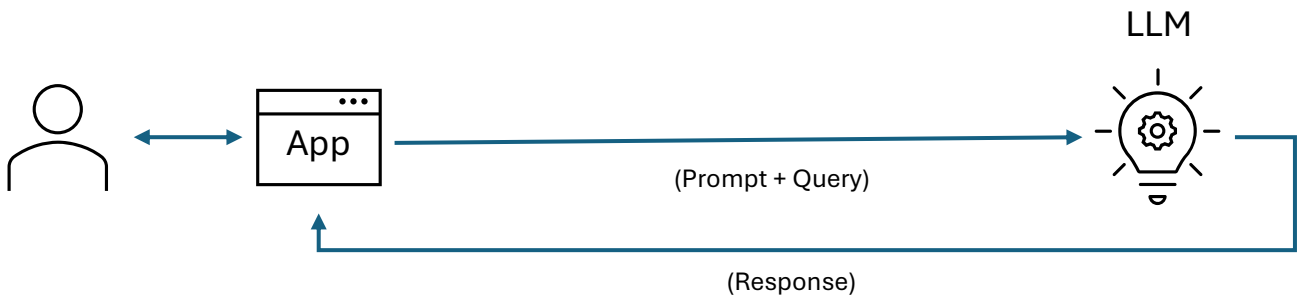
Leverages Proprietary Data: RAG enables LLMs to access and use up-to-date, proprietary business data that wasn't available during the model's initial training.

Real-Time Knowledge Access: RAG allows models to generate responses based on specific enterprise datasets, offering better answers tailored to business needs.

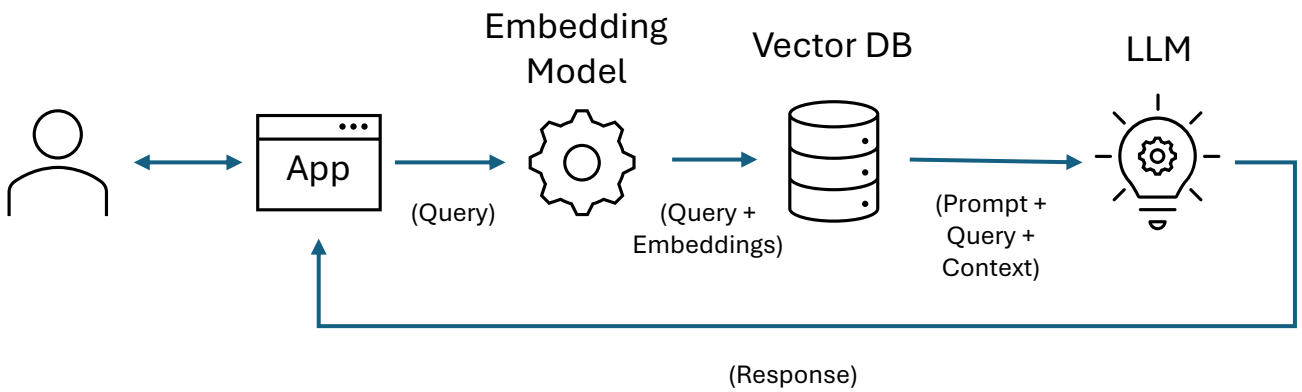
How RAG works?

1. **Question or Query:** User asks a question.
2. **Information Retrieval:** The system retrieves relevant documents or data from a database.
3. **LLM Generation:** The LLM uses this context to generate a more accurate and relevant response.

Without RAG



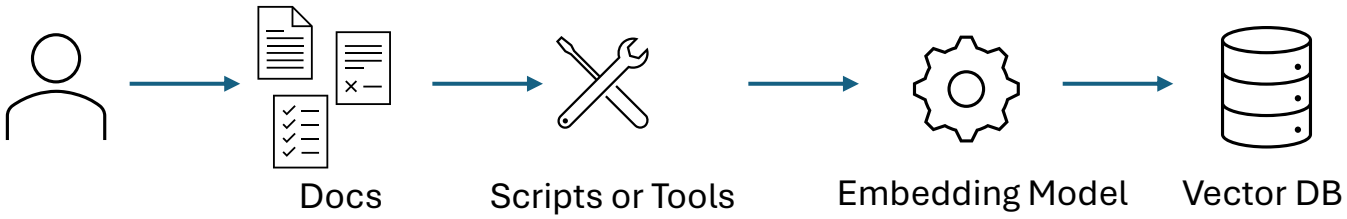
With RAG



Document ingestion

A vector database allows RAG systems to perform fast, accurate searches by comparing semantic embeddings rather than traditional keyword matching.

If the knowledge base of your business stays in a collection of PDF, Word documents, PowerPoint files, etc., then you can build the DB using a document ingestion phase.



Step 1: Select your documents: Find all relevant documents in various formats (PDF, Word, etc.).

Step 2: Data Cleansing and Preprocessing: Convert documents into a clean, machine-readable format, remove unnecessary data

Step 3: Generate embeddings: Generate vector representations of the document's content.

Text Snippet: "Who is Marian Veteanu?"

Embedding: `[0.023, -0.569, 0.245, 0.482, -0.091, 0.361, 0.128, -0.553, 0.734, -0.142, ...]`

Step 4: Store embeddings in a Vector DB: Fast, scalable information retrieval.

About embeddings

What are Embeddings?

- Embeddings are dense, numerical representations of data (like text, images, or documents) in a high-dimensional vector space.
- They capture the semantic meaning of the data, enabling the system to find similarities or relevant information beyond keyword matching.

How to Create Embeddings?

- **Choose an Embedding Model:** Select a model that generates embeddings (e.g., OpenAI's text-embedding-ada-002, Sentence-Transformers, or BERT).
- **Generate Embeddings:** Use the model to convert the text into vectors. Each word, sentence, or document gets transformed into a vector representation.

Example:

```
import openai

openai.api_key = 'your-api-key-here'

# The text you want to generate embeddings for
text = "Who is Marian Veteanu?"

# Call the OpenAI Embedding API
response = openai.Embedding.create(
    input=text,
    model="text-embedding-ada-002"  # OpenAI's embedding model
)

embedding = response['data'][0]['embedding']
```

```
[0.0123, -0.0345, 0.0789, 0.0121, -0.0567, ..., 0.0456]
```

(A list of numbers. 1536 dimensions for OpenAI's text-embedding-ada-002 model)

Vector databases

What are Vector Databases?

- Vector databases are specialized databases designed to store, manage, and search high-dimensional vectors (such as embeddings).
- These vectors represent the semantic meaning of data, allowing the system to retrieve relevant information based on similarity rather than exact matches.
- Vector databases can update dynamically, incorporating new embeddings as data is ingested.

Why Not Use a Regular Database?

- Regular databases (e.g., relational or NoSQL databases) are optimized for exact matches or basic keyword-based search.
- They are not designed to handle the complexity of semantic search, which relies on understanding the meaning of data rather than just matching keywords.
- Vector databases integrate deeply with machine learning pipelines and tools, enabling seamless storage and retrieval of embeddings. Regular databases would require custom solutions or inefficient workarounds to achieve the same functionality.

Use Case Example

A vector database stores embeddings for thousands of customer support tickets. When a new query comes in, the system can quickly retrieve the most relevant tickets based on their semantic similarity to the query.

Vector DB providers

Cloud-Based Vector Databases

Pinecone
https://www.pinecone.io
A fully managed vector database designed for fast and scalable vector search. Pinecone provides filtering, real-time indexing, and is widely used in AI and ML-powered applications.

Azure Cognitive Search with Vector Search
https://azure.microsoft.com/services/search/
Azure Cognitive Search it is fully managed and scales automatically.

Amazon Kendra
URL: https://aws.amazon.com/kendra/
Amazon Kendra is a highly accurate and easy-to-use search service supporting vector search.

Google Vertex AI Matching Engine
https://cloud.google.com/vertex-ai
Google's Vertex AI offers a matching engine that enables high-performance vector similarity search.

Open-Source Vector Databases

Weaviate
https://weaviate.io

Milvus
https://milvus.io

Qdrant
https://qdrant.tech

Vespa
https://vespa.ai

Vald
https://vald.vdaas.org

Faiss (Facebook AI Similarity Search)
https://github.com/facebookresearch/faiss

Prompting using context information

To enhance the accuracy and relevance of the Large Language Model's (LLM) responses, integrating retrieved context (e.g., from vector databases) into the prompt ensures the model has up-to-date, domain-specific knowledge.

How to Integrate Retrieved Information?

Step 1. Retrieve Relevant Data: Use RAG to pull specific, contextually relevant information from your vector database (e.g., proprietary documents or recent data).

Step 2. Embed the Context in the Prompt: Include the retrieved text or data in your prompt to give the LLM a knowledge base to work from.

Step 3. Send augmented prompt to LLM

Example

Prompt without Integration

"What are the latest trends in cloud computing?" ← LLM may return generic, outdated information.

Prompt with Integrated Retrieval

"Based on the following information [*retrieved context*], what are the key trends in cloud computing that we should focus on?" ← LLM uses the specific retrieved context to provide a relevant answer.

Summarize retrieved information if it's too lengthy to fit within the model's context window.

Code Sample: Part 1

A simple example on how to use RAG with OpenAI and Pinecode

- Install required packages

```
pip install openai pinecone-client
```

- Get the API keys from OpenAI and Pinecone (requires sign up)
- Set Up Pinecone Vector Database
- Create embeddings using OpenAI

```
# Sample documents you want to store as embeddings
documents = [
    "OpenAI is an AI research and deployment company.",
    "Pinecone provides a vector DB.",
    "RAG improves LLM accuracy by using external data."
]

# Generate embeddings using OpenAI
embeddings = []
for doc in documents:
    response = openai.Embedding.create(input=doc,
              model="text-embedding-ada-002")
    embedding = response['data'][0]['embedding']
    embeddings.append(embedding)

# Store embeddings in Pinecone
# Use a simple id like 'doc0', 'doc1'
for i, emb in enumerate(embeddings):
    index.upsert(vectors=[(f"doc{i}", emb)])
```

Code Sample: Part 2

A simple example on how to use RAG with OpenAI and Pinecode

- Use OpenAI to Generate a Response

```
query = "What is Pinecone used for?"

query_embedding = openai.Embedding.create(
    input=query,
    model="text-embedding-ada-002"
)['data'][0]['embedding']

# Query Pinecone for similar vectors
result = index.query(queries=[query_embedding], top_k=2)

# Retrieve relevant documents using the match IDs
documents = [
    "OpenAI is an AI research and deployment company.",
    "Pinecone provides a vector DB.",
    "RAG improves LLM accuracy by using external data."
]
retrieved_docs = [documents[int(match['id'][-1])] for match
                  in result['matches']]

# Construct the prompt for OpenAI using retrieved context
context = "\n".join(retrieved_docs)
final_prompt = f"Context:\n{context}\n\nQuestion:
{query}\nAnswer:"

# Generate a response from OpenAI
response = openai.Completion.create(
    engine="text-davinci-003",
    prompt=final_prompt,
    max_tokens=100
)

# Print the generated answer
print(response['choices'][0]['text'].strip())
```

Langchain and other frameworks

Before coding your first RAG application, you may want to checkout the different available frameworks such as Langchain, Transformers (by Hugging Face), LlamaIndex, Haystack, etc.

These frameworks may help you simplify complex workflows.

About Langchain

Langchain is a popular framework designed to build and deploy applications that use Large Language Models (LLMs) efficiently.

It simplifies the integration of LLMs with external data sources, such as vector databases (like Pinecone), APIs, or other tools, to support Retrieval-Augmented Generation (RAG).

It helps with prompt engineering by allowing dynamic input data in prompts. It facilitates combining multiple LLM tasks and data retrieval steps into a single workflow (chains).

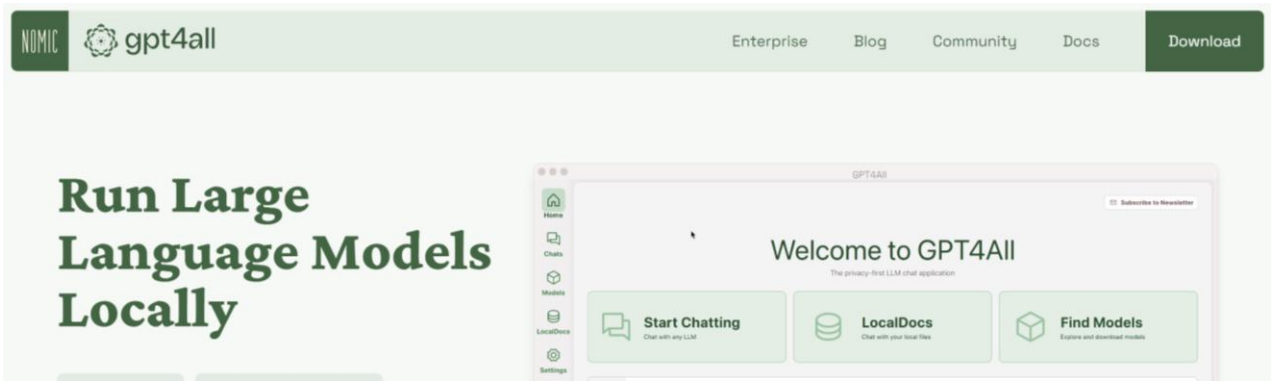
Security, Compliance, and Guardrails

- Ensure that sensitive, proprietary, and internal data used in Retrieval-Augmented Generation (RAG) workflows is securely managed and stored.
- Use access controls to limit who can interact with the vector database and LLM systems, ensuring only authorized personnel and systems have access to sensitive information.
- Adhere to regulatory frameworks such as GDPR, HIPAA, or CCPA. Ensure that data anonymization or pseudonymization techniques are applied when necessary to protect user privacy.
- Implement safeguards to regularly audit and mitigate biased outputs.
- Implement detailed logging and monitoring of all requests and responses in RAG systems to ensure that data usage is transparent and auditable in case of security or compliance reviews.
- Implement ethical guardrails to prevent misuse, ensuring that the RAG system aligns with the company's ethical standards and regulatory obligations.

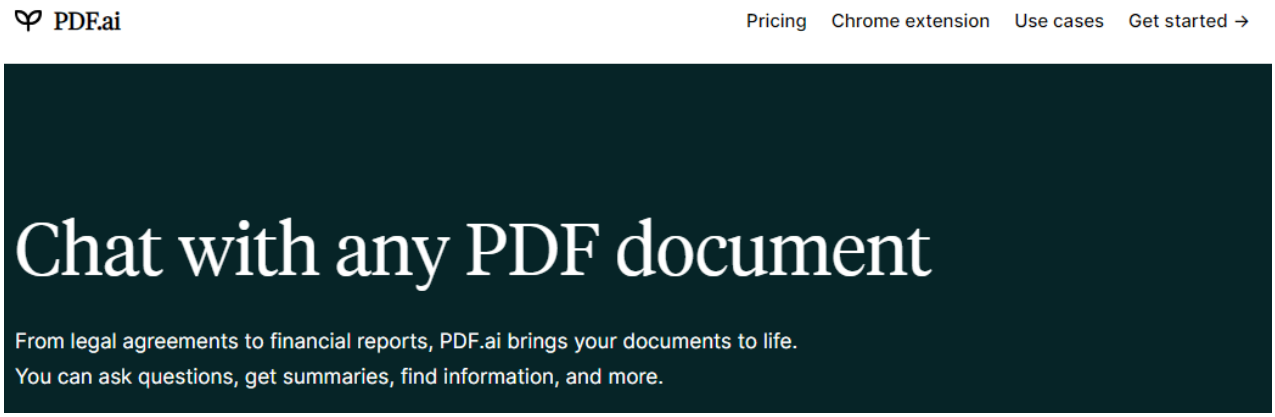
End-user tools

If you're interested in exploring the capabilities of LLM+RAG systems as an end-user, check out the following tools:

https://www.nomic.ai/gpt4all



https://pdf.ai/



Reference:
https://www.linkedin.com/pulse/what-retrieval-augmented-generation-rag-why-its-hot-topic-nawaz-qmczc/
https://www.infoworld.com/article/2336099/retrieval-augmented-generation-step-by-step.html
https://blogs.nvidia.com/blog/what-is-retrieval-augmented-generation/

Marian Veteanu

Technology Architect and Product Leader

Looking to see how I can
add value to your organization?

Message me!

https://www.linkedin.com/in/mveteanu/
https://x.com/mveteanu