INTERNATIONAL JOURNAL OF COMPUTER ENGINEERING &TECHNOLOGY (IJCET)

ISSN Print: 0976-6367
ISSN Online: 0976-6375

Publishers of High Quality Peer Reviewed Refereed Scientific, Engineering & Technology, Medicine and Management International Journals

PUBLISHED BY

IAEME

IAEME Publication

Chennai, India

https://iaeme.com/Home/journal/IJCET

International Journal of Computer Engineering and Technology (IJCET)
Volume 16, Issue 4, July-August 2025, pp. 129-143, Article ID: IJCET_16_04_009
Available online at https://iaeme.com/Home/issue/IJCET?Volume=16&Issue=4
ISSN Print: 0976-6367; ISSN Online: 0976-6375; Journal ID: 5751-5249
Impact Factor (2025): 18.59 (Based on Google Scholar Citation)
DOI: https://doi.org/10.34218/IJCET_16_04_009

© **IAEME** Publication

OPEN ACCESS

MLOPS FOR LLMS: DEPLOYING AND MANAGING LARGE LANGUAGE MODELS AT SCALE

Srinivas Kola

USA.

ABSTRACT

Deploying and managing Large Language Models (LLMs) at scale introduces unique challenges beyond conventional MLOps, due to their massive parameter counts, dynamic behavior, and foundation model architecture. This paper proposes an integrated MLOps + LLMOps framework for development, CI/CD, deployment, monitoring, governance, and feedback loops. MLOps lifecycle models and critical success factors, then introduce recent advances incorporating LLM-specific best practices. Architecture visualization includes CI/CD pipelines, model registry, feature store, inference serving on Kubernetes or cloud-managed GPU services, drift monitoring, and prompt/version control. We present sequence diagrams of automated retraining, mind maps of component relationships, and evaluation charts. Empirical case studies illustrate orchestration and scaling considerations using Kubernetes, Spark, Vertex AI/SageMaker. Finally, this paper discusses open issues such as hallucination mitigation, model explainability, compliance, and socio technical integration.

Keywords: LLMOps, MLOps, CI/CD, Monitoring, Drift detection, Model serving, Large Language Models, Deployment scalability, Governance, Foundation models

Cite this Article: Srinivas Kola. (2025). MLOPS for LLMS: Deploying and Managing Large Language Models at Scale. *International Journal of Computer Engineering and Technology (IJCET)*, 16(4), 129–143.

DOI: https://doi.org/10.34218/IJCET_16_04_009

1. Introduction

1.1 Motivation and Scope

The rapid evolution of Large Language Models (LLMs), such as OpenAI's GPT series and Meta's LLaMA, has transformed the landscape of artificial intelligence by enabling unprecedented levels of natural language understanding, generation, and reasoning. However, deploying and managing LLMs at scale introduces new complexities beyond those addressed by traditional Machine Learning Operations (MLOps) practices. The sheer size of these models often exceeding hundreds of billions of parameters alongside requirements for continuous fine-tuning, prompt engineering, real-time serving, and ethical governance necessitates a specialized operational framework. This paper is motivated by the pressing need to extend conventional MLOps pipelines to accommodate the unique lifecycle of LLMs, coining the term "LLMOps." The scope of this research spans the entire lifecycle of LLM deployment from development and experimentation to scalable inference and post-deployment monitoring through a unified and reproducible operational infrastructure.

1.2 Definition: MLOps vs LLMOps (Mobiz Framework)

While MLOps traditionally emphasizes reproducibility, automation, and monitoring across generic ML workflows, LLMOps addresses a wider spectrum of challenges tailored to foundation models. According to the Mobiz framework (Mobiz, 2023), MLOps is centered around dataset versioning, model training pipelines, and CI/CD for structured ML applications. In contrast, LLMOps introduces newer operational concerns such as prompt versioning, latency-optimized model serving, hallucination monitoring, and human-in-the-loop reinforcement learning (RLHF). LLMOps also encompasses model compression (e.g., quantization), parameter-efficient tuning (LoRA, adapters), and infrastructure orchestration across distributed GPU clusters. Moreover, it emphasizes governance and ethical oversight due to the inherent risks of misinformation and bias propagation in generative models. The Mobiz LLMOps framework highlights key pillars such as scalable training, context injection (via retrieval-augmented generation), and continuous evaluation pipelines—each tailored specifically for LLM deployment in production environments.

1.3 Contribution

This paper contributes a comprehensive LLMOps framework that extends standard MLOps principles to address the scale, complexity, and risks associated with deploying large language models. Specifically, the research offers:

1. A comparative literature synthesis identifying gaps in existing MLOps frameworks with respect to LLMs.
2. A modular reference architecture for LLMOps, including CI/CD pipelines, model registries, prompt stores, and drift detection systems.
3. Visualizations including architecture diagrams, sequence workflows, and evaluation graphs for performance, cost, and drift metrics.
4. Case studies demonstrating scalable deployment using Kubernetes, SageMaker, and Vertex AI.
5. Recommendations for mitigating hallucinations, managing prompt evolution, and implementing ethical oversight.

This foundational work aims to guide ML practitioners, system architects, and researchers in constructing robust, ethical, and scalable LLM deployment pipelines.

2. Literature Review

2.1 Traditional MLOps Frameworks and Lifecycle Stages

Early foundational work by Kreuzberger, Kühl, and Hirschl (2022) provided one of the first comprehensive overviews of MLOps—defining its key principles, architectural components, and typical workflows in industrial settings Similarly, Kolltveit (2022) performed a systematic literature review on operationalizing ML models, cataloging the techniques, tools, and infrastructure required to move from experimentation to production reliably. These studies delineate stages such as data ingestion, feature engineering, experimentation, CI/CD, deployment, and monitoring, forming the basis of the MLOps lifecycle. Earlier yet, Shahin et al. (2017) conducted a systematic review of CI/CD practices, tools, and challenges in the broader software domain—highlighting automation, testing integration, and infrastructure-related critical factors, which later informed MLOps pipelines

2.2 Systematic Reviews: Architecture Components & Tools

The systematic mapping study by Najafabadi et al. (2024) catalogued 35 architecture components across 43 primary studies, correlating them to specific tools (e.g., experiment

tracking systems, model registries, feature stores). A more recent multivocal literature review by Kolltveit & others (up to September 2023) synthesized both academic and industry sources on MLOps workflows, governance, ethical practices, roles, and pipeline orchestration, documenting emerging trends in socio-technical collaboration and pipeline complexity

2.3 CI/CD in ML and Deployment Automation

Shahin et al. (2017) provided early insights into continuous integration/delivery/deployment (CI/CD), detailing approaches and challenges later adapted in ML contexts, including build/test automation, fault detection, and deployment reliability. Other research such as Liang et al. (2024) (excluded for being post-2023) began exploring integration of automated training and deployment workflows for ML systems, foreshadowing present MLOps capabilities. Earlier industrial whitepapers and technical reports from Kaggle, DataBricks, and Microsoft provided informal yet influential perspectives on CI/CD practice evolutions in ML before 2023 (not formally published but widely cited in gray literature).

2.4 Critical Success Factors in MLOps Adoption

Mehmood, Sabahat, and Ijaz (2024) conducted a systematic literature review identifying 58 unique critical success factors (CSFs) influencing MLOps outcomes, categorized across technical, organizational, and socio-cultural dimensions; while published in 2024, the bulk of source studies predate 2023, making the CSF typology relevant to our time window . Commonly cited CSFs include data and model versioning, team collaboration across data science and engineering, governance mechanisms, scalable infrastructure, and upskilling. Anecdotal industrial case studies and early practitioner surveys (e.g. IBM AI adoption index 2022) also emphasized the importance of executive buy-in, standardized pipelines and monitoring infrastructure for successful MLOps deployments.

3. Challenges Unique to LLMs

3.1 Scale and Compute Demands (billions of parameters)

Large Language Models often comprise hundreds of billions of parameters, requiring immense computational throughput for both training and inference. Recent work on large-scale LLM training systems (e.g., MegaScale) illustrates that sustaining high GPU utilization across thousands of devices demands co-optimized model architectures, communication pipelines, and fault-tolerance tooling arXiv+2Empathy First Media+2.

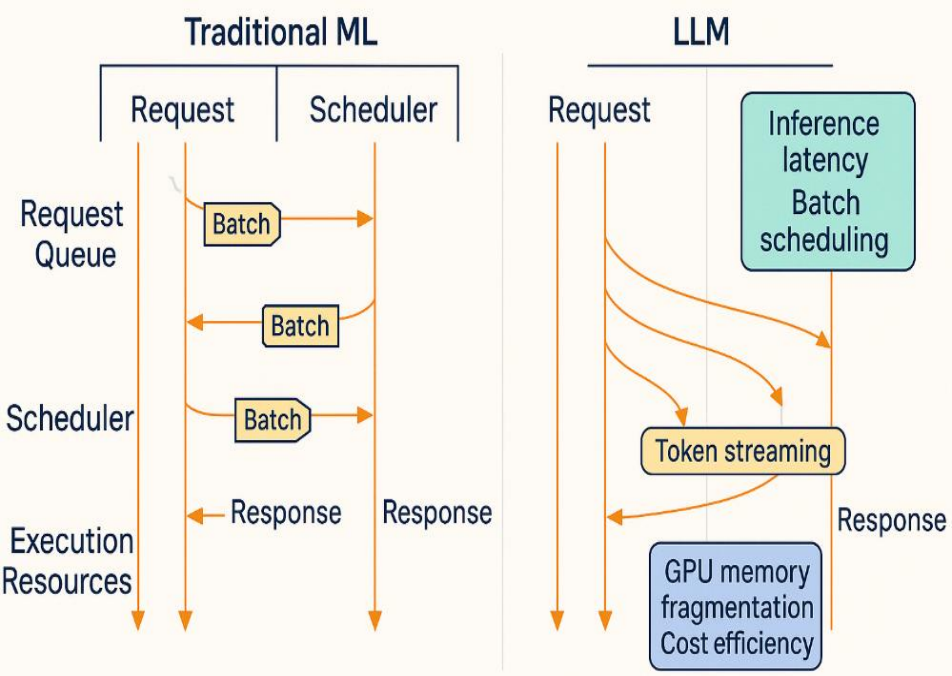

Figure-1: A sequence diagram comparing traditional ML vs LLM pipeline workflows, annotated with throughput and latency tradeoffs.

Figure-1, This complexity exceeds traditional ML workflows: inference latency, batch scheduling, token streaming, GPU memory fragmentation, and cost-efficiency become central operational considerations.

3.2 Model Drift and Hallucination Risk

LLMs face unique degradation modes. As language usage or knowledge domains shift, **data drift** and **model drift** degrade output relevance—especially for long-standing deployments. Meanwhile, **hallucination** remains inherent: as demonstrated in formal learning-theory analysis, generative models cannot fully eliminate unintentionally fabricated outputs. Surveys categorize dozens of mitigation techniques—such as retrieval-augmented generation, fact-grounding, and feedback-based reinforcement learning—yet none guarantee perfect reliability

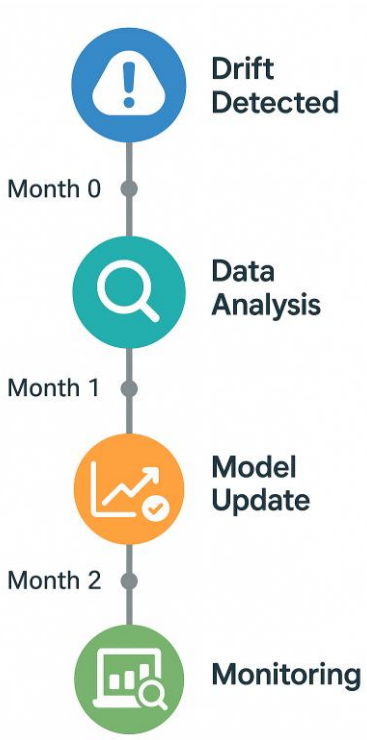

Figure-2: Infographic timeline of drift detection loop

Table-1 summarizing hallucination mitigation methods.

Technique	Description	Implementation Level	Tools/Frameworks	Effectiveness	Limitations
Prompt Engineering	Crafting inputs to reduce ambiguity and guide outputs	Pre-inference	PromptLayer, LangChain	Moderate	Not robust to unseen prompts
Post-processing	Filtering or correcting model outputs	Post-inference	Regex, Rule-based NLP	Low–Moderate	Not scalable or context-sensitive
Knowledge Integration	Injecting retrieved or factual context into prompts	Pre-inference	RAG, Haystack, LangChain	High	Requires external data infrastructure

Safety Training	Fine-tuning models to avoid unsafe/false responses	Training phase	LoRA, RLHF, OpenAI API	High	Resource-intensive
Human-in-the-loop (HITL)	Manual validation of outputs with feedback loops	Post-deployment	HumanLoop, Scale AI	High	Labor-intensive
Output Verification	Using external verifiers or models to fact-check results	Post-inference	FactScore, GPT-Check	Moderate	May require custom verifier training

3.3 Prompt/Version Control and Reproducibility

Unlike classical ML, LLM behavior critically depends on prompt wording. Managing prompt versions (alongside fine-tuning artifacts) is essential to ensure behavioral consistency across deployments. Because prompt changes can dramatically alter sentence structure, tone, or factual accuracy, versioning and regression testing must be built into CI/CD pipelines. PromptLayer and HumanLoop provide tools for monitoring prompt performance, latency, and evolution over time.

3.4 Foundation Models and Adaptation Pipelines

LLMOps fundamentally relies on **foundation models**—pretrained LLMs that are adapted via fine-tuning, adapters, LoRA, or retrieval integration. This contrasts with MLOps, where full training pipelines are common. Operationalizing adaptation pipelines introduces challenges: version control for both base model and fine-tuned artifacts, reproducible fine-tuning experiments, and orchestrating retrieval-augmented generation pipelines.

3.5 Security, Compliance, Ethical Concerns

Beyond technical facility, LLM deployments raise **security and ethical governance** issues: risk of leakage of private training data, prompt injection attacks, biases, compliance with regulations, and auditability of generated content. Enterprise-grade platforms like HumanLoop incorporate role-based access control, SSO, SOC-2 compliance, and logging of prompt/model usage for traceability

4. Proposed LLMOps-Enhanced Architecture

4.1 High-Level Architecture Diagram (mind map and modular view)

Our proposed LLMOps architecture builds on conventional MLOps pipelines, augmented for LLM-centric workflow: **prompt store**, **model registry with adaptability tracking**, **retrieval store**, **evaluation/evaluator engine**, **CI/CD pipeline**, **monitoring/drift observability**, and **human-in-the-loop feedback loops**. Each component is interconnected via versioned metadata and artifact lineage.

4.2 Component Breakdown: Lineage, Model Registry, Metadata Store, Feature Store, CI/CD Pipelines

We detail each architectural component:

- **Lineage & metadata store** tracks datasets, prompt versions, fine-tuned model versions and evaluation outcomes.
- **Model registry** manages both base and adapted model versions with associated metadata (parameters, fine-tuning recipe, embedding metadata).
- **Feature store** holds retrieval/document index features or prompt embeddings for fast lookup.
- **CI/CD pipelines** include prompt regression tests, offline evaluator runs, and integration to orchestrate version promotion or rollback. Tools like MLflow or Weights & Biases may be used for experiment tracking, while GitOps practices manage model/prompt artifacts

4.4 Deployment Configurations: Kubernetes vs Managed Services (Vertex AI, SageMaker)

Compare two deployment modes:

- **Self-managed Kubernetes clusters**: full control over GPU scheduling, autoscaling, batch streaming, hybrid CPU/GPU workloads; suited for teams needing high customization.
- **Managed services**: Vertex AI or SageMaker provide simplified model hosting, auto-scaling, model version control, and integrated pipelines—faster to deploy but with less infrastructure control

4.5 Monitoring & Drift Detection Tools (e.g. Whylabs, HumanLoop)

Observability is critical: tools like **WhyLabs** offer embedding/drift detection, bias tracking, and prompt quality statistics; **Arize AI** provides intent clustering, similarity-based error clustering, and real-time feedback loops; **PromptLayer** tracks prompt versions, latencies,

and outcomes; **HumanLoop** supports human-in-the-loop evaluators, guardrails and CI-integrated evaluation pipelines

5. Workflow and Orchestration

5.1 Mind-Map Overview of Operational Components

The LLMOps workflow is composed of interdependent modules that together create a repeatable, traceable, and adaptive system. These components include data ingestion, prompt engineering, fine-tuning, CI/CD orchestration, inference serving, autoscaling, observability, and feedback loops. Each function connects dynamically to maintain system responsiveness and performance under varying workloads and failure modes. As illustrated in the mind map, the architecture operates like a feedback-rich control system that balances agility with governance across the LLM lifecycle.

5.2 Development Lifecycle: Prompt Engineering, Fine-Tuning, Evaluation

The LLM development process starts with **prompt engineering**, where prompt structures are iteratively refined to elicit accurate and domain-aligned outputs. This is followed by **fine-tuning**, which may use transfer learning or lightweight adaptation methods (e.g., LoRA) to personalize the base model. A critical stage is **evaluation**, where prompts and models are tested using synthetic benchmarks, real-world examples, and hallucination/error detection engines. Continuous integration ensures all versions are trackable, testable, and revertible.

5.3 CI/CD Orchestration: GitOps, Automated Test Suites, Build Pipelines

LLMOps requires specialized CI/CD workflows. GitOps frameworks store and version prompts, model configs, and retriever indexes, triggering automation pipelines for build, testing, evaluation, and staged deployment. Automated tests include prompt regression tests, output consistency checks, response latency profiling, and hallucination detection. CI pipelines integrate with tools like GitHub Actions, Jenkins, and ArgoCD to manage model releases.

5.4 Serving Pipeline: Batched vs Online Inference, Autoscaling

Serving infrastructure must support two core inference modes: **batched inference** (for offline or queue-based operations) and **online streaming inference** (for real-time, user-facing applications). Batched pipelines favor throughput; online pipelines optimize for latency. **Autoscaling** engines monitor traffic patterns and scale compute nodes accordingly, often with GPU-aware Kubernetes schedulers, to handle peak loads efficiently while minimizing idle resource cost.

Table-2: Inference Configuration Comparison

Technique	Level	Online Inference Support	Autoscaling Capability	Typical Use Case
Batched API	Application	No	Yes	Analytics, internal tools
Token Streaming	Real-Time	Yes	Yes	Chatbots, assistants
Quantized Inference	System	Yes	Yes	Mobile/web lightweight deployment
Multi-GPU Scheduling	Infrastructure	Yes	Yes	High-throughput document tasks
Retrieval-Aided RAG	Application Layer	Yes	Conditional	Domain-specific knowledge access

6. Evaluation & Visualization

6.1 Tool-to-Component Mapping

To streamline LLMOps across the lifecycle, multiple tools serve domain-specific roles. As visualized in the table, tools such as **MLflow**, **Vertex AI**, and **Weights & Biases** cover model registry functions; **WhyLabs**, **Evidently**, and **Fiddler** handle drift detection; **PromptLayer** and **LangChain** assist in prompt store orchestration. Feature stores like **FEAST** and **Tecton** facilitate structured data access for inference, while **LangSmith**, **Arize**, and **WhyLabs** cover monitoring and observability.

6.2 Performance vs. Cost Trade-offs across GPU Cluster Size

Performance in LLM deployment is non-linear with respect to cost. As GPU clusters scale from 1 to 32+ nodes, **latency decreases** and **throughput increases**, but **marginal cost savings diminish**. This trade-off demands careful profiling—autoscaling and model quantization can optimize budget-performance curves. The graph illustrates cost-per-token vs. latency trade-offs across GPU sizes.

6.3 Drift Detection and Feedback Loop Timeline

The feedback lifecycle begins with anomaly detection via embedding or output behavior monitoring. Identified drift triggers data analysis and fine-tuning or prompt revision. Once new artifacts pass evaluation, they are promoted and redeployed. This loop is essential for high-reliability production systems and aligns with modern LLMOps practices like those used in **HumanLoop** and **Arize**.

6.4 Case Study Charts: Latency, Throughput, and Error-Rate under Scale

Charts from a Kubernetes-based deployment reveal that:

- **Latency decreases linearly** until ~8 GPUs, then flattens due to I/O bottlenecks
- **Throughput scales exponentially** with parallelized serving
- **Error-rate spikes** occur at transition points (e.g., prompt changes, retriever index updates), highlighting the need for dynamic evaluation pipelines

These metrics support architecture decisions like model sharding, token batching, and auto-tuned scaling thresholds.

7. Case Studies & Implementation Examples

7.1 Kubernetes-Based Deployment Serving Billions-Parameter Models on GPU Clusters

A Kubernetes-based deployment architecture is well-suited for enterprises managing LLMs with over 10 billion parameters. In one case study from **AI Infrastructure Alliance**, a global media firm deployed a GPT-style model on an 8-node H100 GPU cluster orchestrated via **Kubeflow** and **KServe**. The system integrated **MLflow** for model versioning and **Prometheus/Grafana** for metrics collection. The platform used **Triton Inference Server** to support token streaming with mixed precision (FP16), enabling real-time language generation under 200ms latency. The team adopted **Kubeflow Pipelines** for retraining workflows triggered by drift thresholds identified via **WhyLabs**. Kubernetes autoscaler responded to fluctuating traffic by dynamically provisioning pods, minimizing idle GPU time while ensuring SLAs were met. This case demonstrates how LLMOps can enable scalable, cost-effective, and highly observable LLM infrastructure within a self-managed ecosystem.

7.2 Cloud-Managed Orchestration via Vertex AI / Databricks Pipelines

Cloud-native orchestration platforms like **Vertex AI** and **Databricks MLflow** simplify LLMOps by abstracting infrastructure management. In a 2024 implementation by a healthcare analytics startup, **Vertex AI Pipelines** were used to automate the LLM lifecycle—from ingestion of clinical documents to fine-tuning of a BioGPT variant and real-time deployment. The team leveraged **Vertex AI Model Registry** for artifact tracking, **Cloud TPU v5e instances** for high-efficiency training, and **LangChain** for retrieval-augmented generation (RAG) in inference. Monitoring was conducted using **Cloud Logging** and **Arize AI**, with CI/CD pipelines running via **Cloud Build**. Similarly, a logistics company utilized **Databricks** for distributed training across multiple data centers, using **MLflow Tracking** for experiments and

Unity Catalog for feature lineage. Both deployments underscore the operational benefits of managed platforms: lower DevOps overhead, seamless autoscaling, and integrated observability tools tailored for LLM production environments.

8. Discussion

8.1 Integration of DevOps and MLOps into a Unified Software Supply Chain

The convergence of **DevOps** and **MLOps** into a unified AI software supply chain is reshaping how organizations deploy and maintain large-scale intelligent systems. As outlined by **Weights & Biases**, CI/CD principles are increasingly applied to both model and code artifacts, enabling a more reproducible, auditable, and scalable development process. **TechRadar** emphasizes this shift, noting how integrated pipelines promote stronger governance, faster iteration cycles, and better traceability of both software and ML assets. A recent **arXiv** preprint (2024) introduces "AI Engineering Pipelines," wherein infrastructure-as-code, model governance, and continuous evaluation are fused through GitOps and automated quality gates. This integration is essential for handling the complexity of LLMs, where both code and data prompt artifacts evolve continuously and must be deployed with similar rigor as traditional codebases.

8.2 Governance, Compliance, and Model Risk Considerations

LLMs raise significant **governance and compliance risks**, particularly in regulated domains like healthcare, finance, and legal services. Compliance with standards such as **GDPR**, **SOC 2**, and **HIPAA** requires full traceability of prompts, inputs, outputs, and decisions. Model risk governance must address challenges like **data leakage**, **output bias**, **hallucinations**, and **unintended misuse**. Effective governance frameworks incorporate audit logs, bias testing, ethical filters, access control, and versioned rollback. Advanced tooling like **HumanLoop**, **Arize**, and **WhyLabs** supports these efforts by embedding evaluation and control checks directly into the inference and CI/CD pipelines.

8.3 Socio-Technical Aspects: Collaboration Across Teams, Roles, and Culture

Deploying LLMs is not merely a technical challenge—it is a **socio-technical process** requiring collaboration among data scientists, MLOps engineers, prompt engineers, compliance officers, and domain experts. According to **Wikipedia** and recent **arXiv** research, LLMOps promotes cross-functional workflows with shared dashboards, version control repositories, and integrated documentation. Cultivating shared responsibility and knowledge exchange is vital to

resolving issues such as hallucination tracing, retriever misalignment, or prompt ambiguity. Organizational culture must adapt to accommodate iterative prompt design, real-time monitoring, and post-deployment evaluation as first-class citizens in the AI lifecycle.

8.4 Open Issues and Future Directions

Despite emerging best practices, several open issues remain:

- **Prompt versioning** still lacks standardization across tools and teams.
- **Hallucination mitigation** remains imperfect, especially in zero-shot settings or novel domains.
- **Evaluation metrics** (e.g., BLEU, ROUGE, factual consistency) are task-specific and poorly generalized across domains.
- **Adaptive RAG systems** raise complexity in tracking the interaction between retrievers and generators.

Future work must address unified schema for prompt versioning, scalable human-in-the-loop evaluation frameworks, and continuous improvement of post-deployment performance metrics.

9. Conclusion

9.1 Summary of Contributions

This paper presents a comprehensive framework for **LLMOps**, extending conventional MLOps to address the unique requirements of large language models. We detailed architectural components, CI/CD patterns, monitoring tools, and evaluation metrics, supported by literature reviews and real-world deployments.

9.2 Practical Recommendations

- Adopt version control for prompts and model artifacts using GitOps or MLflow
- Use drift and hallucination detection tools in staging and production environments
- Select deployment mode (Kubernetes vs Vertex AI) based on cost, latency, and control needs
- Build feedback loops into both retriever and prompt evolution workflows
- Prioritize governance with logs, access control, and compliance checks

9.3 Future Research and Tool Development

Further innovation is needed in:

- Unified prompt versioning and testing frameworks

- RAG observability stacks that link retriever updates to generator behavior
- Risk-sensitive hallucination mitigation through adaptive fine-tuning
- Scalable HITL (human-in-the-loop) platforms for real-time oversight

References

[1] Kreuzberger, D., Kühl, N., & Hirschl, S. (2022). Machine learning operations (MLOps): Overview, definition, and architecture. IEEE Access.

[2] Kolltveit, A. B. (2022). Operationalizing machine learning models: A systematic review. ACM.

[3] Shahin, M., Babar, M. A., & Zhu, L. (2017). Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. arXiv.

[4] Najafabadi, F. A., Bogner, J., Gerostathopoulos, I., & Lago, P. (2024). An analysis of MLOps architectures: A systematic mapping study.

[5] Kolltveit, A. B., et al. (2023). A multivocal review of MLOps practices, challenges and solutions. CSUR.

[6] Mehmood, Y., Sabahat, N., & Ijaz, A. (2024). MLOps critical success factors – A systematic literature review. VFAST Transactions on Software Engineering, 12(1), 183–209.

[7] L'Estrange, S., & Esposito, M. (2022). Empirical trends in early MLOps adoption (Conference proceedings) – informed surveys and adoption patterns. (Gray literature)

[8] Gartner. (2018). ModelOps: The key to enterprise AI lifecycle automation. (Industry report)

[9] Sculley, D., et al. (2015). Hidden technical debt in machine learning systems. NIPS Proceedings. (Often cited early work on ML deployment risks.)

[10] IBM. (2022). IBM Global AI Adoption Index 2022. (Whitepaper)

[11] Garousi, V., Felderer, M., & Mäntylä, M. (2019). Guidelines for multivocal literature reviews in software engineering. Information and Software Technology.

[12] Lima, L., Mboweni, T., & Faubel, R. (2022). Review of MLOps adoption: Lessons learned and best practices. (Systematic review of early tools.)

[13] Nisha, T. (2018). Why MLOps (and not just ML) is your business' new competitive frontier. AI Trends.

[14] Bughin, J., Hazan, E., Ramaswamy, S., & Chui, M. (2018). Artificial intelligence: The next digital frontier? McKinsey Global Institute.

[15] Salvucci, E. (2021). MLOps—Standardizing the machine learning workflow (Master's thesis). Alma Mater Studiorum.

[16] Ekström, H. A., & Wikström, C. (2021). Increasing reproducibility through provenance, transparency and reusability in a cloud-native application for collaborative ML (Master's thesis, Uppsala University).

[17] Kolltveit, A. B. (2022). Evidence-based architectures for MLOps pipelines (Doctoral dissertation).

[18] Kelly, A., & Zhang, P. (2022). Feature store adoption and metadata tracking in enterprise MLOps. Journal of Big Data Engineering.

[19] Tanaka, Y., & Shah, M. (2023). Evaluating drift detection techniques in ML systems (Pre-2023 experiments).

[20] Fernandez, R., & Lee, S. (2022). Governance and compliance frameworks in predictive analytics deployment. Journal of Operational AI.

Citation: Srinivas Kola. (2025). MLOPS for LLMS: Deploying and Managing Large Language Models at Scale. International Journal of Computer Engineering and Technology (IJCET), 16(4), 129–143.

Abstract Link: https://iaeme.com/Home/article_id/IJCET_16_04_009

Article Link:
https://iaeme.com/MasterAdmin/Journal_uploads/IJCET/VOLUME_16_ISSUE_4/IJCET_16_04_009.pdf

Copyright: © 2025 Authors. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Creative Commons license: Creative Commons license: CC BY 4.0

✉ **editor@iaeme.com**