LLMOps - Methodologies for Developing with LLMs

Deployment and management of LLMs in production environments.

Version: 0.1 Dated: 2025-01-16 Authored by: CM

colum.mccoole@analect.com
colum-mccoole-746b946a
analect/technical-docs-hierarchy

Topic Navigation

How does this topic relate to other relevant content in this space.

LLMOps is more than an Extension of MLOps

Unlike traditional ML models, which operate on structured data, LLMs handle the vast and often messy world of text and code. This introduces a new layer of complexity, which demands special techniques for data ingestion, pre-processing, and training.

A single prompt fed into an LLM might produce varying responses over time. This can lead to inconsistencies in applications powered by LLMs. LLMOps implements strategies to manage output consistency.

A group of practitioners in this field (Applied LLMs) assembled an invaluable resource mid 2024 that runs through their experiences and 'lessons-learned' from a year shipping LLM applications - What We've Learned From A Year of Building with LLMs, Jun. 2024, with a video verison here. They split it out under Tactical, Operational and Strategy heads. We've tried to summarise the *tactical* part in the grey box on the right. I would encourage you to read the full note.

Other important takeaways include:

- **The model isn't the product, the system around it is** - For teams that aren't building models, the rapid pace of innovation is a boon as they migrate from one SOTA (state-of-the-art) model to the next, chasing gains in context size, reasoning capability, and price-to-value to build better and better products.
- **calibrate risk tolerance based on the use-case** - for less critical applications, such as a recommender system, or internal-facing applications like content classification or summarization, excessively strict requirements only slow progress without adding much value.
- **Focus on model evaluation** - LLMOps is about production monitoring and continual improvement, linked by evaluation. The better your evals, the faster you can iterate on experiments, and thus the faster you can converge on the best version of your system.
- **Empower everyone to use new AI technology** - while it may seem expensive to have a team spend a few days hacking on speculative projects, the outcomes may surprise you.

Risks Inherent in LLMs That Need Management

Businesses are mainly focusing on internally-facing LLM-powered applications until they get better at mitigating some of the risks below. Model-explainability remains somewhat under-researched and will be a necessary prerequisite to companies and consumers having trust in these systems.

1. **Bias and Fairness:** LLMs learn from massive text datasets, and unfortunately, these datasets often reflect human biases. This can lead to LLMs generating outputs that perpetuate harmful stereotypes, discrimination, or social inequalities.
2. **Toxicity:** LLMs might produce text that is offensive, hateful, or dangerous.
3. **Hallucinations:** It's not uncommon for LLMs to generate factually incorrect or nonsensical information, creating illusions of knowledge.
4. **Privacy Violations:** Training datasets for LLMs can include private or personal information. If not carefully handled, LLMs might leak or reproduce this sensitive data and compromise individual privacy.
5. **Prompt Injections:** LLMs are vulnerable to prompt injection attacks where malicious inputs can manipulate the model's behavior, leading to unintended or harmful outputs.
6. **Data Leakage:** LLMs may inadvertently reveal sensitive or proprietary information included in the training data, leading to potential breaches of privacy and confidentiality.

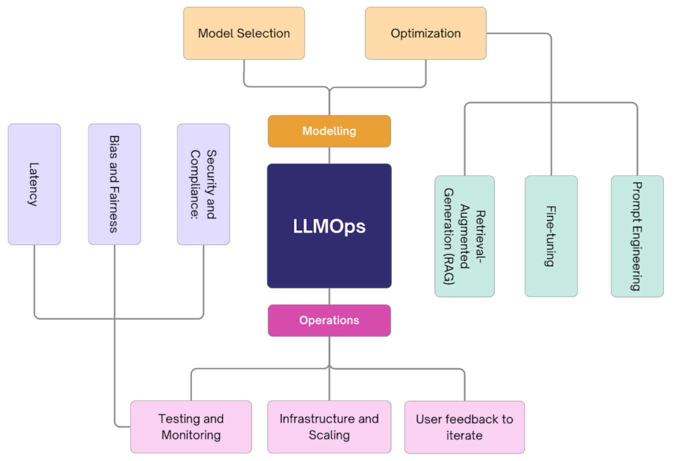

Figure 1. Steps to bring LLMs into production using LLMOps from LLMOps: MLOps for Large Language Models (Giskard)

Tactical Lessons Learned Working with LLMs (applied-llms.org/)

1. **Prompt Engineering** - the right prompting techniques, when used correctly, can get us very far. It's overestimated because even prompt-based applications require significant engineering around the prompt to work well.
   a. The idea of *in-context learning via n-shot prompts* is to provide the LLM with examples that demonstrate the task and align outputs to our expectations.
   b. *Chain-of-Thought (CoT) prompting*, we encourage the LLM to explain its thought process before returning the final answer.
   c. Often accomplished via *Retrieval Augmented Generation (RAG)*, providing the model with snippets of text that it can directly utilize in its response is an essential technique.
   d. Have small prompts that do one thing, and only one thing, well. Split a single prompt into multiple prompts that are simple, focused, & easy to understand.
2. **RAG / Information Retrieval** - providing knowledge as part of the prompt. This grounds the LLM on the provided context which is then used for in-context learning. This is known as *retrieval-augmented generation (RAG)*.
   a. relevance of documents presented as context should maximise metrics such as *Mean Reciprocal Rank (MRR)* and *Normalised Discounted Cumulative Gain (NDCG)*. Also consider information density (favouring the more concise) of two similarly returned documents.
   b. keyword search (or combined in hyrbid search) is still a very good baseline. Algorithms like BM25 have been battle-tested on info retrieval for decades.
   c. give precedence to RAG over fine-tuning for new knowledge since it's cheaper to implement and academic papers suggest the RAG approach has better results.
   d. long-context models are unlikely to make RAG obselete. The inference-time cost of feeding a whole documentation site may not always make sense, in terms of quality of response.
3. **Fine-tuning / optimizing workflows** - consider how to split a single complex task into multiple simpler tasks. When is finetuning or caching helpful with increasing performance and reducing latency/cost?
   a. Small tasks with clear objectives make for the best agent or flow prompts. Use structured outputs for systems orchestrating multiple agents working on a multi-level task.
   b. Have agent systems produce deterministic plans which are then executed in a structured, reproducible way. Otherwise, the likelihood that an agent completes a multi-step task successfully decreases exponentially as no. steps increases.
   c. Caching saves cost and eliminates generation latency by removing the need to recompute responses for the same input.
   d. Consider fine-tuning when all prompt-engineering approaches are exhausted. It will be more costly. One has to annotate fine-tuning data, fine-tune and evaluate models, and eventually self-host them. It may be possible to generate synthetic data and fine-tune on that.
4. **Evaluation & Monitoring** - evaluating LLMs is challenging given the open-ended non-deterministic nature of these models.
   a. Create some assertion-based unit tests per Your AI Product Needs Evals, Hamel Husain, March 2024. These should be triggered by any changes to the pipeline, whether by editing prompt, adding context via RAG, or other modifications.
   b. LLM-as-Judge (where one LLM judges output from another), when implemented well, can achieve decent correlation with human judgments.
   c. A key challenge when working with LLMs is that they'll often generate output even when they shouldn't. Use robust guardrails that detect and filter/regenerate undesired output.
   d. Techniques like CoT (chain-of-thought) help reduce hallucination by getting the LLM to explain its reasoning before finally returning the output.