Welcome to the AI Workshop! 🚀

Day 1: LLM APIs & Code Basics

Hi, I'm Taloot Khan! 👋

- ✅ Backend Engineer with 5 years experience
- ✅ Working with AI & LLMs for 2 years
- ✅ Love football ⚽
- ✅ Passionate about making AI accessible to everyone!

Today: Zero → AI-Powered Chatbot! 🎯

Let's Get to Know Each Other! 👥

Please Share:

1. Your name
2. Your programming background
   (Beginner/Intermediate/Advanced)
3. What do you hope to learn today?
4. Have you used ChatGPT or other AI tools before?

💡 **Don't worry if you're a beginner!**
We're starting from the ground up. Everyone is welcome! 🎉

Should We Be Scared of AI? 🤖

NO! But we should be informed ✅

AI is a Tool 🛠️

Just like the internet, calculators, or programming languages. It's here to amplify what we can do!

Not a Replacement ❌

AI won't replace you. Someone who knows how to use AI will replace you if you don't learn it!

The future belongs to those who work WITH AI 🚀

AI in Industry: The Basics 🎓

1. Large Language Models (LLMs)

- 🤖 Trained on **billions of words**
- 🧠 Predict the next word in a sequence
- 💡 Examples: GPT-4, Claude, Gemini

2. How They Work (Simplified)

```
Input (Your Question) → Processing (Neural Network) → Output (Generated Text)
```

3. Common Use Cases in Industry

- 💬 Customer support chatbots
- 💻 Code generation (GitHub Copilot)
- ✍️ Content creation
- 📊 Data analysis & summarization

🔑 **APIs are the bridge!** They let your code talk to AI models. Like ordering food - you request, kitchen prepares, you receive! 🍕

Step 1: Create Your OpenRouter.ai Account 🔑

Instructions:

1. Go to: **https://openrouter.ai**
2. Sign up for a **free account**
3. Navigate to: **API Keys** section
4. Create a new API key
5. **Copy your API key** - you'll need it soon!

⚠️ **Keep it secret!** Don't share your API key publicly. Treat it like a password!

Why OpenRouter? 🤔

- ✅ Access to multiple models
- ✅ Simple, unified API
- ✅ Free credits to start
- ✅ Perfect for learning!

Take 5 minutes - raise your hand when done! 🖐

Step 2: Get the Workshop Code 📦

1. Clone the Repository

```
git clone [repository-url]
cd 9hour_workshop_complete
cd day1
```

2. Install Dependencies

```
pip install openai python-dotenv requests
```

3. Create .env File

```
OPENAI_API_KEY=your-api-key-here
```

⚠️ **Important:** The `.env` file should NOT be committed to Git. It's already in `.gitignore` to keep your key safe!

💡 **Don't have Git?** Download the repository as a ZIP file from GitHub instead!

Exercise 1: Your First API Call 🎯

What You'll Learn:

- ✅ Load environment variables securely
- ✅ Create an API client
- ✅ Make your first call to an LLM
- ✅ Extract and display the response

📁 File to Open:

```
day1/exercises/
01_first_api_call_starter.py
```

⏱️

Time:

20

minutes

The foundation for everything else! 🏗️

Exercise 1: Solution Explained 💡

1. Load API Key

```
from dotenv import load_dotenv
load_dotenv()
api_key = os.getenv("OPENAI_API_KEY")
```

2. Create Client

```
client = OpenAI(api_key=api_key)
```

3. Make API Call

```
response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        {"role": "user", "content": "Say
hello..."}
    ]
)
```

4. Extract Response

```
ai_message =
response.choices[0].message.content
print("AI Response:", ai_message)
```

💡 **Key Concept:** The response structure supports conversations with multiple turns. We'll see more of this in Exercise 3!

Exercise 1: Now It's Your Turn! 💻

Your Tasks:

1. Open **01_first_api_call_starter.py**
2. Fill in the TODOs
3. Run it: `python 01_first_api_call_starter.py`
4. You should see the AI's response! 🎉

Common Issues:

- ❌ "API key not found"
  → Check your .env file
- ❌ "Module not found"
  → Run pip install
- ❌ Connection error
  → Check internet

Need Help?

- ✅ Check solution file
- ✅ Ask me or neighbors
- ✅ Compare your code

Take your time! Understanding > Speed 🧠

Exercise 2: Prompt Engineering 🎨

What You'll Learn:

✅ System messages (giving AI a role)

✅ Few-shot prompting (teaching by example)

✅ Temperature parameter (controlling creativity)

✅ Structured outputs (getting JSON responses)

The difference between good AI and great AI is in the prompts! 🎯

📁 File:

`02_prompt_engineering_starter.py`

⏱️

Time:

30

minutes

💼 **Industry Insight:** Companies hire "Prompt Engineers" who specialize in this. It's a real skill!

Exercise 2: Key Concepts 🧠

1. System Messages

```
{"role":
"system",
"content":
"You are a
friendly
teacher..."}
```

Sets AI's personality/role. Provides context for all responses.

2. Few-shot Prompting

```
Show examples
→
AI learns
pattern →
Consistent
outputs
```

Teach by example. Very powerful for consistent results!

3. Temperature

```
temperature=0.2
→ Focused
temperature=1.5
→ Creative
```

Range: 0.0 to 2.0
Lower = predictable
Higher = creative

4. Structured Outputs

```
"Format as
JSON
with
'benefits'
array"
```

Request specific formats in your prompt. AI will try to follow!

Exercise 2: Implement Prompt

Engineering 

Your Tasks:

1. Open
   02_prompt_engineering_starter.py
2. Implement 5 functions:
   - basic_prompt()
   - prompt_with_context()
   - prompt_with_examples()
   - temperature_experiment()
   - structured_output_example()
3. Run it and observe the differences!

Experiment! 🧪

- Compare with/without system messages
- See how temperature affects creativity
- Notice how examples change style

⏱️ Time:

30 minutes

Don't just copy - experiment! Try changing prompts and see what happens.

Exercise 3: Building a Chatbot with Memory 🧠

What You'll Build:

- ✅ A Python tutor chatbot
- ✅ Remembers what you talked about
- ✅ Can reference previous messages
- ✅ Has a personality (friendly tutor)
- ✅ Can show conversation history
- ✅ Can clear history and start fresh

This is how real chatbots work! 💬

📁 File:

`03_chatbot_memory_starter.py`

⏱️ Time:

40

minutes

🔑 **Key Insight:** Every message sends the ENTIRE conversation history. That's how the AI remembers context!

Exercise 3: Solution Architecture 🏗️

1. Class Structure

```
class SimpleChatbot:
    def __init__(self, system_prompt):
        self.conversation_history = [
            {"role": "system", "content":
system_prompt}
        ]
```

2. The Chat Method

```
1. Add user message to history
2. Send ENTIRE history to API
3. Get response
4. Add assistant response to history
5. Return response
```

3. Why Send Full History?

> The API needs to see all previous messages to understand context. If you only sent the latest message, the AI wouldn't know what you're referring to!

💡 **The Magic:** Because we send the full history, you can ask follow-up questions like "Can you explain that differently?" and the chatbot knows what "that" refers to!

Exercise 3: Build

Your Chatbot! 

Your Tasks:

1. Open **03_chatbot_memory_starter.py**
2. Implement the SimpleChatbot class:
   - __init__() - Initialize with system message
   - chat() - Add message, call API, return response
   - clear_history() - Reset conversation
3. Test it and have a conversation!

Test Scenarios:

- ✅ Ask a question
- ✅ Ask a follow-up
- ✅ Check if it remembers
- ✅ View history
- ✅ Clear and restart

⏱️ Time:

40 minutes

This is the most complex exercise, but you've got all the building blocks!

🎯 **The Test:** If you ask "Can you give me a Python example?" after asking about variables, and it gives a variable example (not a loop), then your memory is working! 🎉

Congratulations! 🎉

What You've Accomplished:

- ✅ Made your first API call to an LLM
- ✅ Learned prompt engineering techniques
- ✅ Built a chatbot with conversation memory
- ✅ Understood how AI APIs work in practice

Key Takeaways:

- ✅ AI is a tool you can program
- ✅ Prompt engineering matters
- ✅ Context = conversation history
- ✅ APIs bridge code & AI

Next Steps:

- 📚 Review solutions
- 🧪 Experiment more
- ⭐ Try bonus exercise
- 🚀 Prepare for Day 2!

You're not just using AI - you're building it! 🚀

💬 **Questions?** Let's discuss what you learned and any questions you have!