**Basic Chatbot using gemini and streamlit**

## Code Flow

1. **Initialization**:
   * Import necessary libraries
   * Load environment variables (like API keys)
   * Initialize the Gemini LLM (Large Language Model)
   * Set up the chat prompt template
   * Create Streamlit UI elements
2. **User Interaction**:
   * User enters a question in the text input box
   * When text is entered, the application:
     + Formats the question into the prompt template
     + Sends it to Gemini
     + Parses the output
     + Displays the response
3. **Response Generation**:
   * The processing happens through a LangChain "chain" that connects:
     + Prompt template → LLM → Output parser

## Detailed Component Breakdown

### 1. Imports

python

from asyncio import timeout # For handling timeouts (though unused in this code)

import streamlit as st # For creating the web UI

from dotenv import load\_dotenv # For loading environment variables

from langchain\_core.prompts import ChatPromptTemplate # For creating chat prompts

from langchain\_google\_genai import ChatGoogleGenerativeAI # Google Gemini integration

from langchain\_core.output\_parsers import StrOutputParser # For parsing LLM output

### 2. Environment Setup

python

load\_dotenv() # Loads variables from .env file (like API keys)

### 3. LLM Initialization

python

llm = ChatGoogleGenerativeAI(

model="gemini-1.5-pro", # The specific Gemini model to use

temperature=0, # Controls randomness (0 = more deterministic)

max\_tokens=None, # No limit on response length

timeout=None, # No timeout for API calls

max\_retries=2, # Retry failed requests twice

)

**Alternatives/Options**:

* model: Could use "gemini-pro" (older version) or future versions
* temperature: Typically between 0-1 (higher = more creative)
* max\_tokens: Can set a limit (e.g., 1000) to control response length
* Other LLMs: Could use ChatOpenAI for GPT instead of Gemini

### 4. Prompt Template

python

prompt = ChatPromptTemplate.from\_messages([

("system", "You are a chatbot"), # System message setting behavior

("human", "Question:{question}") # User's question placeholder

])

**Alternatives/Options**:

* Could add more system instructions ("You are an expert in...")
* Could include conversation history for chat context
* Could use PromptTemplate instead for non-chat prompts

### 5. Streamlit UI

python

st.title('Langchain demo with Gemini') # App title

input\_text = st.text\_input("Enter your question here!") # Input field

### 6. Chain Setup

python

output\_parser = StrOutputParser() # Converts output to simple string

chain = prompt | llm | output\_parser # Creates processing pipeline

**Alternatives**:

* Could use different parsers like JSONOutputParser for structured data
* Could add more steps to the chain (e.g., for retrieval augmentation)

### 7. Execution

python

if input\_text:

st.write(chain.invoke({'question': input\_text})) # Runs chain with user input