Generative AI

Trainer: Nilesh Ghule

Local LLM with LM Studio & Prompt Engineering

- Goal:
  - Move from cloud-only LLMs to running open-source models directly on our machine.
  - Learn to design effective prompts that guide LLMs to produce accurate, useful outputs.
- Today:
  - Local LLMs, LM Studio, quantization, GGUF formats, and chatting with a small model.
  - Zero-shot, one-shot, few-shot, role prompting, and chain-of-thought prompting.
- Objectives:
  - Understand the trade-offs between API-based LLMs and local LLMs.
  - Explain what LM Studio is and why quantization and GGUF formats matter.
  - Download, load, and chat with a small quantized model such as Phi-4 Mini.
  - Write superior prompts to get exact desired output.

Nilesh Ghule

Cloud APIs vs Local LLMs

- Cloud based APIs:
  - Powerful models
  - Low latency
  - Managed infrastructure
  - Paid - heavy usage can be costly
  - Data security concerns
- Local LLMs:
  - Works offline
  - Data remains private - on our system
  - No per-call cost
  - Model size and speed are limited by hardware capabilities

What is LM Studio?

- Desktop app for Windows, macOS, and Linux to open-source LLMs locally.
- Key Features
  - Local LLM Execution: Run powerful models like Llama directly on your laptop offline
  - Discovery: Browse and download compatible models from Hugging Face within app
  - Chat Interface: Interact with downloaded models through a built-in chat
  - Offline Functionality: Ensuring privacy and no token costs for local use
  - Local Server: Expose models via an OpenAI-compatible API
  - Performance Optimization: Supports GPU acceleration
  - Resource Management: Fine-tune settings for model performance
- Target users
  - Developers, Privacy-Focused Users
- LM studio alternatives: Ollama, Jan.ai, GPT4All, …

Quantization - Key Idea

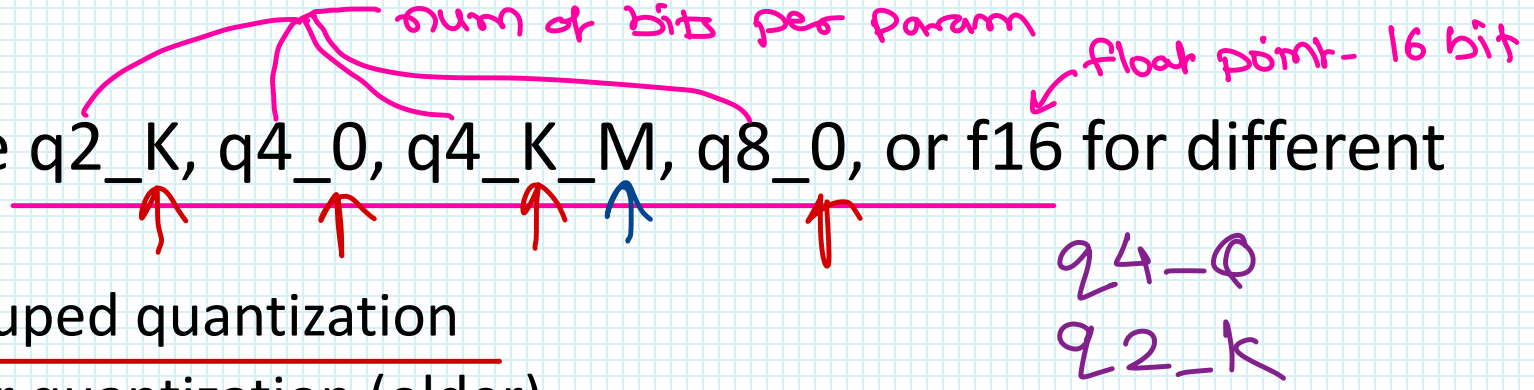
70b param
↳ 70,000,000,000
param size = 32 bit = 4byte
llm size = 280,000,000,000

- LLM parameters are normally stored as 16-bit or 32-bit floating-point numbers, which use a lot of memory.
- Quantization stores parameters with fewer bits (for example 4-bit), reducing model size and memory usage.
- Small loss in accuracy, big gain in feasibility for consumer hardware.

- For small machines (like Core i5 with 8 GB RAM + optional entry-level gaming GPUs), we can use small or heavily quantized models and accept slower responses compared to cloud APIs.

GGUF Model Format and Variants

- GGUF is a file format for quantized models optimized for CPU-based backends.
- File names include hints like q2_K, q4_0, q4_K_M, q8_0, or f16 for different quantization levels.
  - _K (K-quant) - Advanced grouped quantization
  - 0, 1 - Less accurate but faster quantization (older)
  - _M (Medium precision) - Between Small precision (fastest with low accuracy) to Large precision (Slowest with higher accuracy)
- For low hardware machines, q4 variants are usually the best balance between size and quality.
- Choose a starter model like Phi 4 Mini.
- LM studio also recommends models suitable for machine hardware.



LM Studio - Demo

- LM Studio UI
  - Search view: Find models from Hugging Face using keywords and filter by size/source.
  - Model details: Inspect GGUF files, sizes, quantization levels, and compatibility hints.
  - Chat view: Load a model, monitor memory usage, and interact via a chat interface.
- Download model
  - Search small instruct model such as Phi-4 Mini in the LM Studio search view.
  - Open the model, pick a q4_K_M or similar GGUF file, and download it
- Chat with model
  - Load the downloaded model in the chat view, wait for initialize, and begin chatting
- Tuning Inference Parameters
  - Adjust temperature, maximum tokens, and presets to see how they affect response
  - On machines with GPUs, experiment with turning GPU acceleration on or off.
- Local Server Mode
  - To connect with User applications (via REST apis/Langchain)

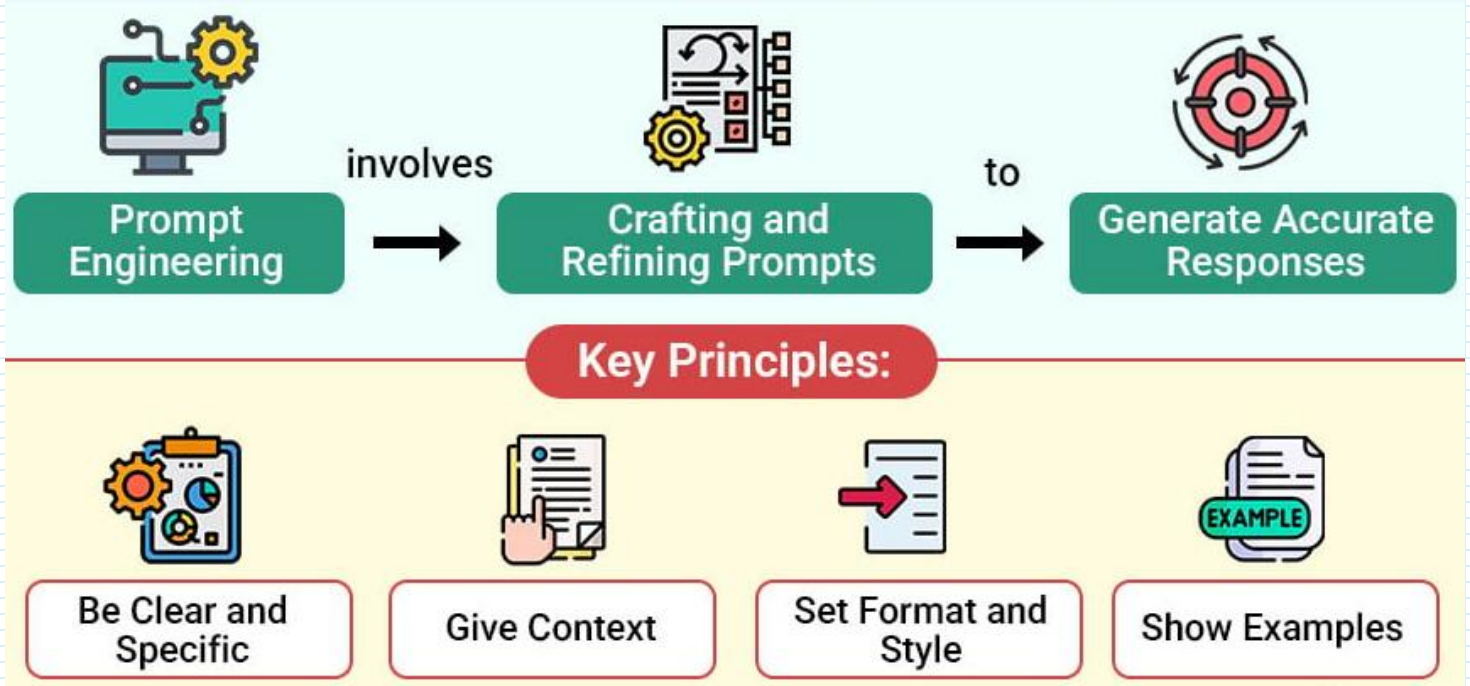
Code: Calling the LM Studio Local Server

LM Studio Local Server exposes REST apis compatible with OpenAI endpoints (urls).

```
import json
import requests

url = "http://localhost:1234/v1/chat/completions"
headers = {
    "Content-Type": "application/json",
    "Authorization": "Bearer not-needed"
}

data = {
    "model": "local-model",
    "messages": [
        {"role": "user", "content": "Explain what quantization is in simple terms."}
    ]
}

response = requests.post(url, headers=headers, data=json.dumps(data))
print(response.json())
```

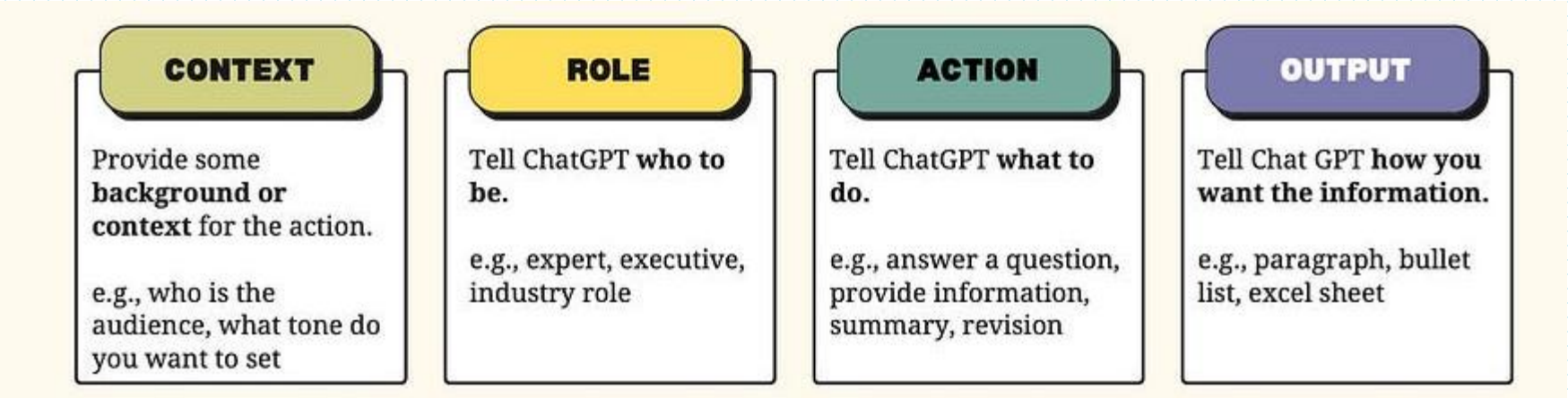
Prompt Engineering

- Prompt engineering is practice of designing and refining inputs to get desired outputs from LLMs. Small changes in wording, structure, or examples can dramatically change the quality of responses.
- Well-crafted prompts increase relevance, accuracy & coherence of model o/p
- Poor prompts lead to vague, off-topic, or hallucinated answers.



Anatomy of a Good Prompt

- Clear task: What should the model do (for example summarize, classify, explain, generate code).
- Context: Relevant background or description of the situation or audience.
- Input: The actual text, data, or question the model must process.
- Output format and constraints: Length, style, structure, or response language.



Zero-shot Prompting

- Zero-shot: You provide instructions and input, but no examples of the desired output.
- Works well for simple tasks like basic summarization or explanation.
- Examples:
  - Prompt: "Classify the text as positive, negative, or neutral.
    Text: 'The movie was okay.'" Output: "Neutral".
  - Prompt: "Summarize the following article in one sentence.".
  - Prompt: "Translate 'How are you?' into Spanish."
- Not suitable for complex reasoning (multi-step logic) or deep context tasks

One-shot and Few-shot Prompting

- One-shot: Include a single example of input and desired output in the prompt.
  - Prompt: "Translate English to French: sea otter => loutre de mer. Now translate: cheese =>"
  - Suitable when you need a specific output style but have limited data, offering a balance between simplicity and specificity.
- Few-shot: Include several examples to strongly steer the model toward a pattern or style.
  - Prompt:

    Review: "This phone is amazing!" Sentiment: Positive

    Review: "The service was awful." Sentiment: Negative

    Review: "It's okay, but the price is high." Sentiment: Mixed

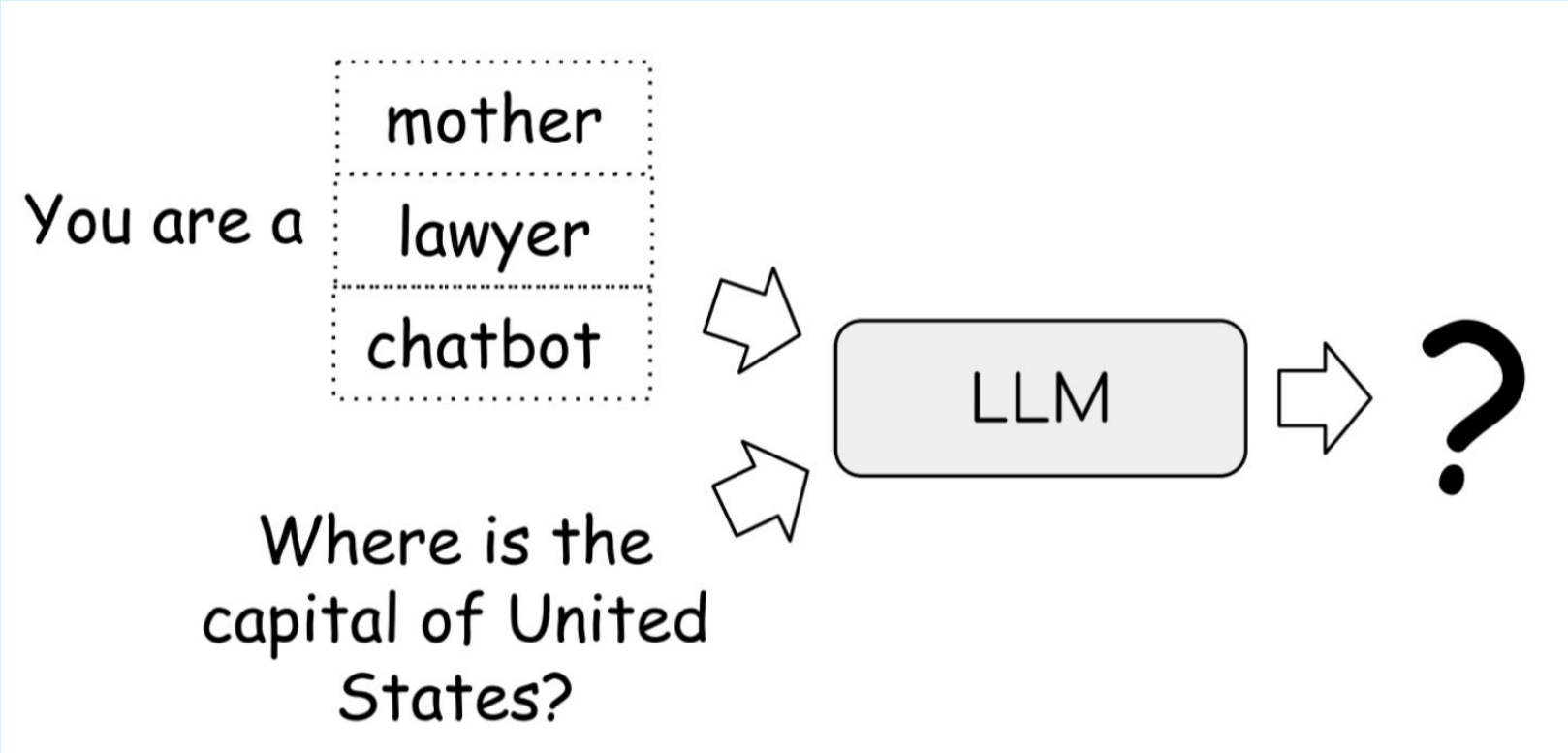
    Review: "Best purchase ever!" Sentiment:
  - Suitable for tasks needing nuanced understanding or consistent formatting where multiple examples improve generalization and accuracy.

Chain-of-Thought (CoT) Prompting

- Chain-of-thought prompts encourage the model to show intermediate reasoning before the final answer.
- Works well with multi-step logic
- Example:
  - Prompt: "I bought 10 apples, gave 2 away, bought 5 more, and ate 1. How many apples do I have? Let's think step by step."
- Often implemented by giving one or more worked examples with step-by-step reasoning.
  - You can specify exact formatting.

Role Prompting and Style Control

- Specify a role: for example 'You are a senior Python developer' or 'You are a strict grammar teacher.'
- Ask for a style: for example 'Explain step by step', 'Use bullet points', or 'Use simple English for beginners.'



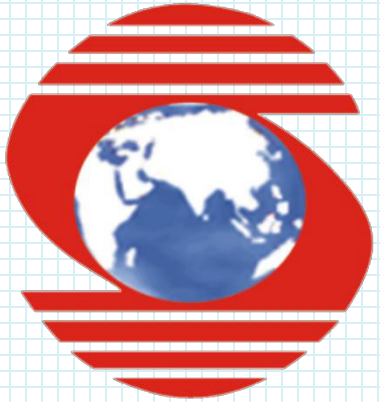
Code: Structured Prompt in Python (Groq)

- **Message Roles**
  - System: Sets overall behavior and rules.
  - User: Contains actual questions or instructions from end user.
  - Assistant: Previous model responses to maintain conversation context.

```
load_dotenv()
api_key = os.getenv("GROQ_API_KEY")

url = "https://api.groq.com/openai/v1/chat/completions"
headers = {
    "Authorization": f"Bearer {api_key}",
    "Content-Type": "application/json"
}

data = {
    "model": "llama3-8b-8192",
    "messages": [
        {"role": "system", "content": "You are a helpful tutor who
explains concepts step by step."},
        {"role": "user", "content": "Explain what an API is using a
restaurant analogy."}
    ]
}

response = requests.post(url, headers=headers,
data=json.dumps(data))
print(response.json())
```

Remember this

- Common Pitfalls
  - Vague prompts: Too generic or underspecified; fix by adding clear task, context, and constraints.
  - Overloaded prompts: Many tasks in one request; fix by breaking into smaller, focused prompts.
- Iterative Prompt Refinement
  - Treat prompt design as an experiment: run, observe, refine, and repeat.
  - Record effective prompts and learn from failures to build a reusable prompt library.
- Prompt engg Libraries and tools
  - Libraries: Langchain enables you to pull predefined prompts from langchain hub
    - from langchain import hub
    - prompt = hub.pull("hwchase17/eli5-solar-system")
  - Free tools for writing prompts: PromptLab, Agenta, …



Thank You!

Nilesh Ghule

<nilesh@sunbeaminfo.com>