Creating Custom Chatbots

MGMT 675: Generative AI for Finance

Kerry Back

What is a Chatbot?

Simple Definition

A chatbot is a program that:

1. Accepts user input (prompt)
2. Sends it to an LLM via API
3. Displays the response
4. Repeats in a loop

- ChatGPT, Claude, Gemini are chatbots
- The LLM is the "brain"
- The chatbot is the interface
- You can build your own!

The API: Talking to an LLM

An **API** (Application Programming Interface) lets your code communicate with an LLM service over the internet.

How It Works

1. Your code sends a request
2. Request includes your prompt
3. LLM processes the prompt
4. API returns the response
5. Your code displays it

What You Need

- API endpoint (URL)
- API key (authentication)
- Model name to use
- Your prompt/messages

OpenRouter: One API, Many Models

What is OpenRouter?

- Unified API for 100+ models
- OpenAI, Anthropic, Google, Meta, etc.
- Single API key for all models
- Some models are free!

`openrouter.ai`

Free Models on Hugging Face

- Hugging Face hosts open models
- OpenRouter provides free access
- Examples:
  - `mistralai/mistral-7b-instruct:free`
  - `meta-llama/llama-3-8b-instruct:free`
  - `google/gemma-7b-it:free`

A Single API Call

Basic Structure

```
import requests

response = requests.post(
    "https://openrouter.ai/api/v1/chat/completions",
    headers={"Authorization": f"Bearer {API_KEY}"},
    json={
        "model": "mistralai/mistral-7b-instruct:free",
        "messages": [{"role": "user", "content": prompt}]
    }
)
answer = response.json()["choices"][0]["message"]["content"]
```

The Problem: LLMs Have No Memory

Each API call is independent. The LLM doesn't remember previous messages unless you send them again.

Without History

- User: "My name is Alice"
- LLM: "Nice to meet you, Alice!"
- User: "What's my name?"
- LLM: "I don't know your name."

With History

- Send all previous messages
- LLM sees full conversation
- Can reference earlier context
- Feels like a real conversation

The Conversation Loop

To create a chatbot, you need a loop that maintains the conversation history.

The Pattern

1. Initialize empty message list
2. Get user input
3. Append user message to list
4. Send **entire list** to API
5. Get response from API
6. Append assistant response to list
7. Display response
8. Go to step 2

- History grows each turn
- LLM sees everything
- Context is preserved
- This is how ChatGPT works!

Message Format

Messages Are a List of Dictionaries

```
messages = [
    {"role": "user", "content": "My name is Alice"},
    {"role": "assistant", "content": "Nice to meet you!"},
    {"role": "user", "content": "What's my name?"}
]
```

Roles

- `user`: Human messages
- `assistant`: LLM responses
- `system`: Instructions (special)

- Each message has role + content
- Order matters (chronological)
- Send full list each API call

The System Prompt

The **system prompt** is a special message that defines the chatbot's personality, knowledge, and behavior. It's the key to customization.

What It Does

- Sets the chatbot's persona
- Defines its expertise
- Specifies response style
- Adds domain knowledge
- Sets guardrails/rules

Placement

- First message in the list
- Role is `system`
- Sent with every API call
- User never sees it directly

System Prompt Examples

Finance Tutor

```
You are a patient finance
tutor. Explain concepts
simply. Use examples with
real companies. Always
check understanding before
moving on.
```

Investment Analyst

```
You are a senior equity
analyst. Be concise and
data-driven. Always cite
sources. Flag risks and
uncertainties clearly.
```

The system prompt transforms a generic LLM into a specialized assistant

Complete Message Structure

Messages with System Prompt

```
messages = [
    {"role": "system", "content": "You are a helpful
        finance tutor..."},
    {"role": "user", "content": "What is a P/E ratio?"},
    {"role": "assistant", "content": "A P/E ratio is..."},
    {"role": "user", "content": "How do I interpret it?"}
]
```

- System prompt stays at the beginning
- User and assistant messages alternate
- Entire list sent with each API call

Putting It All Together

Chatbot Architecture

Component	Purpose
System Prompt	Defines chatbot personality/expertise
Message List	Stores conversation history
Loop	Continuously gets input, calls API, shows output
API Call	Sends messages, receives response

These four components create any custom chatbot

Exercise: Build a Finance Chatbot

Build and deploy a Streamlit chatbot using OpenRouter and a free model.

Requirements

- Custom system prompt
- Conversation history
- Streamlit interface
- Deploy to Koyeb

Ask Claude

"Create a Streamlit chatbot app using OpenRouter with [model]. Make it a [describe persona]. Deploy to Koyeb."

Summary

Key Concepts

- API calls send prompts to LLMs
- LLMs have no memory
- You must send full history
- System prompt customizes behavior

Building Blocks

- OpenRouter API
- Message list (history)
- Conversation loop
- System prompt

A chatbot = Loop + History + System Prompt + API