Full Stack LLMs
A high level overview

Anirban Das, Data Scientist
Applied Research, Capital One

*views are my own
Outline

- Motivation
- Anatomy of the LLM stack using an example app
- Add a round of complication
- Describe building blocks
- Describe challenges
- Parting thoughts
Deployment stack for an LLM based app look complicated but not so much if you think through first principles
Few key things to consider while building production LLM apps 😊

1. The LLMs themselves
2. Data
3. Evaluation framework
4. Serving framework
5. An orchestration layer
6. Auxiliary systems
   - Just bundle most of the things here 😊
Anatomy of the full LLM stack using an application
Of course I took a shortcut and asked GPT4
Example: A simple AI tutoring App

*generated in Microsoft Copilot with prompt: “Create image of an AI tutoring app that uses an LLM inside. Use flat colors. Make it look cool and fun. High quality, high sharpness, muted colors”*
Functional requirements of the AI Tutoring app

1. Input → syllabus / topic / text / pdf
2. Conversational system
3. Student can
   a. Ask clarifying questions or ask to explain
   b. Can ask something outside syllabus
   c. Can ask to critique work
4. Tutor/system can produce pop quizzes
5. Checkpoint student progress
Technical requirements of the AI Tutoring app

1. Multi turn conversations + multiple LLM calls per turn
2. Long + short term memory
3. Knowledge storage + retrieval
4. Guardrails → fact checking, safety etc.
5. Real time inference
6. Agent based system + function calling:
   - explain, summarize, search_web, pop_quiz, checkpoint, code_executor etc.
Developmental stages

- We start with the simplest step, i.e. explain
- We add layers of complications until we reach our final app
- At each additional layer of complication we handle new challenges
- There three other things around the application we will talk at the end
  - Model inference
  - Model guardrails
  - Model training
Simplest form of the app: explain

Basically → just RAG :)}
Challenges and discussion

1. Tech stack: generator / embedding model, vector DB
2. Data:
   a. Input data format
   b. Data cleaning
   c. Chunking
3. How do you improve model performance?
   a. Bigger model?
   b. Instruction (fine)tuning? LoRA / PeFT?
4. Is your model indeed performing better?
5. Prompt Engineering:
   a. Version prompts since evaluation is tied with your versioning of prompts
6. Feedback from users
More complications: making an agent based RAG

Frameworks → autogen, langchain, llamaindex, transformers_agents etc.
What are agents? Why we need them for RAG at all?

**Agents**: System(s) with complex reasoning abilities, that given a task can plan and execute steps in certain order to complete the task using a set of tools.

- Complex workflows → Agents
- Task planner, orchestrator, task executor
- ReAct, adapt etc.
- Function calling

https://medium.com/scisharp/understand-the-llm-agent-orchestration-043ebfaead1f
How would the system work with ReAct?

ReAct

**Question:** What are the 3 Newton’s laws on motion?
**Thought:** I should look at the database too see what I can query.
**Action:** `rag_search`
**Action Input:** `<input to search the vector database using `rag_search`>`: “Newton’s three laws on motion”
**Observation:** Newton’s three laws of motion applied to classical mechanics. They state….
**Thought:** I have the final answer. No further tool usage required.
**Final Answer:** Newton’s three laws of motion applied to classical mechanics. They state….

Challenges and discussion

- Execution control → complicated
- Model selection → complicated
- Orchestrator function → complicated

- At least 2x the number of LLM invocations compared to naive RAG
  - Conditionally calling RAG service? Response caching?
- Multiple LLM calls
  - Inference Optimization → more on this later 😈
- How do you make the LLM aware of all the APIs it can call?
More complications: Adding more capabilities and other tools
More complications: Adding more capabilities and other tools

- Multi-modal capabilities
- Arbitrary function creation and execution
- Abstract away system interactions with APIs
- Task Planner
- LLM
- Memory
- Arbitrary internal or external APIs
- Code Executor
- Arbitrary Tools and APIs e.g. WolframAlpha
- Any Other LLM for audio/video/text/chat/llm/forecasting
- Prompt Engine
- Telemetry, Logs, Feedback gathering
- Concatenated text chunks + prompt
- Similarity Search on user query + prompt
Hand-wavy explanation of how the other features might work!

**Question**: What are the lens equations?

**Thought**: I should look at the database too see what I can query.

**Action**: rag_search

**Action Input**: `<input to search the vector database using rag_search>`: “What are the lens equations?”

**Observation**: No relevant results.

**Thought**: I should use `search_web`

**Action**: `search_web`

**Action Input**: `<input to search the web search_web>`: “What are the lens equations?”

**Observation**: The lens equation expresses the quantitative relationship between the object distance..., https://www.physicsclassroom.com/class/refrn/lesson-5/the-mathematics-of-lenses

**Thought**: I have the final answer. No further tool usage required.

**Final Answer**: According to the website...

---

**Remember/Checkpoint**

**Question**: Checkpoint

**Thought**: I should checkpoint the current progress.

**Action**: checkpoint

**Action Input**: `<input to checkpoint checkpoint>`: current conversation, current topic in progress, function call stack etc.

**Observation**: The current conversation state, and the current tool call stack has been saved to long term memory.

**Thought**: I have the final answer. No further tool usage required.

**Final Answer**: Checkpoint successful

---

**Pop Quiz**

**Question**: Pop Quiz

**Thought**: I should generate a pop quiz using the template on the current topic of study

**Action**: `pop_quiz`

**Action Input**: `<input to pop quiz pop_quiz>`: current conversation, current topic in progress, previous topics completed

**Observation**: Pop quiz generated with topic A, B etc. with display UI and API backend.

**Thought**: I have the final answer. No further tool usage required.

**Final Answer**: Here is your pop quiz, please enter the answers for the respective questions and press submit at the end.

---

**Remember/checkpoint**

**Question**: Checkpoint

**Thought**: I should checkpoint the current progress.

**Action**: checkpoint

**Action Input**: `<input to checkpoint checkpoint>`: current conversation, current topic in progress, function call stack etc.

**Observation**: The current conversation state, and the current tool call stack has been saved to long term memory.

**Thought**: I have the final answer. No further tool usage required.

**Final Answer**: Checkpoint successful

---

this is interesting since `pop_quiz` needs to interact with another API/system to generate the question answer UI and do something on submission
Hand-wavy explanation of how the other features might work!

**critique_picture**

**Question:** Critique the following <insert picture>

**Thought:** I need to understand a picture. I need to call the Gemini API using `critique_picture`

**Action:** critique_picture

**Action Input:** <input to critique_picture>: The picture that was uploaded

**Observation:** Your solution is correct. However the following improvements...

**Thought:** I have the final answer. No further tool usage required.

**Final Answer:** Your solution is correct. However the following improvements...

this is interesting because we hand off the call for example to an external LLM service (could be a local LLM though)

**code_executor**

**Question:** Find the 15th fibonacci number.

**Thought:** I should first generate a python function to find the 15th fibonacci number.

**Action:** generate

**Observation:** The following is a python function to generate the 15th fibonacci number...

```python
def getfib(n):
...
```

**Thought:** I have to execute the function to get the answer

**Action:** code_executor

**Action Input:** <input to code_executor>: `getfib, 15`

**Observation:** 610

**Thought:** I have the final answer. No further tool usage required.

**Final Answer:** 610
Challenges and discussion

- Finetune a model on the apis? or use the context?
  - \texttt{toolformer} or \texttt{toolkengpt} or \texttt{chameleon} or \texttt{gorilla}?
- Worry about context length?
- Multiple types of prompts → inventory+versioning
- Multiple LLMs calls + interaction with external systems → need a structured way of passing data between actors
- Verifying correctness is perhaps not easy and it gets complicated fairly quick
- Need guardrails, alignment, steerability, fault tolerance
More complications: A production app needs **safety**
Challenges and discussion

- Some form of Guardrails is a **must** for customer facing apps
  - Safety
  - Prevention against toxicity
  - Prevention against sensitive data leak
  - Prevention of digression of topic
  - Enforce legal constraints

- Guardrailing both inputs and outputs to prevent prompt injection attacks and jailbreaking attempts

- Adds >1 LLM call(s) if LLM based Guardrails
- Can additionally behave as a fact checking layer

- **NVIDIA Guardrails**, Microsoft Guidance etc.
More Complications: Faster inference
How do you serve these LLMs in practice?

- Serving infra: AWS Sagemaker? Kubernetes?
- Various open source inference frameworks:
  - vLLM, TGI, DJL, TensorRT-LLM, llama.cpp, ExLlama, CTransformers etc.

*https://www.youtube.com/watch?v=TJ5K1CO9Wbs&ab_channel=Anyscale*
Managed serving: E.g. AWS Sagemaker

The Serving Framework
- vLLM
- DJL
- TensorRT LLM
- TGI

The Serving Workflow:
1. Download base model
2. Fine-tune LLM model
3. Register fine-tuned model
4. Deploy using DJL Serving

Important

Managed serving: E.g. AWS Sagemaker
Challenges and discussion

● Perceived latency → important experience metric
● Streaming tokens hides some latency 😈
● Time to 1st token is important
● Some techniques to reduce inference time:
  ○ Prefix Caching
  ○ Continuous Batching → high throughput → 10-20x\(^1\)
  ○ Speculative decoding → low latency → 1.5-3x\(^2\)
● Distributed/Hybrid Inference → 3D parallelism → cost

\[1\] https://www.databricks.com/blog/llm-inference-performance-engineering-best-practices

More complications: You want to train your own model
Training model from scratch is a whole different enterprise!

Training model from scratch is a whole different enterprise!
Training model from scratch is a whole different enterprise!
Challenges and discussion

● Training models is tricky. Some caveats include:
  ○ GPUs are scarce and expensive
  ○ How is your model different from gazillion other models?
  ○ Getting training data is difficult
  ○ Tokenizer makes a lot of difference.
  ○ As you increase the scale of the model, achieving throughput becomes difficult
  ○ All sorts of engineering issues: hard to do MLE + Science simultaneously
  ○ Frequent strange bugs → spend days debugging obscure bugs
  ○ You always want to make a lot of small scale experiments → gather all of your learnings → make fair assumption that when you scale up your observations will still be true → start training larger model
More Complications: Business requirement often complicates things

- We can’t use certain data, but model already trained!
- Changes in the SLA of inference before prod release!
- Different teams own different components
- Changes in answer types
Overall picture of our stack now
Deployment?

RAG Service

- Text Extraction/Document cleanup etc.
- Chunking and Embedding
- Vector Database with MIPS

Similarity Search on user query + prompt
Concatenated text chunks + prompt

Agent Core

- Prompt Engine
- Telemetry, Logs, Feedback gathering

Inference Optimization

Training Models

Orchestrator

- User Query
- Guardrails
- Task Output

Code Executor

Arbitrary Tools and APIs e.g. WolframAlpha

Any Other LLM for audio/video/text/chat llm/forecasting

Task Planner

Memory

Action

36
Deployment?

CICD for RAG service

CICD for prompt engine / feedback telemetry

CICD for guardrail service

CICD for the application

CICD for the serving infra

CICD pipelines maintained by possibly other teams

Offline Training
Some parting thoughts

- LLM apps look complicated but are actually not that different from any other ML system
  - They pose unique challenges, but can be worked through
  - Many capable frameworks under development
- Data quality is paramount to performance
  - Both for fine-tuning and evaluation
  - Data quality scales better than data size
- Domain specific pre and post evaluation is crucial to ensure model quality/performance
- Any non trivial LLM app needs us to focus on
  - Faster inference/scaling
  - Model output safety and regularization
  - Regular model quality check and fixes → Chatgpt Incident
- Training is tricky, unless required, probably improving an open-source model is enough
- Business can add unexpected constraints
Thanks !