Building AI Application with Gemini 2.0

Learn to create a document-based chatbot with memory, powered by one of the world's top-performing LLMs.

Abid Ali Awan

Image by Author

Google is still a strong contender in the LLM race, recently launching its most powerful and accurate multimodal model, Gemini 2.0. In this tutorial, we will explore Gemini 2.0 Flash, learn how to access it using the Python API, and build a document Q&A application with the LlamaIndex framework. Finally, we will create a RAG chatbot with memory for enhanced conversational capabilities.

Understanding Gemini 2.0

Gemini 2.0 represents a significant leap in AI technology, introducing the experimental Gemini 2.0 Flash, a high-performance, multimodal model designed for low latency and advanced capabilities. Building on the success of Gemini 1.5 Flash, this new model supports multimodal inputs (like images, video, and audio) and outputs, including text-to-speech and image generation, while also enabling tool integrations such as Google Search and code execution.

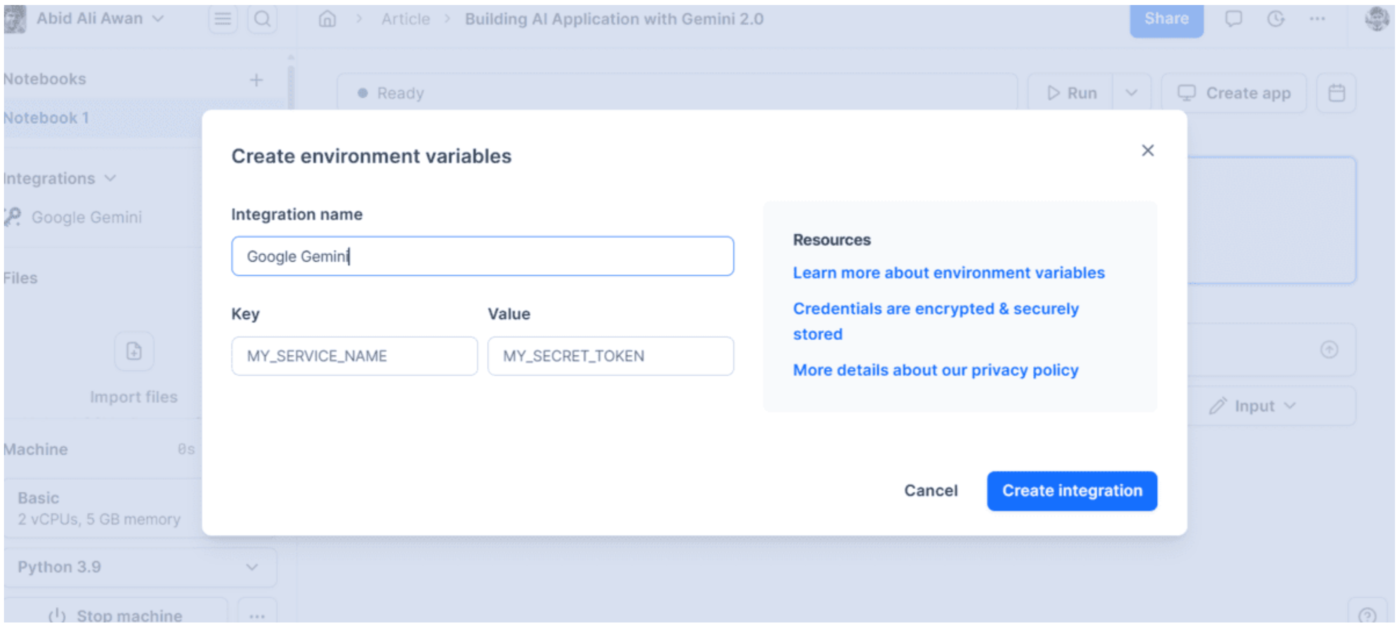
The Experimental Gemini 2.0 Flash model is available to the developers via the Gemini API and Google AI Studio and offers enhanced performance and faster response times. It also powers a more capable AI assistant in the Gemini app and explores agentic experiences.

1. Setting Up

For this project, we are using Deepnote as our coding environment to build and run the AI application. To set up the environment, we first have to install all the necessary Python packages using the PIP command.

```
%%capture
%pip install llama-index-llms-gemini
%pip install llama-index
%pip install llama-index-embeddings-gemini
%pip install pypdf
```

Then, generate a Gemini API key by going to your [Google AI Studio](#) dashboard. Finally, create the alignment variable in Deepnote and provide it with the variable name and the API key.



Create environment variables

Integration name: Google Gemini

Key	Value
MY_SERVICE_NAME	MY_SECRET_TOKEN

Resources
Learn more about environment variables
Credentials are encrypted & securely stored
More details about our privacy policy

Cancel | Create integration

2. Loading the Language and Embedding Models

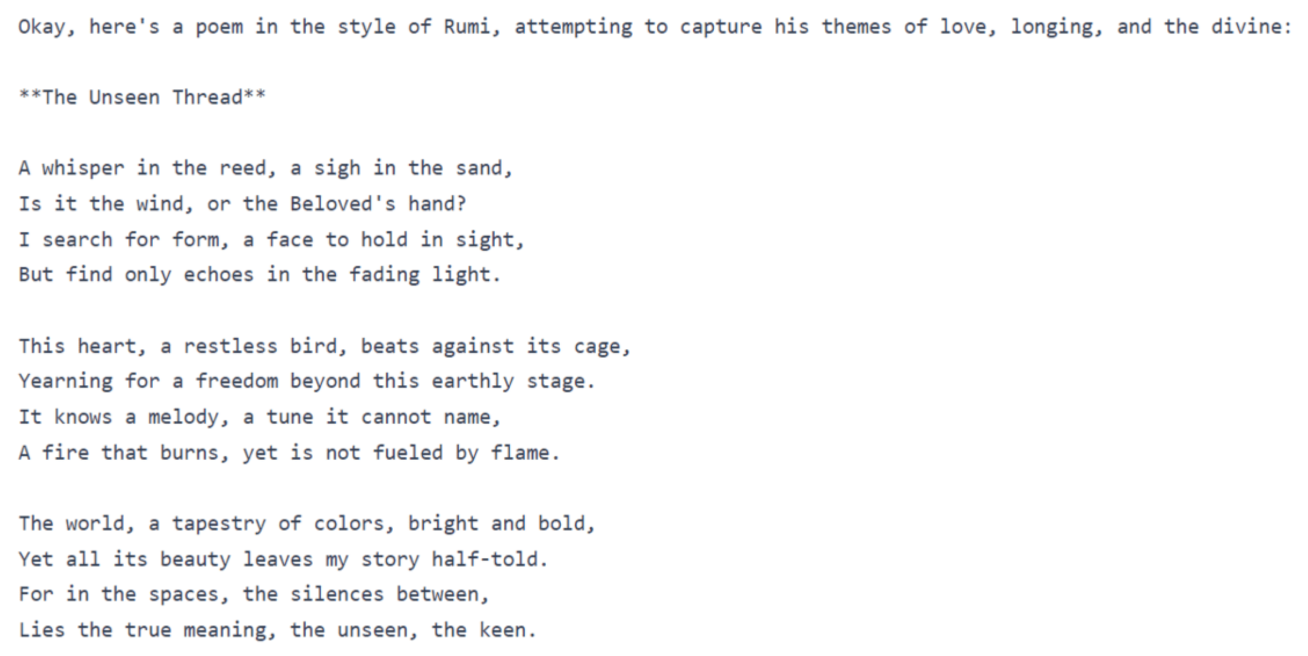
Securely load the API key and create the LLM client by providing the model name. In this case, we are using the Gemini 2.0 Flash experimental model.

```
import os
from llama_index.llms.gemini import Gemini

GoogleAPIKey = os.environ["GEMINI_API_KEY"]

llm = Gemini(
    model="models/gemini-2.0-flash-exp",
    api_key=GoogleAPIKey,
)
```

Provide the LLM client with the prompts to generate the response.

```
response = llm.complete("Write a poem in the style of Rumi.")
print(response)
```

The generated poem is perfect and has a style similar to Rumi's poems.



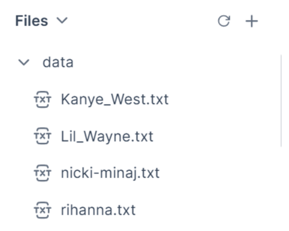
```
Okay, here's a poem in the style of Rumi, attempting to capture his themes of love, longing, and the divine:

**The Unseen Thread**

A whisper in the reed, a sigh in the sand,
Is it the wind, or the Beloved's hand?
I search for form, a face to hold in sight,
But find only echoes in the fading light.

This heart, a restless bird, beats against its cage,
Yearning for a freedom beyond this earthly stage.
It knows a melody, a tune it cannot name,
A fire that burns, yet is not fueled by flame.

The world, a tapestry of colors, bright and bold,
Yet all its beauty leaves my story half-told.
For in the spaces, the silences between,
Lies the true meaning, the unseen, the keen.
```

Next, we will load the Embedding model, which we will use to convert the text into the embedding, making it easy for us to run a similar search.

```
from llama_index.embeddings.gemini import GeminiEmbedding

embed_model = GeminiEmbedding(model_name='models/text-embedding-004')
```

3. Loading the Documentation

Load the Song Lyrics dataset from Kaggle. It consists of TXT files containing lyrics and poems by top US singers.



We will load all the TXT files using the directory reader.

```
from llama_index.core import SimpleDirectoryReader

documents = SimpleDirectoryReader('./data')
doc_txt = documents.load_data()
```

4. Building the Q&A Application

By using the Settings class, we will set the default configuration for our AI application. We are setting the LLM, embedding model, chunk size, and chunk overlap.

```
from llama_index.core import Settings

Settings.llm = llm
Settings.embed_model = embed_model
Settings.chunk_size = 800
Settings.chunk_overlap = 20
```

Convert the TXT document into the meanings and store them into the vector store.

```
from llama_index.core import VectorStoreIndex
from IPython.display import Markdown, display

index = VectorStoreIndex.from_documents(doc_txt, service_context=Settings)
index.storage_context.persist('./VectorStore')
```

Convert the index into a query engine and ask it a question. The query engine transforms the questions into embeddings, compares them with the vector store, and retrieves results with the highest similarity scores. These results are then passed through the LLM to provide detailed context.

```
query_engine = index.as_query_engine()
response = query_engine.query("Which verse do you think is the most thought-prov
display(Markdown(response.response))
```

The query engine correctly identified the answer.

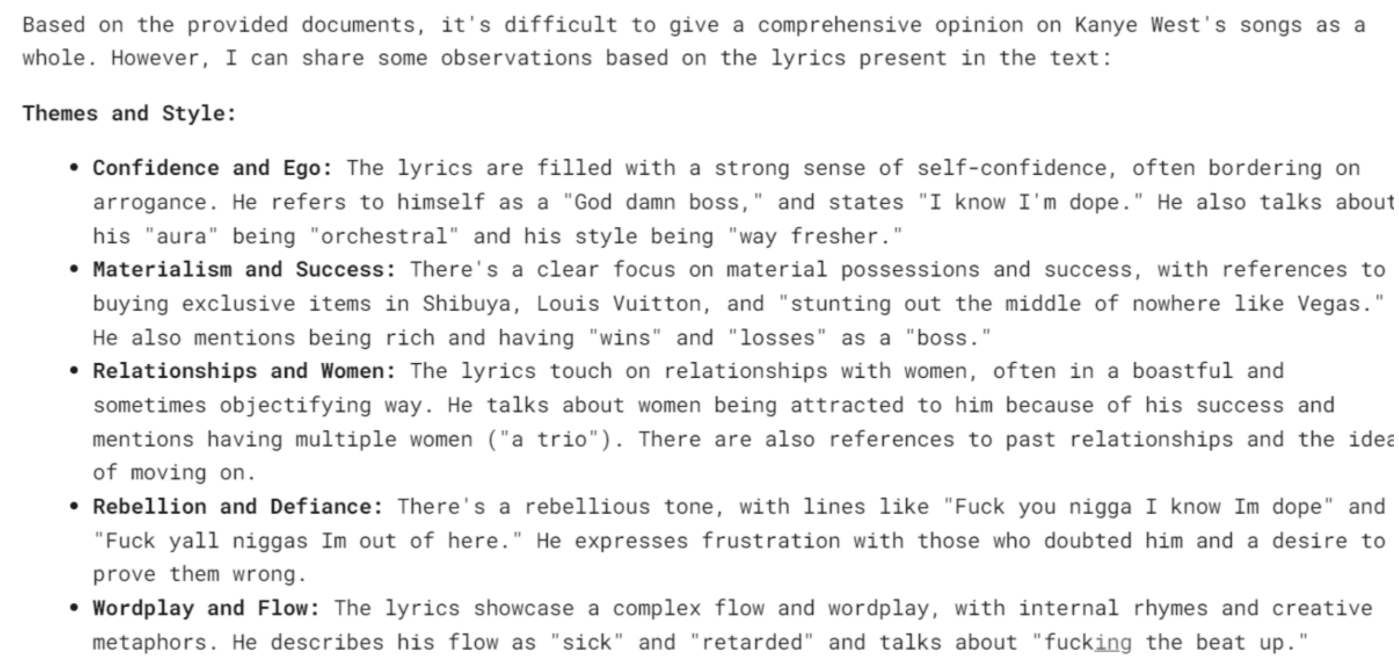
```
"Get a space where my heart was, There's a crater, I got feelings but no hard on
```

5. Building the RAG Chatbot with History

Now, let’s create a chatbot that allows back-and-forth conversations. To achieve this, we will first set up a Chat Memory Buffer to store the conversation history. Then, we will convert the index into a retriever and build a RAG (Retrieval-Augmented Generation) chatbot pipeline with memory.

```
from llama_index.core.memory import ChatMemoryBuffer
from llama_index.core.chat_engine import CondensePlusContextChatEngine

memory = ChatMemoryBuffer.from_defaults(token_limit=3900)

chat_engine = CondensePlusContextChatEngine.from_defaults(
   index.as_retriever(),
   memory=memory,
   llm=llm,
)

response = chat_engine.chat(
   "What do you think about Kanye West songs? "
)
display(Markdown(response.response))
```

The chatbot provides a context-aware answer using the song lyrics.



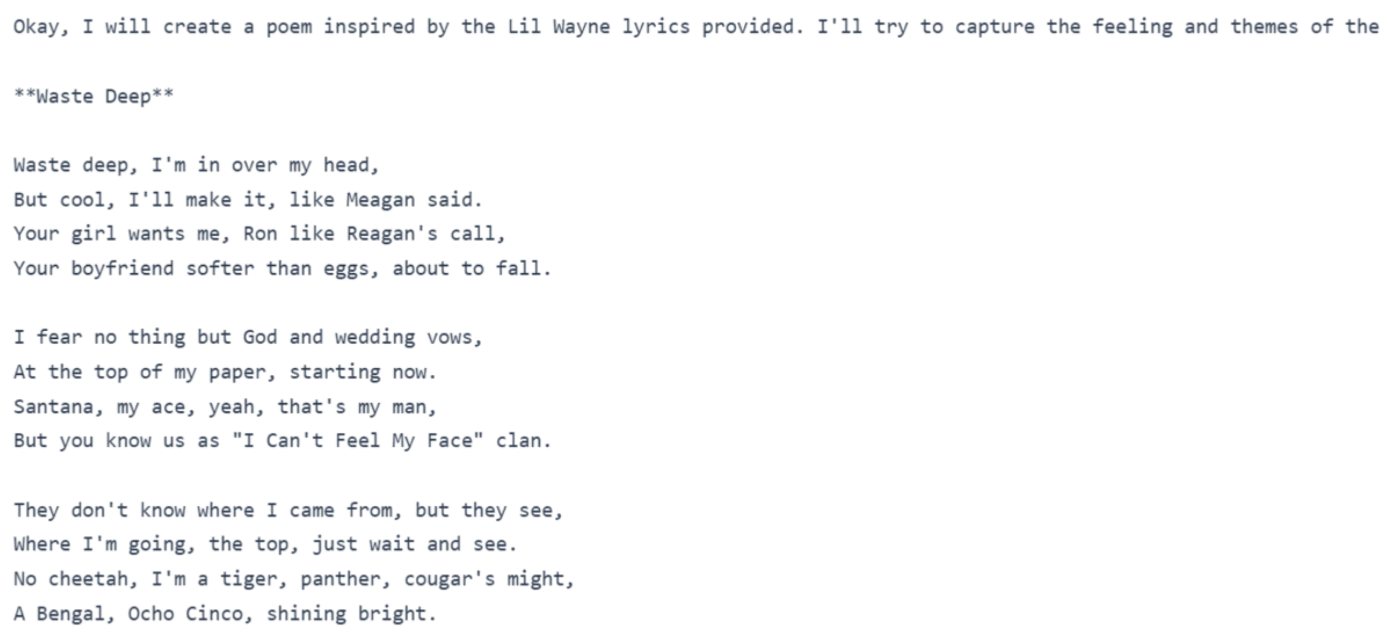
Based on the provided documents, it's difficult to give a comprehensive opinion on Kanye West's songs as a whole. However, I can share some observations based on the lyrics present in the text:

Themes and Style:

- **Confidence and Ego:** The lyrics are filled with a strong sense of self-confidence, often bordering on arrogance. He refers to himself as a "God damn boss," and states "I know I'm dope." He also talks about his "aura" being "orchestral" and his style being "way fresher."
- **Materialism and Success:** There's a clear focus on material possessions and success, with references to buying exclusive items in Shibuya, Louis Vuitton, and "stunting out the middle of nowhere like Vegas." He also mentions being rich and having "wins" and "losses" as a "boss."
- **Relationships and Women:** The lyrics touch on relationships with women, often in a boastful and sometimes objectifying way. He talks about women being attracted to him because of his success and mentions having multiple women ("a trio"). There are also references to past relationships and the idea of moving on.
- **Rebellion and Defiance:** There's a rebellious tone, with lines like "Fuck you nigga I know Im dope" and "Fuck yall niggas Im out of here." He expresses frustration with those who doubted him and a desire to prove them wrong.
- **Wordplay and Flow:** The lyrics showcase a complex flow and wordplay, with internal rhymes and creative metaphors. He describes his flow as "sick" and "retarded" and talks about "fucking the beat up."

Next, let's ask another question and generate the response as a stream. Streaming displays the response token by token..

```
response = chat_engine.stream_chat(
   "Use one of the songs to write a poem. ",
)

for chunk in response.chat_stream:
    print(chunk.delta or "", end="", flush=True)
```



```
Okay, I will create a poem inspired by the Lil Wayne lyrics provided. I'll try to capture the feeling and themes of the

**Waste Deep**

Waste deep, I'm in over my head,
But cool, I'll make it, like Meagan said.
Your girl wants me, Ron like Reagan's call,
Your boyfriend softer than eggs, about to fall.

I fear no thing but God and wedding vows,
At the top of my paper, starting now.
Santana, my ace, yeah, that's my man,
But you know us as "I Can't Feel My Face" clan.

They don't know where I came from, but they see,
Where I'm going, the top, just wait and see.
No cheetah, I'm a tiger, panther, cougar's might,
A Bengal, Ocho Cinco, shining bright.
```

Final Thoughts

Gemini models and Google AI Studio are rapidly improving and now rival the capabilities of OpenAI and Anthropic APIs. While the platform had a slow start, it now enables you to build applications that are significantly faster than those of its predecessors.

Access to Gemini 2.0 is free, allowing you to integrate it into local chatbot applications or develop full-fledged AI systems that seamlessly fit into your ecosystem. Gemini 2.0 supports text, image, audio, and even video input and offers easy tool integrations.

Abid Ali Awan (@1abidaliawan) is a certified data scientist professional who loves building machine learning models. Currently, he is focusing on content creation and writing technical blogs on machine learning and data science technologies. Abid holds a Master's degree in technology management and a bachelor's degree in telecommunication engineering. His vision is to build an AI product using a graph neural network for students struggling with mental illness.

More On This Topic

- Essential Math for Data Science: Eigenvectors and Application to PCA
- Creating a Web Application to Extract Topics from Audio with Python
- 8 Ways to Improve Your Search Application this Week
- RAG vs Finetuning: Which Is the Best Tool to Boost Your LLM Application?
- Build An AI Application with Python in 10 Easy Steps
- AnythingLLM: The LLM Application You've Been Waiting For