Applied Generative AI : LLM Application Development

Bhattaraprot Bhabhatsatam, Ph.D.

https://bhattaraprot.com

Dates: 12 & 19 September 2025

Applied Generative AI : LLM Application Development

- **Objective :** Build practical skills in enhancing LLMs for real-world apps (with a focus on study examples)
- **Warning:** AI is a rapidly evolving tech landscape
  - Tools and techniques we cover today may become obsolete tomorrow
  - Follow updates from sources like Hugging Face, OpenAI, or X
- **Course Philosophy:**
  - Fundamentals first
  - All examples are simplified for learning purposes
  - Not intended for production use—adapt and test in real scenarios
  - Understanding core concepts empowers you to adapt to future changes

Course Overview

Structure:

- Plain LLMs: Basics and simple enhancements
- RAG: Retrieval-Augmented Generation for knowledge integration
- MCP: Multi-Context Prompting for complex interactions
- Memory: Persistent state for conversational apps

Hands-On Focus:

- Code snippets, demos, and exercises
- Prerequisites: Basic Python, familiarity with APIs
- Tools We'll Use: Internet, Ollama, Python

Course Positioning — Fundamentals First

This Course

- Learn how **RAG, MCP, and Memory** work **from scratch**
- Build everything with **Python + Ollama + Streamlit**
- Understand the **core mechanics** before adding automation layers
- Transparency: see every step → chunking, embedding, retrieval, tool calls

In Future / Advanced Courses

- Cloud Platforms
  - AWS Bedrock, Azure OpenAI, GCP Vertex AI, Cloudflare Workers
  - Managed APIs & pipelines for scale and enterprise use
- Frameworks
  - LangChain, LlamaIndex → manage RAG flows and agents
  - Pre-built memory, routing, and orchestration modules
- Automation / Integration
  - n8n, Airflow, Zapier → connect LLMs with business processes
  - Automated pipelines: ingest data, trigger alerts, call APIs

What is an LLM?

Definition: Large Language Model (LLM)

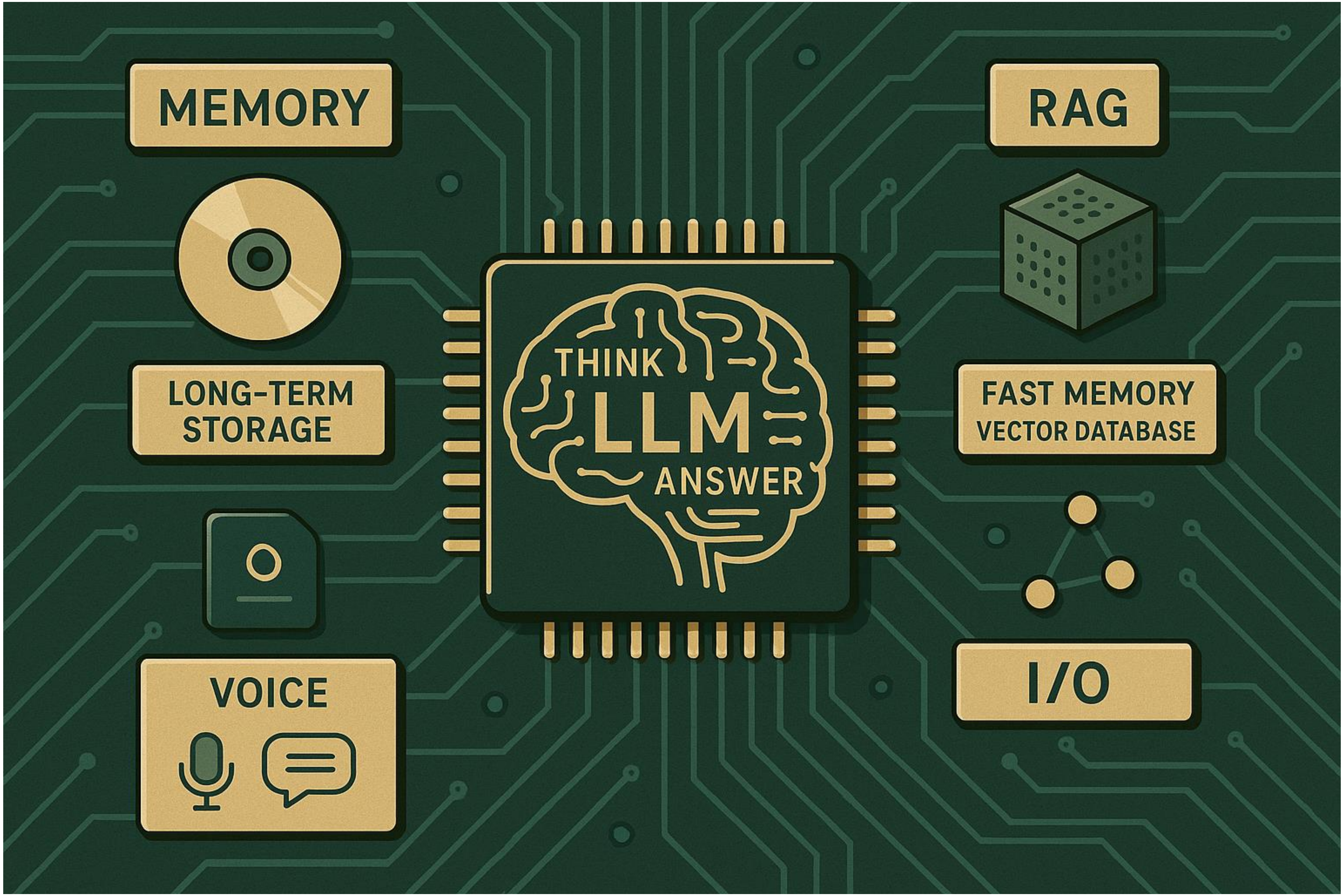
- A type of AI trained on massive text datasets to generate human-like responses
- Examples: GPT-4, Llama 2, BERT

Core Capabilities:

- Text generation (e.g., stories, code)
- Translation, summarization, Q&A
- Pattern recognition from training data

Limitations (Plain LLM):

- Hallucinations: Makes up facts
- No real-time knowledge (cutoff dates)
- Stateless: Forgets previous context without engineering


MEMORY
LONG-TERM STORAGE
VOICE
THINK
LLM
ANSWER
RAG
FAST MEMORY
VECTOR DATABASE
I/O

Lab0 & Lab1

How Does an LLM Work?

High-Level Architecture: Transformer Model

- Input: Tokenized text (words → numbers)
- Processing: Attention mechanisms weigh word relationships
- Self-attention: "What matters in this sentence?”
- Multi-head attention: Parallel focus on different aspects
- Output: Probability distribution over next tokens → Generated text

- Further Reading / Visualization Tool (by Bert Bycroft):: LLM Visualization
  https://bbycroft.net/llm

Lab 2

Enhancing LLMs - Why and How?

- LLMs are powerful but brittle
- Need customization for domain-specific tasks
- Improve accuracy, safety, and efficiency

Key Techniques:

- System Prompt: Guide behavior at the start
  - Example: "You are a helpful coding assistant. Explain concepts simply."
  - Pros: No retraining; quick iteration
- Fine-Tuning: Train on custom data
  - Methods: Full fine-tune (resource-heavy) vs. PEFT (e.g., LoRA for efficiency)
  - Fine-tuning costs compute.
  - Further Reading / Tools: LLaMA Factory: A framework for efficient LLaMA fine-tuning https://github.com/hiyouga/LLaMA-Factory

RAG? (Retrieval-Augmented Generation)

Definition:

- RAG combines LLMs with external knowledge retrieval
- LLM + Search: Fetch relevant docs, then generate response

Why Use RAG?

- Fixes hallucinations: Grounds answers in real data
- Handles up-to-date info (beyond model cutoff)
- Scalable: No need to retrain for new knowledge
- Use Cases: Chatbots with company docs, Q&A over PDF
- When to Use: Any app needing factual accuracy

How Does RAG Work?

1. Index Data: Embed documents (e.g., via Sentence Transformers) into a vector store (e.g., FAISS, Pinecone)
2. Query: User asks question → Embed query
3. Retrieve: Find top-k similar docs via cosine similarity
4. Augment: Stuff retrieved chunks into prompt
5. Generate: LLM responds using context

Challenges:

- Chunking: How to split docs?
- Relevance: Tune embedding model

Lab 3