Project 4: Generative AI and Artistic Expression

Bibek Dhungana

December 3, 2025

1 Creative Work

[Paste the generated poem here. For example:]

The silicon mind awakens in the dark,
Seeking gradients in the manifold's arc...

2 Description of the Work

2.1 High-Level Description

The creative work presented above is a philosophical poem generated entirely by a local Large Language Model (LLM). The content of the poem explores the concept of machine understanding, using technical terminology from the CS 4260/5260 course slides as metaphors for the AI's internal struggle.

Stylistically, the work mimics a reflective, slightly melancholic tone, juxtaposing rigid computer science definitions (from the RAG context) with abstract existential questions.

2.2 Meaning and Interpretation

To me, this work represents the gap between "processing" and "understanding." The AI uses words like "optimization," "search," and "states," which it retrieved from the course slides, to describe its own existence.

I want the observer to understand that while the AI can flawlessly retrieve and organize these definitions, the "yearning" expressed in the poem is a simulation—a mathematical prediction of what yearning sounds like, rather than the feeling itself.

2.3 Generation Process

The generation process involved a technique known as **Retrieval-Augmented Generation (RAG)**. I built a custom Python pipeline using the LangChain framework to perform the following steps:

1. **Ingestion:** All lecture slides (PDFs) were loaded and split into text chunks of 1200 characters.
2. **Embedding:** These chunks were converted into vector representations using the `sentence-transformers/all-MiniLM-L6-v2` model via HuggingFace.
3. **Storage:** The vectors were stored in a local Chroma database.
4. **Retrieval & Generation:** I used the `gpt-oss:20b` model via Ollama. The system first summarized the technical slides to establish a "ground truth" of knowledge, and then used that summary as a prompt constraint to generate the poem.

Minimal iteration was required for the code structure, but I adjusted the prompt specifically to ensure the AI used the technical terms as *metaphors* rather than just listing definitions.

3 Discussion of AI Tools

3.1 Training Data Influence

The model used, `gpt-oss:20b`, was likely trained on a vast corpus of internet text, including technical documentation and literature.

3.2 Observations on the Creative Process

Using a local RAG pipeline for creative writing was distinct from using a web-based tool like ChatGPT.

4 Enhancing the Process with Course Concepts

4.1 Integration with Search and Planning

Currently, the system uses a simple similarity search (KNN) to find context. To enhance this, we could integrate **Planning algorithms**. Instead of generating the poem in one pass, a planning agent could outline the poem's stanza structure first (e.g., Stanza 1: The Problem, Stanza 2: The Search, Stanza 3: The Solution).

Additionally, the retrieval step currently relies on specific keywords. As discussed in class, a semantic search or a **Beam Search** approach during the decoding phase could explore multiple potential poetic lines to find the one that maximizes both rhyme scheme and semantic relevance to the slides.

4.2 Attribution and Nearest Neighbors

One of the major ethical issues in Generative AI is the lack of citation. My project actually solves this naively. By using RAG, I can trace exactly which "chunks" of the PDF were retrieved to generate the summary that informed the poem.

As mentioned in the prompt, a **Nearest Neighbor search** against the training instances is exactly what my code performs via `vectorstore.as_retriever(search_kwargs={"k": 7})`. While this works for local documents, scaling this to the entire training data of a 20-billion parameter model is computationally infeasible without approximate nearest neighbor algorithms (like HNSW), which we touched upon in discussions of efficiency.

5 Permission to Share

I am comfortable with my work (the poem and this report) being shared with classmates.

Appendix: Source Code

Below is the Python code used to generate the creative work.

```python
import os
import sys

from langchain_community.document_loaders import PyPDFDirectoryLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_chroma import Chroma
from langchain_community.llms import Ollama
from langchain_core.prompts import PromptTemplate
from langchain_core.output_parsers import StrOutputParser
from langchain_core.runnables import RunnableMap

DOCS_FOLDER = "/Users/bibek/source/vanderbilt/classes/vanderbilt-fall-2025/AI/slides"
MODEL_NAME = "gpt-oss:20b"

def main():
    print(f"--- Scanning folder: {DOCS_FOLDER} ---")

    loader = PyPDFDirectoryLoader(DOCS_FOLDER)
    docs = loader.load()

    if not docs:
        print("No PDFs found!")
        return

    print(f"Loaded {len(docs)} pages from multiple PDFs.")

    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=1200,
        chunk_overlap=300
    )
    splits = text_splitter.split_documents(docs)
    print(f"Created {len(splits)} text chunks.")

    embeddings = HuggingFaceEmbeddings(
        model_name="sentence-transformers/all-MiniLM-L6-v2"
    )
    vectorstore = Chroma.from_documents(splits, embeddings)
    retriever = vectorstore.as_retriever(search_kwargs={"k": 7})

    llm = Ollama(model=MODEL_NAME)

    prompt = PromptTemplate(
        input_variables=["context", "question"],
        template=(
            "Use the following context to answer the question.\n\n"
            "Context:\n{context}\n\n"
            "Question: {question}\n\n"
            "Answer:"
```

```
        )
    )

    # LCEL chain
    chain = (
        RunnableMap({
            "question": lambda x: x["question"],
            "context": lambda x: retriever.invoke(x["question"]),
        })
        | (lambda inputs: {
            "question": inputs["question"],
            "context": "\n\n".join(doc.page_content for doc in inputs["
    context"])
        })
        | prompt
        | llm
        | StrOutputParser()
    )

    # Run Summary
    summary_query = (
        "Synthesize the most important concepts from these slides. "
        "List the top 5 key themes and define them."
    )
    summary = chain.invoke({"question": summary_query})
    print(summary)

    print("\n--- Creative Output ---\n")

    # Run Creative Generation
    creative_query = (
    "Using the following summary of the slides:\n\n"
    f"{summary}\n\n"
    "Write a short philosophical poem about a computer trying to
    understand "
    "the world, using these key themes and technical terms as metaphors."
    )
    poem = chain.invoke({"question": creative_query})
    print(poem)

if __name__ == "__main__":
    main()
```

Listing 1: Local RAG Generation Script