International Journal of Computer Trends and Technology
ISSN: 2231–2803 / https://doi.org/10.14445/22312803/IJCTT-V74I10P104

Volume 74 Issue 10, 36-41, October 2026
© 2026 Seventh Sense Research Group®

Review Article

LLM-Ops vs. DevOps: A Comparative Review of Infrastructure Requirements for Generative AI

Lohith Reddy Kalluru

Independent Researcher / Cloud Developer III, United States.

[1]*Corresponding Author : lohithreddykalluru@gmail.com*

Received: 19 August 2026 Revised: 22 September 2026 Accepted: 10 October 2026 Published: 29 October 2026

Abstract *- Generative AI requires infrastructure considerations beyond the traditional DevOps practices. DevOps evolved around deterministic software systems whose behavior is explained primarily by source code, build pipelines, container images, infrastructure-as-code, and runtime service monitoring. However, these controls cannot independently reproduce or evaluate generative-AI behavior that also depends on model versions, prompts, retrieved context, inference settings, and tool interactions. Large Language Model Operations (LLM-Ops) must therefore manage probabilistic model behavior, large model checkpoints, prompt artifacts, vector databases, retrieval pipelines, semantic evaluation, graphics-processing-unit-accelerated inference, and agentic tool execution. Existing comparisons commonly discuss model deployment and monitoring but provide limited infrastructure-level synthesis across these operational requirements. This review examines which infrastructure requirements distinguish LLM-Ops from conventional DevOps and which additional controls are needed when generative-AI systems retrieve external knowledge or execute tools. The two practices are compared across artifact management, compute, networking, storage, deployment, monitoring, evaluation, security, sustainability, and operational governance. The review also identifies agentic workflows and tool-oriented execution as an emerging infrastructure layer involving command-line tools, application programming interfaces, vector databases, external services, and interfaces such as the Model Context Protocol. The findings indicate that LLM-Ops does not replace DevOps. Instead, it extends established DevOps practices through model-aware infrastructure, continuous semantic evaluation, retrieval-aware data management, governed tool execution, and energy-aware deployment strategies.*

Keywords *- Devops, Generative Ai, LLM-OPS, Rag, Gpu Infrastructure.*

1. Introduction

DevOps refers to the now mainstream operational approach to building, testing, deploying, and monitoring software systems. Common DevOps practices include Continuous Integration, Continuous Delivery (CI/CD), automated testing, infrastructure-as-code, containerization, observability, and feedback-driven release cycles. CI/CD systems automate software delivery workflows from build to test to deployment stages. Infrastructure-as-code enables environments to be versioned as configuration files [1-4].

Generative AI systems pose a distinct set of operational problems. Where the behavior of conventional software systems is largely deterministic and encoded in source code, the behavior of large language model applications are based on model weights, prompts, embeddings, retrieval pipelines, context windows, fine-tuning datasets, external tool responses, and model-serving infrastructure.

Model responses are probabilistic and sensitive to application context. A slight modification to a prompt, retrieval configuration, deployed model version, or called out tool response may alter the result produced by the system.

LLM-OPS covers operational requirements beyond conventional DevOps and addresses model selection, prompt versioning, dataset governance, model deployment, inference performance, semantic evaluation, hallucination monitoring, cost control, safety controls, and production observability. Although some of these topics have been discussed in prior studies, they are often addressed in the separate context of infrastructure-level differences between DevOps and LLM-Ops [5], which creates a research gap in understanding how compute, data, evaluation, security, sustainability, and tool execution requirements change when generative AI systems go into production.

The gap is narrowed by this review by comparing DevOps with LLM-Ops in a common set of infrastructure dimensions. The innovation lies in the joint consideration of traditional operational issues and new agentic requirements. The tool-centric execution of activities and controlled access to APIs,


This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)

databases, command-line tools, and Model Context Protocol interfaces are elevated to an infrastructure layer, further developing DevOps principles.

2. Review Methodology

This study uses a comparative review method rather than evaluating a single product, implementation, or benchmark. DevOps and LLM-Ops are compared across artifact management, compute, storage, networking, deployment, evaluation, observability, security, sustainability, and tool-oriented automation. These dimensions were selected because they recur across the cited literature and directly affect the production operation of generative AI systems.

The evidence base includes peer-reviewed papers, technical reports, standards, and vendor documentation on DevOps, LLM serving, Retrieval-Augmented Generation, the Model Context Protocol, AI security, model evaluation, and data-center energy demand. Sources are used when they provide direct evidence for at least one comparison dimension, such as GPU memory, serving latency, vector retrieval, semantic evaluation, security controls, or external tool execution. Peer-reviewed and standards-based sources are used for general claims, while vendor documentation is used mainly for product-specific technical specifications. This approach improves traceability between each infrastructure claim and its supporting source.

3. DevOps and LLM-Ops: Conceptual Difference

DevOps and LLM-Ops differ mainly in the artifacts and runtime behavior that they must manage. In DevOps, source code is the primary artifact. The code is built into an application package or container image and then deployed through continuous integration and continuous delivery pipelines. The same tested package can be released across multiple environments.

Table 1. Operational comparison between DevOps and LLM-Ops

Dimension	DevOps	LLM-Ops
Primary artifact	Source code, binaries, containers	Model weights, prompts, embeddings, datasets
Execution behavior	Deterministic	Probabilistic and context-sensitive
Compute model	CPU-centric or general cloud compute	GPU/TPU/NPU-accelerated inference
Testing method	Unit, integration, regression testing	Semantic evaluation, hallucination checks, model judging
Data dependency	Application databases	Databases, vector stores, retrieval pipelines
Deployment goal	Continuous delivery	Continuous delivery + continuous evaluation
Monitoring	Logs, metrics, traces	Logs, metrics, traces, prompt drift, output quality
Security risks	Code vulnerabilities, misconfiguration	Prompt injection, data leakage, unsafe tool use
Scaling concern	Service replicas and traffic routing	GPU memory, batching, latency, token throughput

4. Infrastructure Requirements for Generative AI

4.1. Compute and Memory Requirements

Generative AI inference is usually limited by GPU memory capacity, memory bandwidth, and token latency. Larger models need more memory to fit model weights and keep the key-value cache resident for inference. This is why many cutting-edge AI infrastructures are designed around GPU dense nodes instead of generic CPU-based cloud instances.

NVIDIA H200 features 141GB HBM3e memory and 4.8TB/s memory bandwidth. This showcases the direction towards High Bandwidth Memory (HBM) designs for generative AI and HPC workloads. GB200 NVL72 highlights the trend of rack-scale AI systems by bundling 36 Grace CPU and 72 Blackwell GPU together in a liquid cooled NV Link domain [9,10].

Traditional DevOps scaling usually increases application replicas, adds nodes, or uses autoscaling groups. LLM-Ops adds model-specific scaling constraints. These include batch size, memory efficiency, model parallelism, tensor parallelism, quantization, context length, and GPU scheduling. Operations teams must therefore coordinate software-level orchestration with accelerator memory, bandwidth, and scheduling limits.

4.2. Storage and Artifact Management

Artifact management in DevOps commonly consists of container images, dependency packages, build artifacts, and configuration files. Artifact management for LLM-Ops requires substantially more artifacts. Individual model checkpoints may be several GBs to hundreds of GBs in size. Prompts need versioning since a few characters can change how a model behaves. Vector indexes used for embedding storage need to be tracked since the retrieval results get surfaced in generated responses.

A production LLM system can include base models, fine-tuned models, LoRA adapters, tokenizers, prompt templates, safety policies, vector indexes, evaluation datasets, and model configuration files. These artifacts make release governance more complex than in conventional software systems. Reproducibility depends on the code version and also on the model version, prompt version, retrieval snapshot, and inference hyperparameters [20].

4.3. Networking and Distributed Inference

Traditional web applications mostly depend on communication between users, load balancers, application services, and databases. This is usually called north-south traffic. Generative AI workloads also require communication between GPUs, servers, and storage systems. This type of communication is important for distributed inference and training. Distributed inference could include tensor parallelism, pipeline parallelism, parameter synchronization and shared key-value cache. They all need fast GPU to GPU and node-to-node communication. Thus, high-performance interconnects like NVLink, InfiniBand, and RDMA over Converged Ethernet are also considered for large model serving. In LLM-Ops, network latency and bandwidth can directly affect token generation speed and response time. Slow communication between GPUs can reduce serving performance. Therefore, networking is a core infrastructure requirement for distributed inference, not only a deployment concern.

4.4. Model Serving and Inference Optimization

LLM serving has performance constraints that differ from those of conventional web services. Autoregressive models generate tokens sequentially, and the key-value cache grows with sequence length. Inefficient cache management can waste GPU memory and reduce batching capacity. The vLLM project introduced PagedAttention to improve key-value cache management and reported higher throughput under similar latency conditions than earlier serving systems [6].

LLM-Ops serving platforms should focus on optimizing for time to first token, tokens per second, efficient request batching, prefill and decode scheduling, streaming responses, and cost per inference. Inference optimizations like quantization, speculative decoding, and continuous batching have been leveraged for model parallelism and cache reuse to improve production serving performance [16-18, 21-22].

5. Data Infrastructure, RAG, and Evaluation

Generative AI applications can require access to knowledge that is not reliably stored in model parameters alone. Retrieval-Augmented Generation combines parametric memory in model weights with non-parametric memory in external documents, indexes, and vector databases. Early RAG research showed that external retrieval can improve knowledge-intensive generation tasks [7]. In LLM-Ops, the retrieval pipeline can include document ingestion, chunking, embedding generation, vector indexing, metadata filtering, hybrid search, reranking, and retrieval evaluation. Examples of vector-search platforms include Pinecone, Qdrant, Weaviate, Milvus, and Elasticsearch-based vector search [24-29]. Vector retrieval can also be combined with keyword search when exact identifiers, part numbers, dates, or proper nouns are important. This makes retrieval quality an operational concern that must be evaluated together with model output quality. Example evaluation measures include answer relevance, faithfulness, context precision and recall, hallucination rate, toxicity, refusal correctness, and tool-use correctness. RAGAS provides one example of automated evaluation across retrieval and generation dimensions [8]. Continuous evaluation therefore becomes a production control that complements continuous delivery.

Table 2. Evaluation difference between DevOps and LLM-Ops

Evaluation area	DevOps	LLM-Ops
Correctness	Pass/fail test assertions	Semantic similarity, factuality, faithfulness
Regression	Unit and integration regression	Prompt/model/retrieval regression
Observed output	Logs, API responses, database state	Model answers, citations, tool traces
Failure type	Code defect or service failure	Hallucination, irrelevant retrieval, unsafe response
Quality gate	Test coverage and build status	Evaluation score, safety score, cost and latency

6. Agentic Workflows and Tool-Oriented Infrastructure

A major operational difference between DevOps and LLM-Ops is the emergence of agentic workflows. Traditional DevOps automation is usually implemented with scripts, CI/CD pipelines, infrastructure-as-code templates, or scheduled jobs. Agentic AI systems can select tools, invoke APIs, query databases, run command-line operations, read files, generate reports, and coordinate multi-step workflows. These capabilities create infrastructure requirements for controlled tool access, execution policy, and end-to-end traceability.

This allows for an entirely new layer of infrastructure: tool-oriented execution. Models should not be permitted to execute arbitrary tools without input validation, authentication, policy checks, and observability. The infrastructure needs to govern which tools can be accessed,

what arguments can be passed, how outputs are returned, how secrets are handled, and how execution is traced.

The Model Context Protocol is one emerging open standard for tool-oriented execution. MCP servers can make resources, prompts, and tools available to AI language models. The official specification defines tools as executable functions, while resources are contextual data and prompts are reusable message workflows [11,12].

MCP will not be enough for production-ready LLM-Ops. Enterprises will also need execution harnesses surrounding tools. A Command-Line Interface (CLI) harness can validate arguments, prevent dangerous operations, enforce allowlists, record execution logs, implement timeouts, and return structured results back to the model. This becomes critical when allowing agents to access cloud platforms, Kubernetes clusters, databases, file systems, or CI/CD workflows.

Table 3. Tool-oriented infrastructure layers for agentic LLM-Ops

Layer	Infrastructure role
Agent interface	Receives user goals and converts them into model-driven plans.
Tool registry	Defines tools, schemas, permissions, and metadata.
MCP/API adapter	Standardizes model-to-tool communication.
CLI harness	Executes controlled command-line operations.
Policy and security	Enforces authentication, authorization, and sandboxing.
Observation	Records tool calls, decisions, latency, errors, and outputs.
Evaluation	Measures task success, safety, relevance, and correctness.

7. Observability, Security, and Sustainability

DevOps observability commonly uses logs, metrics, and traces. OpenTelemetry defines these telemetry signals as core sources of operational evidence [19]. LLM-Ops requires the same signals and also needs model-specific context such as prompt versions, retrieved context, model responses, token usage, latency, tool-call traces, evaluation scores, and user ratings. Security requirements also expand. In addition to code vulnerabilities, access control, dependency risk, and infrastructure misconfiguration, LLM systems can be exposed to prompt injection, training-data poisoning, model denial of service, sensitive-information leakage, and unsafe tool execution. Controls can include retrieval filtering, output validation, human approval gates, tool permissions, and audit trails [13,14]. Sustainability is also relevant because AI training and inference increase data-center energy demand. Data centers consumed about 415 TWh of electricity in 2024, and the IEA projects about 945 TWh by 2030 in its base case [15]. Energy-aware LLM-Ops can therefore use quantization, smaller task-specific models, batching, caching, distillation, efficient retrieval, and model scheduling to reduce resource use [15,21,22].

8. Discussion

The comparison shows that DevOps and LLM-Ops are not competing disciplines. DevOps remains necessary for software delivery, automation, cloud deployment, and service reliability. However, LLM-Ops adds several new requirements that are specific to generative AI.

First, LLM-Ops expands the artifact boundary from code to models, prompts, embeddings, and evaluation datasets. Second, it changes testing from deterministic pass/fail validation to semantic and probabilistic evaluation. Third, it changes compute requirements from general-purpose CPU infrastructure to GPU-optimized serving platforms. Fourth, it introduces retrieval-aware data infrastructure through vector databases and hybrid search. Fifth, it requires new observability methods for model behavior, prompt drift, hallucination, and tool execution. Finally, agentic AI introduces tool-oriented infrastructure where models interact with external systems and require strict governance.

Therefore, organizations adopting generative AI should avoid treating LLM applications as ordinary web services. They should preserve mature DevOps practices but extend them with model governance, evaluation pipelines, retrieval validation, inference optimization, secure tool orchestration, and energy-aware deployment planning.

9. Practical Implications and Future Scope

The practical implication is that a generative AI workload should not be operated as a conventional web-service endpoint without additional controls. A production LLM-Ops workflow should govern prompts, retrieval quality, model versions, cost, tool invocation, and AI-specific risks. For example, a conventional deployment pipeline may verify that an API starts successfully. An LLM-Ops pipeline should also verify that responses are grounded in retrieved context, tool calls are authorized, evaluation thresholds are met, and latency or token costs remain within defined limits.

These controls must operate throughout the model lifecycle, not only at deployment time. Model size affects GPU memory use. Context length affects key-value cache pressure. Retrieval design affects answer quality. Tool access affects security exposure. For this reason, LLM-Ops requires coordination among platform engineers, security engineers, data engineers, model developers, site-reliability engineers,

and application teams. A practical adoption sequence is to establish reproducible model and prompt versioning, add retrieval and semantic evaluation gates, instrument model and tool telemetry, enforce tool-access controls, and then optimize accelerator cost and energy use.

Infrastructure area	DevOps baseline	LLM-Ops extension
Release pipeline	Build, test, deploy containers	Add prompt, model, retrieval, and evaluation gates
Runtime platform	Service autoscaling and health checks	Add GPU scheduling, batching, cache, and token latency controls
Data layer	Application data stores	Add vector indexes, document pipelines, hybrid retrieval, and reranking
Security layer	Identity, secrets, and network policies	Add prompt-injection defense, tool permissions, sandboxing, and audit trails.
Reliability layer	Logs, metrics, traces, SLOs	Add answer quality, hallucination, drift, tool traces, and cost observability.
Sustainability layer	Resource utilization monitoring	Add quantization, smaller models, caching, distillation, and energy-aware scheduling.

Future work should define common standards for agentic infrastructure, secure execution around Model Context Protocol interfaces, repeatable methods for measuring RAG quality against changing enterprise data, and cost-aware inference scheduling. Additional work should also define common telemetry schemas for prompts, retrieved context, tool calls, internal model outputs, and human feedback. These standards would improve auditability and simplify integration with existing DevOps and site-reliability engineering practices.

10. Conclusion

This review compares DevOps and LLM-Ops for production generative AI systems. DevOps remains the foundation for automation, deployment, observability, and service reliability, but it does not by itself address all operational requirements of probabilistic, model-based, retrieval-augmented, and tool-using systems. LLM-Ops extends DevOps with model artifact management, prompt governance, semantic evaluation, GPU-optimized inference serving, retrieval infrastructure, AI-specific observability, and security controls for agentic workflows.

Agentic workflows and tool-oriented infrastructure are also emerging as important areas for LLM-Ops research. Large language models can interact with APIs, databases, command-line tools, and cloud systems, so production platforms must control these actions through policy, sandboxing, auditability, and continuous evaluation. For practitioners, the recommended path is to retain mature DevOps controls and add model versioning, retrieval evaluation, secure tool orchestration, AI-specific observability, and resource-aware inference management. Future research should develop benchmarks for agentic infrastructure, secure MCP-based execution, cost-aware scheduling, and sustainable deployment patterns for large language models.

Conflicts of Interest

The author declares that there is no conflict of interest regarding the publication of this paper.

Funding Statement

This research received no external funding.

References

[1] Len Bass, Ingo Weber, and Liming Zhu, *DevOps: A Software Architect's Perspective*, Addison-Wesley, 2015. [Google Scholar]

[2] Ramtin Jabbari et al., "What is DevOps? A Systematic Mapping Study on Definitions and Practices," *Workshops: Proceedings of the Scientific Workshop Proceedings of XP2016*, Edinburgh Scotland UK, pp. 1-11, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[3] Red Hat, What is CI/CD?, Technical Article, 2025. [Online]. Available: https://www.redhat.com/en/topics/devops/what-is-ci-cd

[4] Microsoft, What is Infrastructure as Code?, Azure DevOps Documentation, 2024. [Online]. Available: https://learn.microsoft.com/

[5] Saurabh Pahune, and Zahid Akhtar, "Transitioning from MLOps to LLMOps: Navigating the Unique Challenges of Large Language Models," *Information*, vol. 16, no. 2, pp. 1-35, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[6] Woosuk Kwon et al., "Efficient Memory Management for Large Language Model Serving with PagedAttention," *Proceedings of the 29th Symposium on Operating Systems Principles*, Koblenz Germany, pp. 611-626, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[7] Patrick Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1-16, 2020. [Google Scholar] [Publisher Link]

[8] Shahul Es et al., "RAGAS: Automated Evaluation of Retrieval Augmented Generation," *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, St. Julians, Malta, pp. 150-158, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[9] NVIDIA, NVIDIA H200 Tensor Core GPU, Technical Documentation, 2024. [Online]. Available: https://www.nvidia.com/en-us/data-center/h200/

[10] NVIDIA, NVIDIA GB200 NVL72, Technical Documentation, 2025. [Online]. Available: https://www.nvidia.com/en-us/data-center/gb200-nvl72/

[11] Anthropic, Model Context Protocol Specification, Technical Specification, 2025. [Online]. Available: https://modelcontextprotocol.io/

[12] Anthropic, Introducing the Model Context Protocol, Technical Article, 2024. [Online]. Available: https://www.anthropic.com/

[13] OWASP Foundation, OWASP Top 10 for Large Language Model Applications, Security Guidance, 2025. [Online]. Available: https://genai.owasp.org/

[14] Chloe Autio et al., "Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile," *National Institute of Standards and Technology*, Gaithersburg, pp. 1-64, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[15] International Energy Agency, Energy and AI, Technical Report, 2025. [Online]. Available: https://www.iea.org/reports/energy-and-ai

[16] Ying Sheng et al., "FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU," *Proceedings of the 40th International Conference on Machine Learning*, vol. 202, pp. 31094-31116, 2023. [Google Scholar] [Publisher Link]

[17] Yaniv Leviathan, Matan Kalman, and Yossi Matias, "Fast Inference from Transformers via Speculative Decoding," *Proceedings of the 40th International Conference on Machine Learning*, vol. 202, pp. 19274-19286, 2023. [Google Scholar] [Publisher Link]

[18] Gyeong-In Yu et al., "Orca: A Distributed Serving System for Transformer-Based Generative Models," *16th USENIX Symposium on Operating Systems Design and Implementation*, Carlsbad, CA, USA, pp. 1-19, 2022. [Google Scholar] [Publisher Link]

[19] OpenTelemetry, Observability Primer, Technical Documentation, 2026. [Online]. Available: https://opentelemetry.io/docs/concepts/observability-primer/

[20] Matei Zaharia et al., "Accelerating the Machine Learning Lifecycle with MLflow," *IEEE Data Engineering Bulletin*, vol. 41, no. 4, pp. 39-45, 2018. [Google Scholar] [Publisher Link]

[21] Tim Dettmers et al., "LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale," *Advances in Neural Information Processing Systems*, vol. 35, pp. 30318-30332, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[22] Elias Frantar et al., "GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers," *International Conference on Learning Representations*, pp. 1-16, 2023. [Google Scholar] [Publisher Link]

[23] James Jie Pan, Jianguo Wang, and Guoliang Li, "Survey of Vector Database Management Systems," *arXiv preprint*, pp. 1-25, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[24] Gordon V. Cormack, Charles L.A. Clarke, and Stefan Buettcher, "Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods," *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 758-759, 2009. [CrossRef] [Google Scholar] [Publisher Link]

[25] Pinecone, Pinecone Documentation, Technical Documentation, 2026. [Online]. Available: https://docs.pinecone.io/

[26] Qdrant, Qdrant Documentation, Technical Documentation, 2026. [Online]. Available: https://qdrant.tech/documentation/

[27] Weaviate, Weaviate: The AI Database Developers Love, Technical Documentation, 2026. [Online]. Available: https://weaviate.io/

[28] Milvus, Milvus Vector Database Documentation, Technical Documentation, 2026. [Online]. Available: https://milvus.io/

[29] Elastic, Vector Search in Elasticsearch, Elastic Documentation, 2026. [Online]. Available: https://www.elastic.co/