Large Language Model

LLM Development Toolkit

Lecture 02 - Part 2 (Upgraded)

April 21, 2025

Agenda

- Introduction: LLMs as a Toolkit
- Prompt Engineering
- Understanding LLM Architectures
- Tools: Open-Source and Paid
- Setting up LLMs Practically
- Agentic AI and LLMs
- Summary and Q&A

Why Think of LLMs as a Toolkit?

Build on top of transformers.

- **LLMs are not magic**: They require crafted prompts, proper infrastructure, and optimization for reliable operation. Raw models alone are not enough.
- **Building applications**: Real-world LLM apps combine models, retrieval systems (RAG), external tools, and pipelines.
- **Practical engineering is crucial**: Success depends on handling scaling, failures, latency, and integration challenges.

Prompt Engineering: Introduction

Prompt is what guide the LLM to provide the answer. If we have very big prompts written, then the answers you get would be big as well. You can think of a prompt like a structured set of instructions that you get LLM to generate an output. If you have good prompts, you will get better outputs.

- **Prompt as control**: A prompt directs what the LLM should focus on — it's like writing a mini-program through text.
- **Good prompts = Better outputs**: Specific, well-structured prompts yield much more reliable and useful results.
- **Programming by language**: Unlike code, prompts shape behavior dynamically through careful wording.

Prompt Engineering Techniques

System prompt is like when you build or when you customize LLM you are setting a global behavior.
COT is more like prompting in a step-by-step manner instead of just asking for an answer directly.
Few-Shot prompting is when you give set of instructions or examples like an example, you want to get a LLM to generate some sample questions you to practice for the exam. So, what you can do is you can ask to give all the specifications you want at the end as an example and then give past questions based on that particular structure basically.

- **System Prompts**: Set global behavior (e.g., You are an expert legal advisor.)
- **Chain-of-Thought (CoT)**: Ask the model to reason step-by-step instead of jumping to conclusions.
- **Few-Shot Prompting**: Give examples inside the prompt to teach the task style.
- **Zero-Shot Prompting**: Expect the model to generalize without examples.
- **Tool-Use Prompting**: Direct models to call APIs, calculators, or plugins when needed.

Reference: Chain-of-Thought Paper

Zero-Shot Prompt is not an ideal way of doing things, it is like you just give an instruction hoping for the best that gives best possible answer.
Tool-Use Prompt is when you want a model, LLM rather LLM agent to basically for other API and based on those APIs or interacting with different plugins, APIs and do whatever you require or whatever you requesting to do.

Designing Effective Prompts

- **Be clear and specific**: Avoid ambiguity. Tell the model exactly what you expect.
- **Provide context and background**: The more the model knowsäbout the situation, the better it can respond.
- **Define the output format**: Specify if you want an answer as text, bullet points, table, or code.
- **Give examples**: Show the model what a good output looks like through in-context learning.

Examples of Good Prompts

Bad Prompt:

- "Tell me about history." — Too vague, invites random responses.

Good Prompt:

- "Summarize the history of naval warfare in 5 bullet points, focusing on major innovations between 1500 and 1900."

Tip: Use role-play setups, e.g., "Act as a historian specializing in naval technology."

Why Prompt Engineering Matters

- **Prompt quality = Output quality**: Clear prompts reduce errors, hallucinations, and vague responses.
- **Small changes, big impact**: Even wording order can drastically change results.
- **Essential for serious apps**: Chatbots, tutoring systems, search engines all depend on prompt quality.

LLM Architecture Essentials

- **Tokenizer**: Breaks text into tokens (smaller units of words/symbols) for the model to process.
- **Embedding Layer**: Converts tokens into high-dimensional vectors carrying semantic meaning.
- **Transformer Block**: Core engine — uses self-attention to model relationships between words.
- **Output Head**: Maps model predictions back into human-readable text.

all the input

as well as word order.

Reference: Attention is All You Need

In most cases what we are using is multi-headed attention, so we have multiple output heads that focuses on different parts of the text. So, we are able to get different outputs and then we combine them to get the best one out of that focus on get the output.

Popular Open-Source and Paid Tools

Open-Source:

- llama.cpp: Lightweight local LLM inference.
- Ollama: Easy install-and-use local LLM runner.
- Hugging Face Models: Huge repository of pre-trained models.
- LangChain: Framework for chaining prompts, models, APIs together.
- Haystack: End-to-end production-ready RAG system.

Paid/Hosted:

- OpenAI API: Direct API access to powerful models (GPT-4, GPT-3.5).
- Anthropic Claude: Safer, instruction-following models.
- OpenRouter: Unified API gateway for many LLM providers.

Strengths and Weaknesses

- **Open-Source**: Free, customizable, but needs technical setup and hardware.
- **Paid APIs**: Quick and powerful, but costly and less flexible for special needs.
- **Local LLMs**: Offer privacy and control, but may struggle with model size and speed.
- **Cloud-hosted LLMs**: Scale easily but introduce latency, API limits, and higher operational costs.

Setting Up LLMs: Practical Considerations

- **Model Size Matters**: 7B models need 8-16 GB RAM, 65B models need high-end GPUs.
- **Quantization Helps**: Shrinks models by using lower precision (int8/fp16) without major accuracy loss.
- **Memory/Compute**: Plan based on expected load — inference can be memory intensive.
- **Latency Tradeoffs**: Smaller models are faster but less powerful. Match model size to use case.

Reference: Huggingface Quantization Guide

Typical LLM Workflows

- **Small Projects**: Ollama + LangChain for prototyping local assistants.
- **Enterprise Apps**: vLLM server + Haystack RAG for scalable solutions.
- **Cloud Deployments**: OpenAI API + lightweight frontend apps.

Workflow Example: User Query $\rightarrow$ Retriever (Docs) $\rightarrow$ Enriched Prompt $\rightarrow$ LLM $\rightarrow$ Final Output

Recommended Starting Points

- Use OpenAI API + LangChain to quickly build working prototypes.
- Use Ollama to experiment and fine-tune small LLMs offline.
- Use RAG pipelines to enhance retrieval accuracy and answer relevance.
- Quantize large models to run them on moderate hardware.

Moving Beyond Static Prompts

What is Agentic AI?

- **Agentic AI**: Models that plan, reason, act, and adapt across steps — not just single reply generation.
- LLM becomes part of a feedback loop rather than a one-shot answer engine.
- Crucial for complex tasks like research assistants or autonomous bots.

Why Agents?

- **Limit of Static Prompts**: Can't adapt if the situation changes mid-process.
- **Agents handle complexity**: Plan $\rightarrow$ Act $\rightarrow$ Observe $\rightarrow$ Re-plan if needed.
- **Adaptive workflows**: Adjust course of action dynamically based on results.

Components of an LLM Agent

- **Memory**: Stores prior actions, results, and conversation state.
- **Tools**: API access, databases, search engines, calculators.
- **Planner**: Decomposes goals into achievable sub-tasks.
- **Executor**: Executes actions, monitors outputs, and feeds back into planning.

How LLM Agents Work (Simplified Loop)

1. Receive goal from user.
2. Plan initial action or query.
3. Call necessary tools / generate intermediate outputs.
4. Observe results, analyze gaps.
5. Re-plan or finalize output.

Inspired by frameworks like LangGraph, AutoGPT.

Example Agent Frameworks

- AutoGPT: Autonomous goal-driven agents.
- LangChain Agents: Modular action planning.
- BabyAGI: Recursive task list managers.
- OpenAI Assistants API: Natively build agents with OpenAI models.

Use Cases of LLM Agents

- Research automation.
- Multi-step data processing and analysis.
- Dynamic task orchestration (retrieval + writing + verifying).
- Customer support assistants.

Practical Considerations

- **Cost**: Multi-step agent plans mean more token usage $\rightarrow$ higher billing.
- **Latency**: Each decision cycle adds time.
- **Failure Handling**: Agents must handle retries, dead-ends, unexpected responses.
- **Safety**: Critical to bound agent behaviors to avoid misuse.

Summary

- LLM success = Combining prompts, architecture, retrieval, tools, and agents.
- Choose right size models and tools for your needs.
- Master Prompt Engineering, Quantization, RAG, and Agentic AI for future-proof systems.

Questions?