Querying LLMs with Retrieved Data

1. Overview of AI Workflows	2. Why Postgres as a vector store	3. Storing and managing vectors	4. Querying the vector store
• Look at high-level architecture - LLMs, vector stores and JSON • Look at key vocabulary and concepts (embeddings, vectors, hybrid queries, etc.)	• What is a vector store? Key concepts and use cases. • Why Postgres and how does it compare with other market tools • Setting up Postgres with vector capabilities (pgvector) • **Lab**: Install and configure Postgres using Docker	• Generating embeddings: Overview of tools and workflows • Storing and organizing embeddings in Postgres • Strategies for handling large datasets including chunking • Dense and sparse vectors • **Lab:** Generate embeddings for a dataset and store them	• Techniques for similarity search: k-NN, cosine similarity • Using indexes to optimize vector queries • Reranking results • **Lab:** Query stored vectors to retrieve similar items (document/image search)
5. Querying LLMs with retrieved data	**6. NoSQL with JSON in Postgres**	**7. Integrating Vector, Relational and JSON Data**	**8. Putting it all together**
• Recap on querying LLMs vis APIs • Best practices for combining vector retrieval with LLM prompts • Prompt configuration parameters (temperature, top-k, etc) • **Lab:** Build a pipeline where vector store results enhance LLM responses (context-aware Q&A, etc)	• Overview of JSON/JSONB support in Postgres • Querying JSONB data with SQL • Indexing JSONB data for performance • **Lab:** Design a schema mixing vector, relational and JSONB data for a sample project	• Building hybrid queries to power advanced workflows • **Case study:** Combining embeddings, metadata (relational) and configurations (JSON) • **Lab:** Implement a hybrid query to support a sample AI use case	• Full stack pipeline demo: Retrieve data, query the LLM and return results • Debugging and optimising the workflow • Spotlight on LLM frameworks • **Lab:** Build a working application combining all elements

Plan for the session

- Understand the Fundamentals of LLM Querying
- Apply Best Practices for Combining Vector Retrieval with LLM Prompts
- Optimize LLM Queries with Configuration Parameters
- Lab: **Build a Pipeline for Context-Aware LLM Responses**

How do we interact with LLMs?

- Large Language Models (LLMs) generate text based on input prompts
- They rely on probabilisitic word prediction using pre-trained embeddings
- Two common ways to interact with LLMs:
  - Direct Input: User provides a prompt, LLM generates an output
  - Retrieval-Augmented Generation (RAG): LLM is provided with external data to refine responses

📌 **Key Takeaway**: LLMs do not "remember" external data unless explicitly provided in context.

Using APIs to Query LLMs

- Most production use cases rely on **LLM APIs** rather than self-hosted models.
- APIs provide:
  - **Standardized endpoints** (e.g., POST /v1/completions)
  - **Customization parameters** (temperature, max_tokens, top-k)
  - **Integration with retrieval pipelines**
- Example providers:
  - OpenAI (gpt-4, gpt-3.5-turbo)
  - Cohere, Anthropic (Claude), Google (Gemini)
  - Open-source models (via Hugging Face Inference API)

Demo

Common Challenges in LLM Querying

- **Context Length Limitations**: Most LLMs can only process a fixed number of tokens (~4K-128K).
- **Hallucinations**: LLMs may generate plausible but incorrect information.
- **Deterministic vs. Stochastic Behavior**:
  - Lower temperature = more predictable responses
  - Higher temperature = more creative but less reliable responses
- **Lack of Domain Knowledge**: LLMs may not have up-to-date or domain-specific knowledge unless retrieved externally.

📌 **Key Takeaway**: To improve reliability, external knowledge should be retrieved and provided as context.

Enhancing LLM Queries with Retrieved Data (RAG)

- **What is RAG?**
  - Retrieves external data before querying an LLM.
  - Provides additional knowledge in the prompt.
- **Steps in RAG-Based Querying**:
  - **User Query** → **Vector Search**: Find relevant stored data.
  - **Retrieve Results** → **Context Injection**: Add retrieved text to the prompt.
  - **Query LLM with Enhanced Context** → Improved response accuracy.
- **Example Use Case**:
  - Searching an internal knowledge base before asking an LLM.
  - AI-powered search assistants retrieving FAQs before generating answers.

📌 **Key Takeaway**: Combining vector retrieval with LLMs ensures **more accurate, relevant, and reliable** responses.

Demo

Optimizing Vector-Augmented LLM Queries

✅ **Retrieve Before You Generate**

- Always fetch relevant data first, then pass it into the LLM.

✅ **Limit the Number of Retrieved Items**

- Avoid overloading the LLM with too much context (3-5 items max).

✅ **Format Context in a Structured Way**

- Use **bullet points, JSON, or key-value pairs** in the LLM prompt.

✅ **Use Clear Prompt Engineering**

- Tell the LLM exactly how to use the retrieved data (e.g., *"Summarize the books and categorize them by experience level"*).

📌 **Key Takeaway:**
The quality of LLM responses depends on how well the vector-retrieved data is structured and provided.

Problem	**Cause**	**Fix**
LLM Hallucinates Facts	LLM generates responses without sufficient grounding in real data.	**Ensure retrieved data is structured** before sending it to the LLM. Clearly separate facts from instructions.
Retrieved Results Are Not Relevant	Poor-quality embeddings or incorrect similarity metric used.	Use a **better embedding model** (bge-m3), **filter by metadata**, and **fine-tune retrieval parameters (top-k, similarity threshold)**.
LLM Response Is Too Generic	The LLM is responding broadly instead of using retrieved context.	Provide **explicit instructions in the prompt**, e.g., *"Base your answer only on the books provided."*
Query is Slow	**Query is Slow**	**Query is Slow**

Tuning LLM Responses with Parameters

📌 **Key Points:**

- LLM outputs can be controlled by adjusting key parameters.
- The same query can produce **deterministic, creative, or factually rich responses** depending on settings.
- Understanding parameter effects allows **fine-tuning for optimal results**.
- 💡 **Example Use Case:**
- Setting temperature=0.0 ensures **precise, factual responses** (e.g., legal documents).
- Setting temperature=0.9 enables **creative storytelling** (e.g., marketing copy).

Parameter	Description	Effect on Output
temperature	Controls randomness (0.0 = deterministic, 1.0 = highly creative).	Lower values ensure predictable responses, higher values increase variability.
top_p	Nucleus sampling – sets diversity by filtering top probability tokens.	Lower values (e.g., 0.5) create **focused answers**, higher values (e.g., 0.9) allow **more variety**.
max_tokens	Limits response length.	Prevents excessively long outputs, useful for API efficiency.
frequency_penalty	Reduces repetition of words/phrases.	Higher values discourage repeated content in responses.
presence_penalty	Encourages introducing new topics.	Higher values make the response more exploratory and broad.

📌 **Key Takeaway:**
Different parameters adjust precision, randomness, verbosity, and repetition.

Parameter Settings for Different Tasks

Use Case	Recommended Parameters
Fact-Based Q&A	temperature=0.0, top_p=0.5, max_tokens=150
Creative Writing	temperature=0.9, top_p=1.0, max_tokens=500
Summarization	temperature=0.3, top_p=0.8, max_tokens=250
Conversational AI	temperature=0.7, top_p=0.9, max_tokens=150, presence_penalty=0.6
Structured Data Extraction	temperature=0.1, top_p=0.5, max_tokens=100

📌 **Key Takeaway:**
Tuning parameters aligns the model output with specific needs (accuracy, creativity, conciseness, etc.).

Effects of Changing Temperature & Top-P

🔍 **Scenario:** *Asking OpenAI: "Tell me an interesting fact about space."*

Parameter Setting	Expected Response
temperature=0.0	*"The speed of light in a vacuum is approximately 299,792 kilometers per second."* (Strict factual answer)
temperature=0.9	*"Did you know that a day on Venus is longer than a year on Venus?"* (More unusual/interesting fact)
top_p=0.5	*"There are over 100 billion galaxies in the observable universe."* (More direct fact)
top_p=1.0	*"Some scientists believe there might be unknown forms of life thriving in the extreme conditions of exoplanets!"* (Diverse and speculative fact)

📌 **Key Takeaway:**
Higher temperature and top_p values increase creativity, while lower values improve precision.

Fine-Tuning LLM Outputs Effectively

✅ **Start with Defaults & Adjust Gradually**

- OpenAI defaults work well for most cases; tweak based on observed behavior.

✅ **Use Low temperature for Factual Tasks**

- Set temperature=0.0 for reliability (e.g., financial reports, legal summaries).

✅ **Limit Response Length (max_tokens)**

- Helps manage API costs and avoids excessively verbose responses.

✅ **Use frequency_penalty to Reduce Repetition**

- Good for avoiding redundant answers in long outputs.

✅ **Balance temperature and top_p**

- Avoid setting **both high**—use one or the other to fine-tune randomness.

Demo

Lab


Questions
?