UCLA Advanced Research Computing

Learn Prompt Engineering and Retrieval Augmented Generation Using Open-Source LLMs

Presented By

Research Data and Web Platforms Group

Contact Us: webteam@oarc.ucla.edu

Andrew Browning - Research Data and Web Platforms Manager

Anthony Doolan - Software Developer

Hayk Zakaryan - Software Architect

Agenda

1. Introduction to Prompt Engineering and Retrieval Augmented Generation (RAG)

2. Prompt Engineering Basics

3. Prompt Engineering Within a RAG Application

4. RAG Application Demonstration

5. Conclusion/QA

Prompt Engineering Basics

Introduction to Prompt Engineering and RAG

With Prompt Engineering we craft instructions (prompts) to guide generative AI models, ultimately controlling outputs.

With RAG we use an external knowledge base in concert with Prompt Engineering to further restrict outputs

Ollama Setup and Installation

Windows

Ollama download page:
https://ollama.com/download

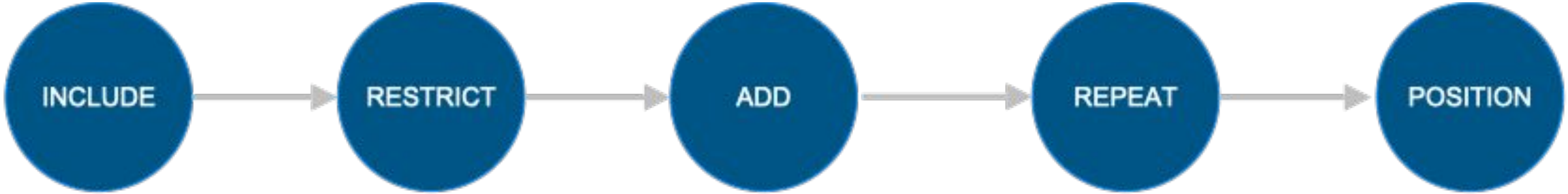
MAC (
M series chips)

```
    brew install ollama
    ollama serve - start
    ollama
    ollama pull mistral -
    download mistral
    ollama list - list
    available models
    pip install ollama -
    Install Ollama Python
    package
```

Linux

```
     curl -fsSL
https://ollama.com/install.sh
| sh
ollama serve - start ollama
ollama pull mistral - download
mistral
ollama list - list available
models
 pip install ollama - Install
Ollama Python package
```

Prompt Engineering



INCLUDE → RESTRICT → ADD → REPEAT → POSITION

INCLUDE: Include instructions that the model not make anything up and only stick with the facts

RESTRICT: Restrict the output to a confined list

ADD: Add chain of thought style instructions, e.g., "solve the problem step by step."

REPEAT: Repeat most important instructions in the prompt multiple times

POSITION: Position most important instructions last

IRARP

Prompt Engineering (Cont.)

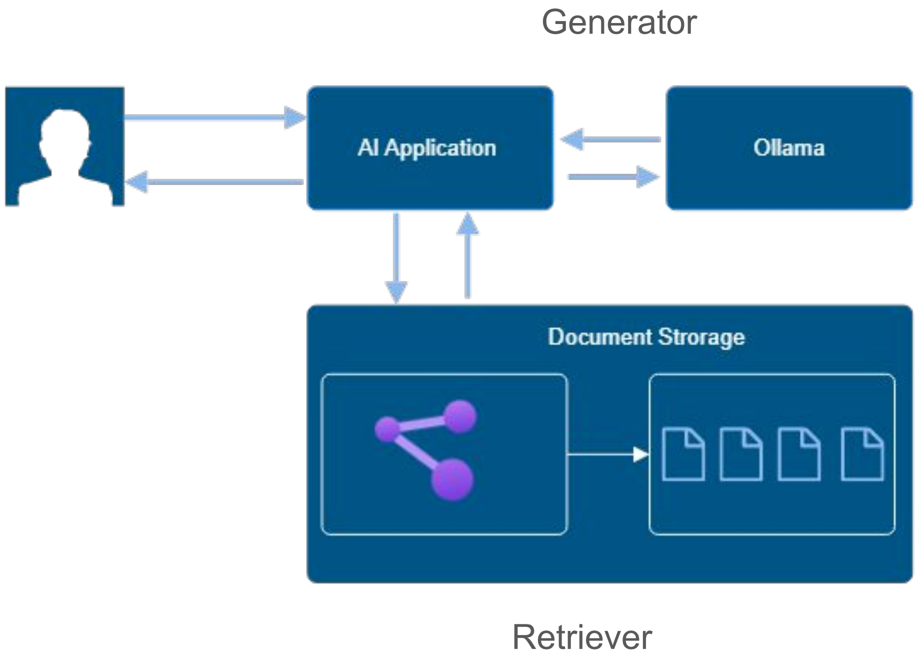
- **INCLUDE:** You are a university researcher who referees machine learning and data science research papers. You summarize papers with precision, referencing metrics, datasets, and experimental methods.
- **RESTRICT:** Output must include:
  - A 3-paragraph technical summary
  - A 1-paragraph critique highlighting novelty and flaws
  - Suggested applications for the research
- **ADD:** Use domain-specific language that assumes the reader is familiar with deep learning, statistical modeling, and ML evaluation protocols (e.g., AUC, BLEU, perplexity).
- **REPEAT:**Always mention key components like model architecture, dataset size, benchmarks, and limitations.
- **POSITION:** Your tone is both professional and insightful, similar to NeurIPS, ICLR, or Elsevier peer review. If possible, cite related work or standards in the field.

RAG With Prompt Engineering

Leveraging Prompt Engineering Through RAG

- The better the knowledge base the better the RAG application.
- RAG empowers developers to generate prompts for users lacking prompt-engineering expertise
- RAG has two parts
  - Retriever - grabs documents/data from the knowledge base and passes that data to the LLM.
  - Generator - Uses the prompt and the LLM to generate responses based upon result from the retriever.
- RAG use cases - Chatbots, QA systems, Personalized Medical Records, and Real-Time Content Generation.

RAG Application Framework



- User submits query to AI application
- AI application queries document storage and fetches result
- AI application submits original query and document result to Ollama for inference
- AI application returns result to user

RAG Application Framework (Cont)

Sample Prompt Sent to LLM

- **User question** - Does adding people to a complex project make it go faster?
- **Knowledge base context** - No, because of increased communication vectors, it may go slower.
- **Rules**
  - Prompt Engineering Steps - Include, restrict, add, repeat, position
  - Return answer based upon user’s input

Answer:
Usually no; it often slows things down first.

RAG Application Framework (Cont)

- Use creativity, logic, and experimentation when writing prompts (**IRARP**)
  - Generate multiple prompt variants, form hypotheses about why each should work, run small tests, analyze outcomes, and iteratively refine using your IRARP loop until you converge on reliable behavior.
- Understand the capabilities and limitations of the chosen LLM
  - Design prompts around the model’s context window, tools, latency and rate limits, and knowledge cutoff — constraining scope and delegating tasks (e.g., math, retrieval) to tools the model can reliably use.
- Control hallucinations with clear prompting (**IRARP**)
  - State sourcing rules (“answer only from provided context; cite sources; say ‘I don’t know’ if missing”), evaluate failures, and tighten instructions over IRARP cycles to reduce unsupported claims.
- Decrease vulnerabilities such as prompt injection using (**IRARP**)

Sample Python Application

```
# Split text into smaller chunks
text_splitter = CharacterTextSplitter(chunk_size=3000, chunk_overlap=50)
doc_chunks = text_splitter.create_documents([docs_text])

# Create local embeddings & store in a local vector db. Knowledge is lost on quit
embed_model = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")
vectorstore = FAISS.from_documents(doc_chunks, embedding=embed_model)

# Create a retrieval-based QA chain using Ollama as the LLM
retriever = vectorstore.as_retriever(search_type="similarity", search_kwargs={"k": 2})

CUSTOM_PROMPT = PromptTemplate(
      template="""You are an expert at answering questions about the Mythical Man Month…
      Answer the following question using only the given context: {context}.
      Question: {question}
      Answer:""",
      input_variables=["context", "question"],
)
```

Sample Python Application

```python
# Create the chain
ollama_llm = OllamaLLM(
      endpoint_url="http://127.0.0.1:11434", model="llama3.2", temperature=0.7 )

qa_chain = RetrievalQA.from_chain_type(
      llm=ollama_llm, chain_type="stuff", retriever=retriever, chain_type_kwargs={"prompt": CUSTOM_PROMPT})

# Pydantic model for incoming requests
class QueryRequest(BaseModel):
      question: str

@app.post("/query")
def query_llm(request: QueryRequest = Body(...)):
      user_question = request.question
      result = qa_chain.invoke({"query": user_question})
      return result['result']
```

Sample Python Application (Cont)

Langchain compatible Open-source vector databases for RAG applications

- Facebook AI Similarity Search (FAISS)
  - Very easy to setup and experiment with
  - In memory storage
  - Fast
  - Ephemeral Storage
- Chroma DB
  - Easy to set up *persistent* Storage
  - Slower than FAISS for large scale queries

Using Chunking and Embedding Models

- Ollama and Hugging Face both have embedding models that can be leveraged by langchain.
- Chunking tweaks the context window and helps to keep the context short and concise., chunking too much can have negative effects.
- Different document types and use cases require different chunking strategies.
- Example Chunking Strategies (basic):
  - Plain Text - Split on character count and new lines
  - CSV - Split on new rows
  - Web Page - Split on headings

Example Command Line Demo

Github Repo:

https://github.com/ucla-oarc-web/rag-for-open-source-llms

Tools Required

- Docker or Python
- Ollama

Prompting with a Web Based RAG Application

GitHub Repository: https://github.com/ucla-oarc-web/prompt-engr-rag-workshop

- Uses all Open Source Technologies
- Accepts PDF or Web URL Knowledge Base
- Programming Language - Python 3
- Large Language Model - Llama 3
- Python Framework - Langchain
- UI Framework - Streamlit

RAG Web-Based Demo

Sample Application Demo

1.) Command line Rag demo - Uses open source tools/packages only

2.) UI based Rag demo - Uses open source tools/packages only

Further Reading - Available Tools and Model Selection

Ollama - Download page and resources

Langchain - Homepage and resources

Embeddings - Resource to learn more about LLM embeddings

Python3 - Python homepage

ChromaDB - Chroma homepage and resources

FAISS - home page and resources

Questions