Acta Informatica Pragensia
2026, Volume 15, Issue 2, 601–620
https://doi.org/10.18267/j.aip.313

PRAGUE UNIVERSITY OF ECONOMICS AND BUSINESS

Miscellanea Open Access

Cloud-Based Large Language Model Deployment: A Comparative Analysis of Serverless and Bring-Your-Own-Container Architectures

Mateusz Ploskonka

Faculty of Informatics and Statistics, Prague University of Economics and Business, Czech Republic

Corresponding author: Mateusz Ploskonka (plom04@vse.cz)

Editorial Record

First submission received:
January 29, 2026

Revision received:
March 15, 2026

Accepted for publication:
March 16, 2026

Academic Editor:
Michal Munk
Constantine the Philosopher University in Nitra, Slovakia

This article was accepted for publication by the Academic Editor upon evaluation of the reviewer's comments.

How to cite this article:
Ploskonka, M. (2026). Cloud-Based Large Language Model Deployment: A Comparative Analysis of Serverless and Bring-Your-Own-Container Architectures. *Acta Informatica Pragensia*, 15(2), 601–620. https://doi.org/10.18267/j.aip.313

Abstract

Background: Large Language Models (LLMs) have transformed research and industry applications; however, cloud deployment decisions remain complex and poorly documented, particularly for academic researchers operating under budget constraints. Systematic guidance on infrastructure selection for LLM-based research is limited.

Objective: This study provides a comprehensive empirical evaluation of cloud-based LLM deployment architectures, examining inference efficiency, serverless platform availability, and architectural trade-offs across major cloud providers to deliver actionable guidance for budget-constrained researchers.

Methods: The author evaluated 32 open-source LLMs ranging from 0.6 billion to 1 trillion parameters across serverless and Bring Your Own Container (BYOC) deployment configurations. Using the Belebele benchmark, we analyzed cost–efficiency relationships, serverless platform availability, and metrics exposure across Amazon SageMaker, Amazon Bedrock, Azure Serverless, and Hugging Face–compatible providers.

Results: Model performance follows a logarithmic scaling relationship with parameter count (R^2=0.727) and deployment cost (R^2=0.639). Models in the 30–50B parameter range achieve 85–90% of maximum accuracy at a fraction of the cost of frontier models. However, serverless availability remains fragmented: only 34.4% of examined models are accessible via serverless endpoints, with minimal cross-platform redundancy (6.2%). Deployment architecture introduces a fundamental trade-off: serverless platforms expose 71% fewer metrics than BYOC approaches while eliminating infrastructure management overhead and idle costs.

Conclusion: These findings provide practical guidance for researchers selecting cloud infrastructure under budget constraints. Models in the 7–14B range offer optimal cost efficiency, while the 30–50B range maximizes accuracy per dollar for demanding tasks. The results also challenge the prevailing emphasis on ever-larger models, as diminishing returns become substantial beyond 30B parameters. Persistent gaps in serverless availability and observability highlight the need for greater standardization in cloud platforms.

Index Terms

LLMs; Cloud computing; Serverless architecture; Cost optimization; Performance evaluation; Model deployment; Infrastructure selection.

1 INTRODUCTION

Large Language Models (LLMs) have transformed industries, education, and everyday digital interactions since the public release of ChatGPT by OpenAI in November 2022. Their rapid adoption is reflected in both academic research and practical applications across diverse domains.

A large-scale analysis of 59,293 LLM-related publications shows a substantial increase in research activity following the introduction of ChatGPT, with growth observed not only in computer science but also in social sciences, psychology, health professions, and interdisciplinary fields (Li et al., 2024a).

This trend is further evidenced by publication data. The arXiv repository recorded 3,554 LLM-related submissions in 2023, 15,050 in 2024, and 31,272 in 2025 (as of November 22). Similarly, Google Scholar reports an increase from 47,000 publications in 2023 to 146,000 in 2025. In parallel, the number of models released on Hugging Face has grown rapidly, from 325,321 in 2023 to over 1 million in 2025. These trends confirm sustained expansion in both research output and model availability.

Despite this growth, LLM adoption in research remains constrained by infrastructure complexity and cost. Although inference costs are decreasing (Appenzeller, 2024; Xiao et al., 2025), they remain a significant barrier for many researchers. For example, training the Llama-3.3-70B model required approximately 39.3 million GPU hours on H100 hardware, corresponding to an estimated cost of $270 million based on public cloud pricing (Meta AI, 2024). While inference workloads are less expensive, they can still generate substantial operational costs. Hosting the same model for 24 hours on a major cloud provider can exceed $325. Such costs limit the feasibility of multi-model evaluations, large-scale experiments, and extended inference workloads.

At the same time, researchers face complex architectural decisions when deploying LLMs in cloud environments. Options include fully managed serverless platforms and Bring Your Own Container (BYOC) approaches, each with distinct trade-offs in cost, scalability, and observability. However, systematic, empirically grounded guidance for selecting between these architectures remains limited, particularly for research-oriented use cases with constrained budgets and flexible performance requirements.

This paper addresses this gap by providing an empirical analysis of cloud-based LLM deployment strategies. It examines architectural trade-offs, cost–performance relationships, and platform availability across multiple cloud providers. The goal is to provide practical guidance for researchers selecting infrastructure for LLM-based work. The study focuses on two primary areas: (1) cloud-based LLM deployment architectures and (2) model selection and evaluation strategies. Within this scope, the paper addresses the following research questions:

- What are the key considerations when selecting cloud architectures for LLM-based research projects?
- Which deployment strategies are most suitable for different research workloads?
- How should deployment architectures scale with model size and computational requirements?
- What principles can guide the selection of LLMs under cost and performance constraints?

Key Contributions

This paper makes the following contributions to the field of cloud-based LLM deployment:

1. A comprehensive multi-cloud empirical comparison of serverless (Amazon Bedrock, Azure Serverless) and BYOC (Amazon SageMaker) deployment architectures, evaluated across 32 open-source models spanning 0.6B to 1T parameters — the first study of this scope in the literature.
2. Quantification of logarithmic scaling laws in production deployment contexts, identifying a 30–50B parameter sweet spot that achieves 85–90% of maximum accuracy at a fraction of frontier model costs, challenging the prevailing industry emphasis on ever-larger models.
3. A systematic analysis of serverless LLM availability fragmentation, demonstrating 34.4% coverage across major platforms and only 6.2% cross-platform redundancy, with implications for vendor lock-in and deployment flexibility.
4. A comparative observability analysis revealing that serverless platforms expose 71% fewer operational metrics than BYOC approaches, highlighting critical cloud platform maturity gaps requiring standardization.
5. Actionable deployment guidance for budget-constrained researchers, including cost efficiency rankings, architecture selection criteria, and a public evaluation repository supporting full reproducibility.

2 LITERATURE REVIEW

2.1 LLM inference architectures and serving systems

The foundational challenge in LLM deployment stems from the autoregressive nature of token generation, which requires multiple forward passes per inference request. Yu et al. (2022) introduced Orca, a distributed serving system that pioneered iteration-level scheduling rather than request-level scheduling. By invoking the execution engine to run only a single iteration of the model per scheduling decision, Orca achieved a 36.9× throughput improvement over NVIDIA Faster Transformer at equivalent latency levels. This work established the importance of fine-grained scheduling for generative models, where requests in a batch may complete at different iterations.

Building on this foundation, Kwon et al. (2023) addressed the severe memory fragmentation problem in key-value (KV) cache management through PagedAttention, an algorithm inspired by virtual memory and paging in operating systems. Their vLLM system partitions the KV cache into non-contiguous blocks, achieving near-optimal memory usage with less than 4% waste compared to traditional pre-allocation schemes that utilize only 20-38% of allocated memory. The combination of PagedAttention with continuous batching enabled vLLM to achieve 24× throughput improvements over HuggingFace Transformers and 2.7×-8× improvements over Orca variants on production workloads (Kwon et al., 2023).

NVIDIA's TensorRT-LLM (NVIDIA, 2023, 2025) provides an alternative approach through comprehensive kernel optimizations, including FlashAttention implementations, quantization support (FP8, INT4, AWQ), and in-flight batching. Performance evaluations demonstrate 8× throughput improvements for GPT-J-6B and 4.6× speedups for Llama 2 70B on H100 GPUs (NVIDIA, 2023). Recent additions include speculative decoding support through ReDrafter integration, achieving up to 2.7× throughput improvements on H100 GPUs with tensor parallelism (NVIDIA, 2025).

The evolution toward serverless LLM deployment has introduced new architectural considerations. The ServerlessLLM system (Fu et al., 2024), accepted to OSDI 2024, implements a scalable checkpoint storage layer on GPU servers through an innovative checkpoint format, multi-tier loading subsystem, and locality-friendly architecture. More recently, Aegaeon (Xiang et al., 2025) demonstrated production-scale GPU pooling at Alibaba Cloud Model Studio, reducing GPU requirements from 1,192 to 213 (82% savings) while serving tens of models ranging from 1.8B to 72B parameters through advanced request scheduling and model multiplexing.

A comprehensive survey by Li et al. (2024b) provides a systematic taxonomy of these serving systems and identifies key optimization opportunities across KV cache management, computation optimization, and cloud deployment strategies.

2.2 Cost optimization techniques

Quantization has emerged as a critical technique for reducing both memory requirements and inference costs. Lin et al. (2025) introduced Activation-aware Weight Quantization (AWQ), which identifies that only 1% of model parameters contribute significantly to outputs. By preserving higher precision for these salient weights while quantizing the remaining 99% to INT4, AWQ achieves 4× memory reduction with minimal quality degradation. Frantar et al. (2023) developed GPTQ (Generative Pre-trained Transformer Quantization), a post-training quantization method widely adopted in the community through TheBloke's model distributions.

Production deployments demonstrate substantial cost savings through quantization. Oracle's implementation (Oracle, 2024) reports 10% latency reduction for Llama 3.2-90B with FP8 quantization, while Llama 3.3-70B achieves 99%+ quality recovery with 30% latency reduction and 50% throughput improvement. Experimental INT4 quantization shows over 50% per-GPU throughput improvement while reducing GPU requirements to 25% of FP16 baseline. Comparative analyses of quantization methods (FriendliAI, 2024) indicate that SmoothQuant offers the best speed performance, followed by AWQ, while OWQ, SpQR, and SqueezeLLM provide superior accuracy at the cost of computational complexity.

Beyond quantization, cloud infrastructure strategies offer significant cost optimization opportunities. Spot instance utilization provides 50-80% cost savings for interruptible workloads with proper checkpointing (GMI Cloud, 2025; Microsoft Azure, 2023). Case studies from production deployments illustrate effective strategies: Meta implements

hierarchical caching with dynamic scaling for variable loads; LinkedIn uses model distillation for skills-extraction features, maintaining accuracy while reducing costs; Mercari demonstrates 95% model size reduction through 8-bit quantization, achieving 14× inference cost reduction (Gun.io, 2025).

The systematic review by Saleh et al. (2025) provides a comprehensive framework for measuring cost-performance trade-offs, establishing methodologies that complement our empirical evaluation approach.

2.3 Scaling and orchestration strategies

Kubernetes has emerged as the dominant platform for orchestrated LLM deployment due to its support for GPU scheduling, auto-scaling, and declarative configuration. The Horizontal Pod Autoscaler (HPA) enables scaling based on custom metrics including gpu_utilization, inference_queue_depth, and tokens_per_second (Collabnix, 2024). KEDA (Kubernetes Event-Driven Autoscaling) extends these capabilities for queue-based scaling patterns common in asynchronous inference scenarios. Recent implementations demonstrate stabilization windows of 300 seconds for scale-down to prevent thrashing, while allowing aggressive scale-up with 60-second windows during traffic surges (Collabnix, 2024).

Serverless patterns for LLM deployment introduce additional architectural considerations. Knative provides scale-to-zero capabilities with GPU support, though cold start latency remains challenging for multi-gigabyte models (Yadav, 2025). Xu et al. (2025) present design principles for cloud-native LLM deployments addressing container orchestration, service mesh integration, and observability requirements specific to LLM workloads. Their work demonstrates that cloud-native architectures can dynamically adapt to workload fluctuations through Kubernetes-based autoscaling, mitigating performance bottlenecks while optimizing resource allocation.

Recent research on auto-scaling mechanisms includes TokenScale (Lai et al., 2025), which provides timely and accurate auto-scaling for disaggregated serving through token velocity analysis, and FlexPipe (Lin et al., 2026), which adapts pipeline parallelism configurations dynamically through in-flight refactoring in fragmented serverless clusters. These systems address the unique challenges of LLM auto-scaling: long startup times (30-120 seconds for model loading), variable request duration based on prompt and generation parameters, and significant throughput benefits from request batching.

2.4 Architectural patterns and design decisions

The systematic literature review by Schmid et al. (2025) identifies recurring architectural patterns in LLM deployments, including separation of prefill and decode phases, prompt caching strategies, and cascade routing with confidence-based model selection. This comprehensive analysis of 18 research articles examines which software architecture tasks LLMs are used for, automation levels, and evaluation approaches. The findings reveal that while LLMs are increasingly applied to various architecture tasks and often outperform baselines, areas such as generating source code from architectural design, cloud-native computing and architecture, and conformance checking remain underexplored.

For edge deployment scenarios, the survey by Semerikov et al. (2025) contrasts cloud-centric approaches with resource-constrained edge environments. While edge deployment introduces different constraints (limited memory, central processing unit (CPU)-only inference), the survey highlights techniques like aggressive quantization (INT4, INT8) and model distillation that have applicability across deployment contexts. Their analysis demonstrates that hybrid edge-microservices architectures achieve 46% lower P99 latency and 67% higher throughput compared to monolithic approaches, while supporting 10,000 concurrent users with 100ms latency constraints.

2.5 Research gaps and positioning

While existing literature provides extensive coverage of individual optimization techniques and system designs, several gaps remain that motivate our empirical study:

- **Research gap 1:** Multi-Cloud Comparative Analysis. Existing work focuses primarily on single-provider implementations or theoretical frameworks. Comprehensive empirical comparisons of Amazon SageMaker, Azure ML, and Nebius type of providers for LLM deployment—including cost structures and managed service capabilities—remain absent from the literature.

- **Research gap 2:** Research-Oriented Decision Frameworks. Current deployment guides target production systems with stringent Service Level Objectives (SLOs) requirements. Research scenarios present different constraints: limited budgets, varying workload patterns, exploratory experimentation, and tolerance for higher latency in exchange for cost savings. Systematic guidance for architecture selection under these constraints is lacking.
- **Research gap 3:** Size Scaling Thresholds. Although our prior work identified performance plateaus at 30B parameters for RAG tasks, the implications for deployment of Native LLM architectures remain unanswered.

This paper addresses these gaps through a systematic empirical evaluation of cloud deployment strategies for LLM-based research workloads, providing actionable guidance for architecture selection, deployment strategy optimization, and scaling decisions based on model size and computational requirements.

3 RESEARCH METHODS

3.1 Research design and approach

This study empirically evaluates cloud-based LLM deployment architectures. The author evaluated 32 open-source LLMs, ranging from 0.6 billion to 1 trillion parameters, across serverless and Bring Your Own Container (BYOC) deployment configurations. Research design addresses four research questions through three integrated experimental components: (1) model performance evaluation using the Belebele benchmark, (2) serverless availability analysis across major cloud providers, and (3) architectural metrics comparison between deployment paradigms. This approach enables analysis of both quantitative performance and operational characteristics.

3.2 Model selection and deployment configurations

The author evaluated 32 LLMs with parameter counts ranging from 0.6 billion to 1 trillion, released between September 2023 and August 2025. Model inclusion required open licensing, public availability, and open-source implementation to ensure reproducibility for academic researchers. An overview of selected models is available in Table 1.

Table 1. *LLMs selected for the research study, grouped by parameter size.*
Models range from 0.6B to 1T parameters, released between September 2023 and August 2025.

Parameter Range	Models	Key Developers
< 10B	Qwen.ai - Qwen-3-0.6B (May-25); Google - Gemma-3-1B-Instruct (Mar-25); Meta-llama - Llama-3.2-1B-Instruct (Sep-24); Tiiuae - Falcon-3-1B-Instruct (Dec-24); Google - Gemma-2-2B-Instruct (Jul-24); Meta-llama - Llama-3.2-3B-Instruct (Sep-24); Tiiuae - Falcon-3-3B-Instruct (Dec-24); Qwen.ai - Qwen-3-4B (May-25); Google - Gemma-3-4B-Instruct (Mar-25); Mistral.ai - Mistral-7B-Instruct-v0.1 (Oct-23); Tiiuae - Falcon-3-7B-Instruct (Dec-24); Meta-llama - Llama-3-8B-Instruct (Apr-24); Meta-llama - Llama-3.1-8B-Instruct (Jul-24); Qwen.ai - Qwen-3-8B (May-25); Google - Gemma-2-9B-Instruct (Jun-24)	Google, Meta-llama, Mistral.ai, Qwen.ai, Tiiuae
10-30B	Tiiuae - Falcon-3-10B-Instruct (Dec-24); Mistral.ai - Mistral-Small-Instruct (Mar-25); Google - Gemma-2-27B-Instruct (Jun-24); Google - Gemma-3-27B-Instruct (Mar-25)	Google, Mistral.ai, Tiiuae
30-100B	Qwen.ai - Qwen-3-32B (May-25); Meta-llama - Llama-3.1-70B-Instruct (Jul-24); Meta-llama - Llama-3-70B-Instruct (Apr-24); Meta-llama - Llama-3.3-70B-Instruct (Dec-24)	Meta-llama, Qwen.ai
≥ 100B	OpenAI - Gpt-oss-120B (Aug-25); Mistral.ai - Mistral-Large-Instruct (2407) (Feb-24); Qwen.ai - Qwen-3-235B-A22B-Instruct (2507) (May-25); Nvidia - Llama-3.1-Nemotron-Ultra-253B-v1 (Apr-25); Zaiorg - GLM-4.5 (Aug-25); AI21 - Jamba-Large-1.5 (May-24); Meta-llama - Llama-3.1-405B-Instruct-FP8 (Jul-24); NousResearch - Hermes-4-405B (Aug-25); Moonshotai - Kimi-K2-Instruct (Jan-25)	AI21, Meta-llama, Mistral.ai, Moonshotai, NousResearch, Nvidia, OpenAI, Qwen.ai, Zaiorg

For the primary performance and cost evaluation (Section 4.1), all 32 models were deployed using Amazon SageMaker JumpStart hosted endpoint architecture. This configuration provides:

1. Managed infrastructure: Pre-configured Elastic Compute Cloud (EC2) instances with GPU acceleration,
2. Model repository integration: Direct access to pre-trained models from SageMaker JumpStart catalog,
3. Consistent deployment environment: Standardized configuration across all models enabling controlled comparison. This architecture is presented in Figure 1.

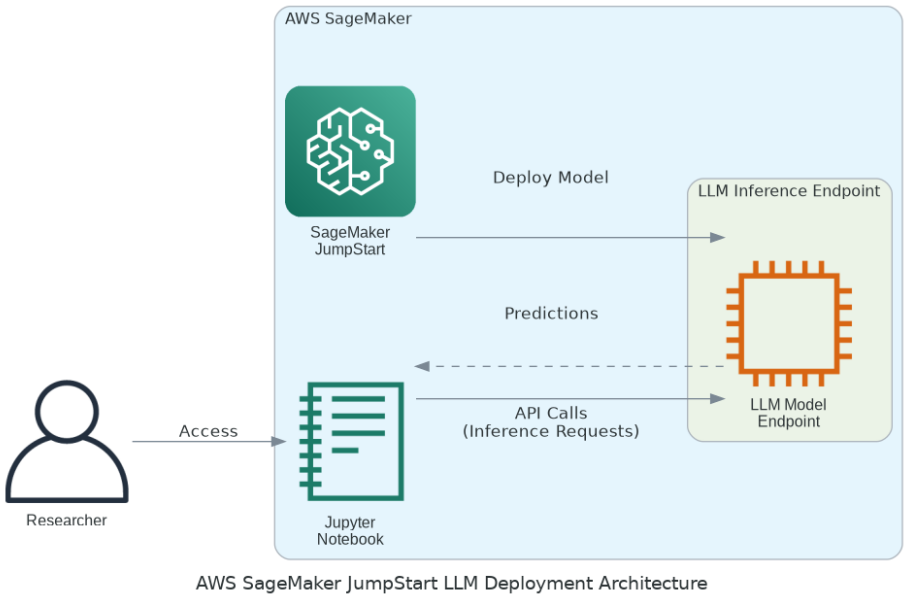

Figure 1. *Amazon SageMaker JumpStart Hosted Endpoint Architecture. This architecture represents the managed cloud deployment configuration, where pre-trained LLM models from SageMaker JumpStart are deployed to dedicated EC2 instances. The Jupyter Notebook environment connects to the inference endpoint via representational state transfer application programming Interface (REST API) calls.*

For architectural comparison and availability analysis (Sections 4.2-4.3), the author evaluated three additional deployments.

1. Amazon Bedrock Serverless Architecture: Serverless deployment configuration providing API-based access to foundation models without infrastructure provisioning. This architecture eliminates the need for EC2 instance management through fully managed, pay-per-use endpoints with automatic scaling as presented in Figure 2.

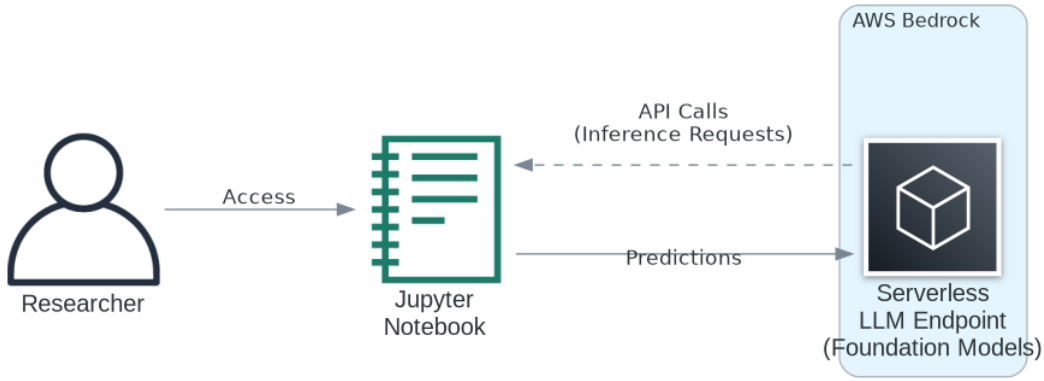

Figure 2. *Amazon Bedrock Serverless Endpoint Architecture. This architecture represents the serverless deployment configuration, where foundation models are accessed via Amazon Bedrock's managed API without infrastructure provisioning. Researchers interact with pre-trained models through API calls from a Jupyter Notebook environment, with Bedrock handling all scaling and resource management automatically.*

2. Azure Serverless Architecture: Microsoft Azure's managed serverless inference platform, evaluated for cross-platform availability comparison and multi-cloud deployment strategy assessment.

3. Nebius Token Factory Architecture: Third-party specialized LLM serving platform representing external API-based deployment options outside traditional cloud ecosystems (Figure 3).

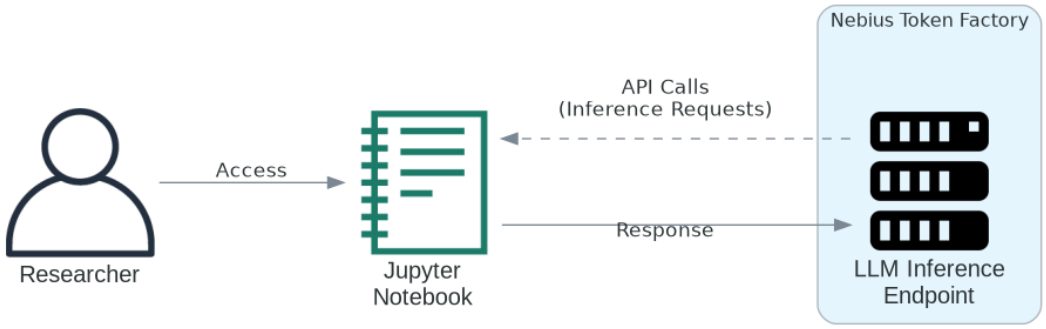

Figure 3. *Nebius Token Factory Inference Architecture. This architecture represents an external API-based deployment configuration using Nebius Token Factory. Researchers access LLM inference capabilities through direct API integration from a Jupyter Notebook environment, with Nebius managing the underlying infrastructure and model serving independently of cloud provider services.*

Overall overview of deployment architectures is presented in Table 2.

Table 2. *Deployment architecture configurations.*

Architecture	Type	Platform	Key Features	Use Case
Primary: BYOC	Managed Infrastructure	Amazon SageMaker JumpStart	Pre-configured EC2 instances, full metrics visibility, model repository integration	Performance/cost evaluation (Section 4.1)
Serverless	Fully Managed API	Amazon Bedrock	No infrastructure management, pay-per-use, automatic scaling	Availability analysis (Section 4.2)
Serverless	Fully Managed API	Azure Serverless	Cross-platform comparison, managed inference	Availability analysis (Section 4.2)
API-based	Third-party Platform	Nebius Token Factory	External serving platform, API integration	Availability analysis (Section 4.2)

3.3 Evaluation methodology

Our evaluation methodology comprised four integrated components as summarized in Table 3.

Table 3. *Evaluation components and procedures.*

Component	Data Source	Method	Output Metric
Performance	Belebele benchmark (multiple-choice reading comprehension)	API calls to deployed models, ground truth comparison	Weighted Accuracy (WA)
Cost	Amazon EC2 pricing (us-east-1, Nov 2025)	Instance type identification, hourly rate retrieval	Hourly deployment cost ($/hr); Cost efficiency (Accuracy/$)
Serverless Availability	Manual platform assessment (Amazon Bedrock, Azure, HuggingFace providers)	Binary availability check per model-platform pair	Coverage % by platform, family, size category
Architecture Metrics	Platform documentation (SageMaker, Bedrock)	Metric identification and categorization across 5 dimensions	Comparison matrix (28 metrics)

3.4 Statistical analysis

The author assessed relationships between model characteristics and performance using correlation analysis and regression modeling. Table 4 summarizes the statistical methods and implementation tools.

Table 4. *Statistical methods and tools.*

Analysis Type	Method	Purpose / Tools
Correlation	Pearson correlation coefficient (r) Spearman rank correlation (ϱ) Two-tailed tests (α=0.05)	Measure linear and monotonic relationships. Tool: Python scipy.stats
Regression	Linear, Polynomial (degree 2) Logarithmic (parameters and cost)	Predict accuracy from parameters/cost. Tool: Python scipy.stats
Model Evaluation	R^2 (coefficient of determination) RMSE, p-value	Measure variance explained and statistical significance
Infrastructure	Amazon Web Services (AWS) -us-east-1 SageMaker JumpStart, Jupyter	Primary deployment platform, notebook environment
Data Processing	Pandas matplotlib, seaborn Custom Python scripts	Data manipulation and aggregation Visualization Cost calculation (AWS pricing)

4 RESULTS

4.1 Impact of LLM model size and cost on response quality

To quantify the response quality of LLM systems, each model was evaluated using the Belebele dataset. Weighted accuracy was computed as the primary performance metric to provide an unbiased assessment of answer quality. Each model was deployed under Amazon SageMaker JumpStart hosted endpoint architecture. Deployment cost was estimated using EC2 pricing from November 2025 in the us-east-1 region.

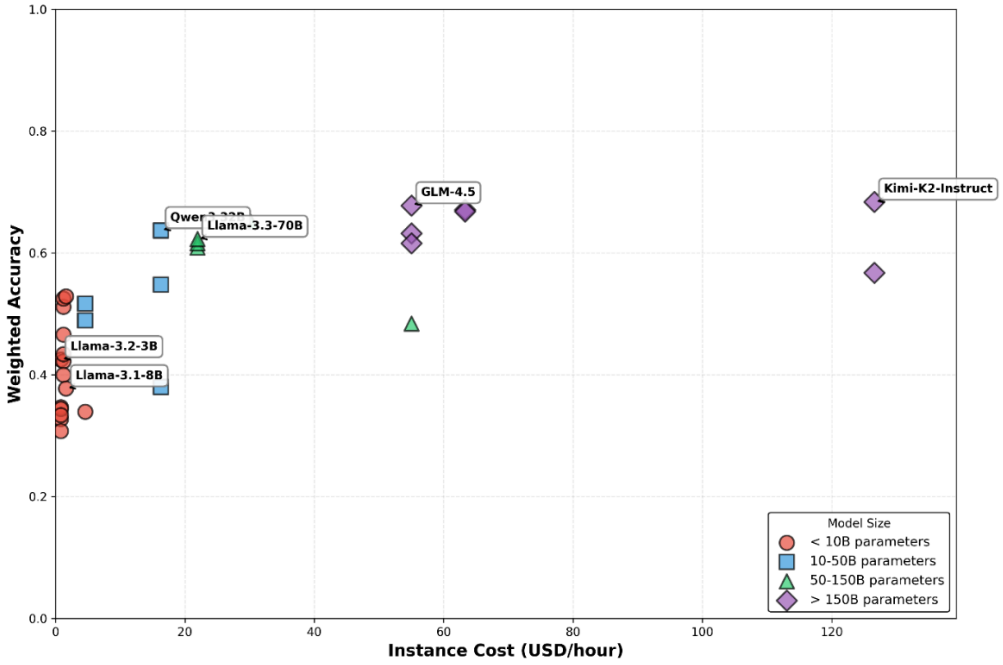

Figure 4. *Weighted accuracy for each model with hourly inference cost plotted on the x-axis and weighted accuracy on the y-axis.*

Figure 4 illustrates the evaluation outcomes for each model across all architectural configurations, with hourly inference cost plotted on the x-axis and weighted accuracy on the y-axis. Following this, statistical analysis was performed to evaluate relationships between variables. The author examined correlation patterns, regression models, and marginal efficiency metrics.

4.1.1 Correlation analysis

We conducted both Pearson and Spearman correlation analyses to assess linear and monotonic relationships between model characteristics and performance outcomes (Table 5).

Table 5. *Correlation analysis of model characteristics and performance.*

Relationship	Pearson r	Spearman ϱ	p-value	Interpretation
Parameters → Accuracy	0.598	0.838	<0.001	Strong monotonic, moderate linear
Cost → Accuracy	0.635	0.826	<0.001	Strong monotonic, moderate linear
Parameters → Cost	0.897	0.983	<0.001	Very strong linear and monotonic

Note: All correlations were statistically significant at p < 0.001.
Pearson r measures linear correlation; Spearman ρ measures monotonic correlation. N = 32 models.

Parameters exhibited a strong positive correlation with weighted accuracy (Spearman $\varrho = 0.838$, $p < 0.001$), indicating a robust monotonic relationship between model size and task performance. However, the substantially lower Pearson correlation coefficient ($r = 0.598$, $p < 0.001$) suggests this relationship is non-linear, with diminishing returns at larger scales.

Similarly, deployment cost demonstrated strong monotonic correlation with accuracy (Spearman $\varrho = 0.826$, $p < 0.001$), though again with a lower linear correlation (Pearson $r = 0.635$, $p < 0.001$). The strongest relationship observed was between model parameters and deployment cost (Pearson $r = 0.897$, Spearman $\varrho = 0.983$, both $p < 0.001$), indicating that infrastructure costs scale nearly proportionally with model size, likely due to GPU memory requirements.

4.1.2 Regression models for accuracy prediction

We evaluated four regression models to predict weighted accuracy from model parameters (Table 6).

Table 6. *Regression model performance for accuracy prediction.*

Model Type	Equation	R^2	RMSE	p-value
Linear (Parameters)	$Acc = 0.464 + 0.000351 \times P$	0.358	0.091	<0.001
Polynomial (deg 2)	$Acc = 0.441 + 0.000866 \times P - 6.52\times10^{-7} \times P^2$	0.502	0.080	<0.001
Logarithmic (Parameters)	$Acc = 0.329 + 0.126 \times \log_{10}(P)$	0.727	0.059	<0.001
Logarithmic (Cost)	$Acc = 0.362 + 0.149 \times \log_{10}(Cost)$	0.639	0.068	<0.001

Note: P = model parameters in billions. Acc = weighted accuracy. Bold indicates best-performing model. N = 32 models

Linear regression yielded poor predictive power ($R^2 = 0.358$), as did polynomial regression of degree 2 ($R^2 = 0.502$). The logarithmic model provided substantially superior fit:

$$\text{Accuracy} = 0.329 + 0.126 \times \log_{10}(\text{Parameters}) \quad (R^2 = 0.727,\ p < 0.001)$$

This logarithmic relationship indicates that each 10-fold increase in parameters yields approximately 0.126 units of accuracy gain, regardless of starting scale. For instance, scaling from 1B to 10B parameters produces similar accuracy improvements as scaling from 100B to 1,000B parameters, despite the latter requiring 90-fold more parameters.

When predicting accuracy from deployment cost, logarithmic regression achieved $R^2 = 0.639$ ($p < 0.001$), following the equation:

$$\text{Accuracy} = 0.362 + 0.149 \times \log_{10}(\text{Cost}) \ (R^2 = 0.639, p < 0.001)$$

This indicates that cost and accuracy also follow a logarithmic relationship, with diminishing accuracy returns at higher cost tiers. The moderately inferior performance of the cost-based model ($R^2 = 0.639$) compared to the parameter-based model ($R^2 = 0.727$) suggests that model architecture characteristics, beyond infrastructure requirements alone, influence task performance.

4.1.3 Cost efficiency metrics

We quantified cost efficiency using the ratio of weighted accuracy to hourly deployment cost (accuracy per dollar). Small models demonstrated substantially higher cost efficiency than their larger counterparts (Table 7).

Table 7. *Cost efficiency rankings – Top 10 most cost-efficient.*

Rank	Model	Parameters (B)	Cost ($/hr)	Accuracy	Acc/$
1	Llama-3.2-3B	3.0	0.80	0.425	0.528
2	Falcon3-7B	7.0	1.21	0.525	0.433
3	Gemma 3 1B	1.0	0.80	0.347	0.431
4	Llama-3.2-1B	1.0	0.80	0.344	0.428
5	Qwen3 4B	4.0	1.21	0.512	0.422
6	Falcon3-1B	1.7	0.80	0.334	0.415
7	Qwen3 0.6B	0.6	0.80	0.327	0.407
8	Llama-3.1-8B	8.0	1.21	0.466	0.385
9	Gemma-2-2B	2.0	0.80	0.308	0.383
10	Gemma 3 4B	4.0	1.21	0.434	0.358

The top-performing model in terms of cost efficiency was Llama-3.2-3B (0.528 accuracy per dollar, 3B parameters, $0.80/hr), followed by Falcon3-7B (0.433 accuracy per dollar, 7B parameters, $1.21/hr) and Gemma 3 1B (0.431 accuracy per dollar, 1B parameters, $0.80/hr). In contrast, the largest models exhibited markedly lower cost efficiency. Kimi-K2-Instruct (1T parameters, $126.59/hr) achieved only 0.0054 accuracy per dollar, representing a 98-fold reduction in cost efficiency compared to Llama-3.2-3B, despite delivering 61% higher absolute accuracy (0.684 vs. 0.425). Similarly, Hermes-4-405B demonstrated the lowest cost efficiency at 0.0045 accuracy per dollar.

4.1.4 Marginal efficiency analysis

To quantify diminishing returns, we calculated marginal efficiency as the average accuracy gain per billion parameters added within specific size ranges (Table 8).

Marginal efficiency declined dramatically with increasing model scale: 0.6B-3B range: 0.295 accuracy units per billion parameters, 3B-10B range: 0.086 accuracy units per billion parameters (71% reduction), 10B-50B range: 0.026 accuracy units per billion parameters (91% reduction from baseline), 50B-150B range: 0.007 accuracy units per billion parameters (98% reduction),150B-1000B range: 0.002 accuracy units per billion parameters (99% reduction).

The steepest decline occurred between the smallest two ranges, with a 71% reduction in marginal efficiency when transitioning from the 0.6-3B to the 3-10B parameter regime. Beyond 50B parameters, marginal efficiency approached near-zero values, suggesting a practical performance plateau.

Table 8. *Marginal efficiency by model parameter range.*

Parameter Range	N Models	Avg Accuracy	Avg Cost ($/hr)	Marginal Efficiency	Decline from Baseline
0.6B - 3B	7	0.333	0.80	0.295	— (baseline)
3B - 10B	8	0.443	1.59	0.086	71%
10B - 50B	7	0.514	11.61	0.026	91%
50B - 150B	5	0.597	30.21	0.007	98%
150B - 1000B	5	0.639	69.72	0.002	99%

Note: Marginal Efficiency = average accuracy per billion parameters within the range. Calculated as: (mean accuracy) / (mean parameters). Decline from baseline calculated relative to the 0.6-3B range. N = 32 total models.

Concurrently, average deployment costs increased substantially across size ranges: $0.80/hr (0.6-3B), $1.59/hr (3-10B), $11.61/hr (10-50B), $30.21/hr (50-150B), and $69.72/hr (150-1000B). The transition from the 50-150B to 150-1000B range incurred an average cost increase of $39.51/hr (131%) while delivering only 0.042 units of accuracy improvement (6.6%).

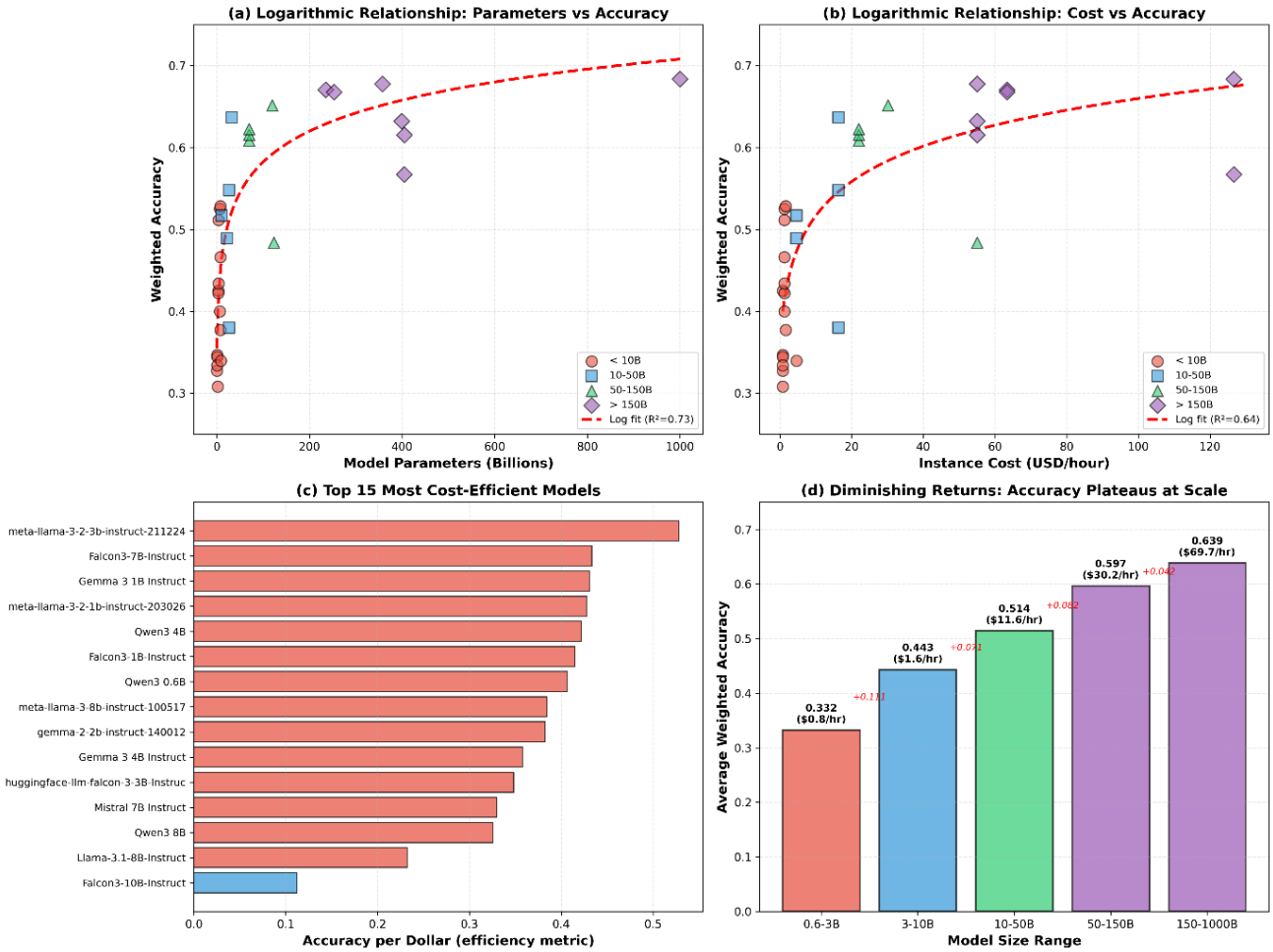

Figure 5. *Cost-performance analysis across 32 LLMs. (a) Logarithmic scaling: Accuracy = 0.329 + 0.126×log$_{10}$(Params), R^2=0.727. (b) Log cost scaling: Accuracy = 0.362 + 0.149×log$_{10}$(Cost), R^2=0.639. (c) Top 15 models by cost efficiency (accuracy/$). (d) Diminishing returns by model size range. All p < 0.001.*

4.1.5 Performance-cost sweet spot identification

We identified an optimal deployment range by analyzing the trade-offs between absolute accuracy and cost efficiency. Models in the 30-50B parameter range achieved 85-90% of maximum observed accuracy (0.51-0.60 vs. maximum 0.68) while maintaining moderate deployment costs ($5.67-$16.29/hr). This represents a 50-75% cost reduction compared to models exceeding 150B parameters, with minimal accuracy degradation.

Specifically, Qwen3 32B (32B parameters, $5.67/hr, 0.637 accuracy) demonstrated strong performance at moderate cost, positioning it at the intersection of high absolute accuracy and reasonable cost efficiency, with a cost efficiency ratio of 0.112 accuracy per dollar. Models smaller than 10B, while exhibiting superior cost efficiency (>0.35 accuracy/$), achieved substantially lower absolute accuracy (0.33-0.53), limiting their applicability to tasks requiring high performance. Figure 5 illustrates these relationships across four dimensions: (a) the logarithmic parameter-accuracy relationship with fitted regression line ($R^2 = 0.73$), (b) the logarithmic cost-accuracy relationship ($R^2 = 0.64$), (c) cost efficiency rankings for the top 15 models, and (d) average accuracy by parameter size range demonstrating diminishing returns. All statistical relationships reported were significant at $p < 0.001$.

4.2 Serverless LLM endpoint availability analysis

4.2.1 Platform coverage and cross-platform availability

Of the 32 LLMs examined, 11 models (34.4%) were available through serverless endpoints, while 21 models (65.6%) were not available on any serverless platform. Amazon Bedrock provided access to 5 models (15.6%), and Azure Serverless supported 8 models (25.0%). In comparison, 19 models (59.4%) were accessible through at least one Hugging Face-compatible provider (Groq, Novita, Together AI, Nebius AI, or Fireworks AI), representing a difference of 8 models (25.0 percentage points) between Hugging Face provider availability and serverless platform availability.

Among the 11 models available on serverless platforms, 2 models (6.2%) were accessible on both Amazon Bedrock and Azure Serverless: Gpt-oss-120B and Llama-3.1-70B-Instruct. The remaining 9 models (81.8% of serverless-available models) were exclusive to a single platform, with 3 models available only on Amazon Bedrock and 6 models available only on Azure Serverless. Additionally, 13 models (40.6% of total) were available through Hugging Face providers but not available on any serverless platform.

4.2.2 Model family distribution

Serverless availability varied across the ten model families examined. Table 9 presents the coverage for each family.

Table 9. *Serverless availability by model family.*

Family	n	AWS	Azure	Either	Coverage (%)
GPT	1	1	1	1	100.0
Llama	9	3	5	7	77.8
Mistral	3	0	2	2	66.7
Qwen	5	1	0	1	20.0
Families with Zero Serverless Availability					
GLM	1	0	0	0	0.0
Falcon	4	0	0	0	0.0
Jamba	1	0	0	0	0.0
Hermes	1	0	0	0	0.0

Family	n	AWS	Azure	Either	Coverage (%)
Gemma	6	0	0	0	0.0
Kimi	1	0	0	0	0.0

The Llama family had 7 of 9 models (77.8%) available on serverless platforms, with 3 models on Amazon Bedrock and 5 models on Azure Serverless. The Mistral family had 2 of 3 models (66.7%) available, both on Azure Serverless. The single GPT model (100%) was available on both platforms. The Qwen family had 1 of 5 models (20.0%) available on Amazon Bedrock. Six model families had zero models available on serverless platforms: Falcon (0 of 4 models), Gemma (0 of 6 models), GLM (0 of 1 model), Hermes (0 of 1 model), Jamba (0 of 1 model), and Kimi (0 of 1 model). These six families collectively comprised 13 models (40.6% of total models examined).

4.2.3 Model size category distribution

Models were categorized by parameter count into three size categories: Large (≥100B parameters, n=10), Medium (10-100B parameters, n=7), and Small (<10B parameters, n=15). Table 10 presents serverless availability across these categories.

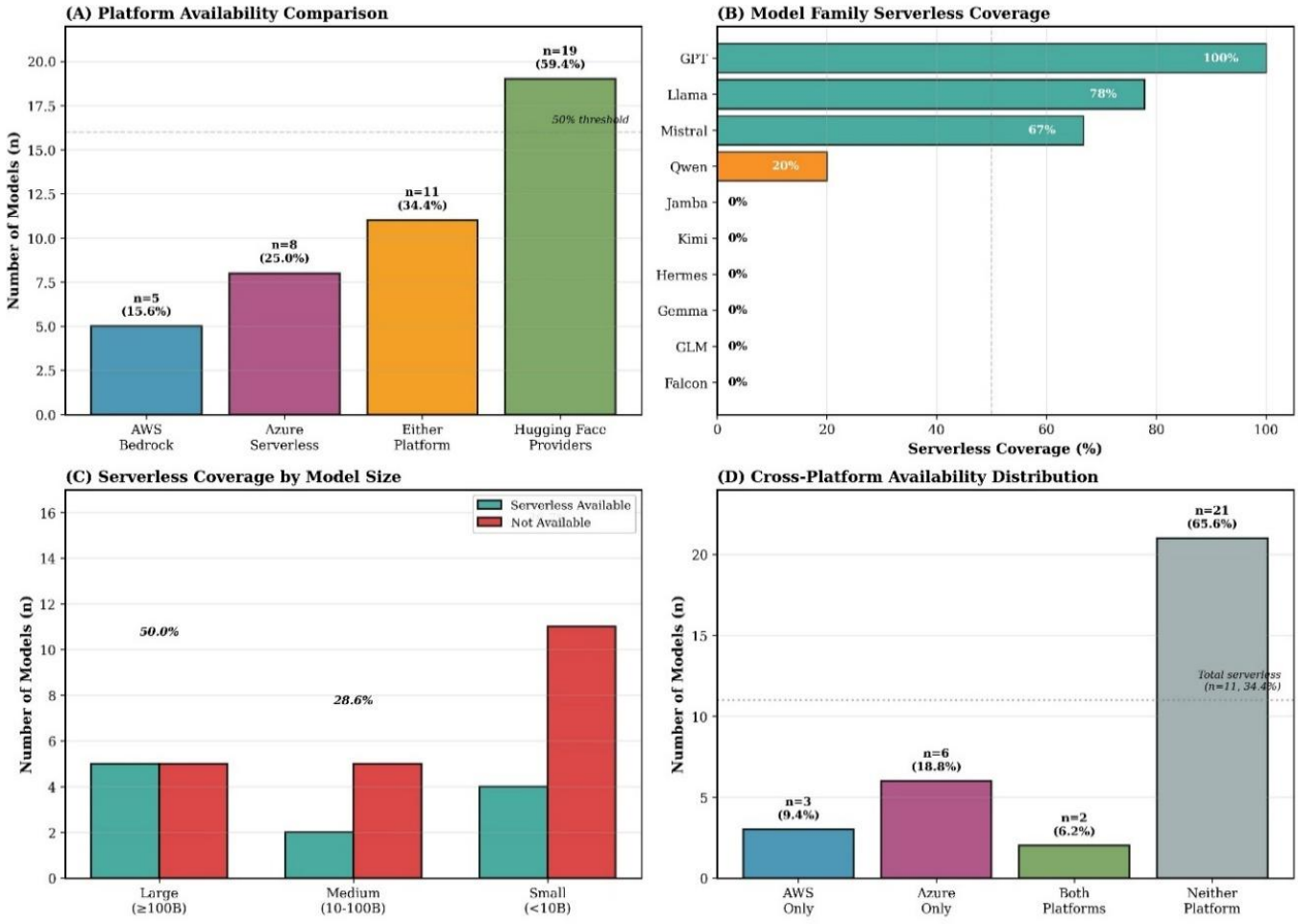

Figure 6. *Serverless endpoint availability across 32 LLMs. (A) Comparison of model counts across Amazon Bedrock, Azure Serverless, either serverless platform, or Hugging Face providers. (B) Serverless coverage percentage by model family, ordered from highest to lowest coverage. (C) Number of models available and not available on serverless platforms by size category. Percentages above bars indicate serverless coverage within each category. (D) Distribution of models by cross-platform availability status (AWS only, Azure only, both platforms, neither platform).*

Large models had 5 of 10 models (50.0%) available on serverless platforms, with 3 models on Amazon Bedrock and 3 models on Azure Serverless. Medium models had 2 of 7 models (28.6%) available, with 2 models on Amazon

Bedrock and 1 model on Azure Serverless. Small models had 4 of 15 models (26.7%) available, with 0 models on Amazon Bedrock and 4 models on Azure Serverless. Small models comprised 46.9% of the total models examined but represented 36.4% of the serverless-available models (4 of 11). Large models comprised 31.3% of total models and represented 45.5% of serverless-available models (5 of 11). Medium models comprised 21.9% of total models and represented 18.2% of serverless-available models (2 of 11). Figure 6 summarizes the finding on serverless availability.

Table 10. *Serverless availability by model size category.*

Size Category	n	AWS	Azure	Either	Coverage (%)
Large (≥100B)	10	3	3	5	50.0
Medium (10-100B)	7	2	1	2	28.6
Small (<10B)	15	0	4	4	26.7

4.3 Metrics and data insights related to architecture

To comprehensively evaluate the operational characteristics and observability capabilities of different LLM deployment architectures, we systematically analyzed the metrics exposed by serverless (Amazon Bedrock) and BYOC (Bring Your Own Container) approaches exemplified by Amazon SageMaker. Our analysis categorizes metrics across five dimensions: latency, throughput, resource utilization, cost, and reliability, revealing fundamental trade-offs between deployment flexibility and operational visibility.

4.3.1 Deployment architecture metrics comparison

Table 11 presents a comprehensive comparison of 28 metrics across both deployment architectures. The analysis reveals a critical asymmetry: while SageMaker exposes 42+ built-in metrics with full infrastructure visibility, Bedrock's serverless abstraction exposes approximately 12 core metrics, prioritizing simplicity over granular control. This trade-off reflects the fundamental architectural difference between managed infrastructure (BYOC) and fully abstracted serverless platforms.

Table 11. *Comparative analysis of metrics availability in serverless (Bedrock) and BYOC (SageMaker) deployment architectures.*

Category	Metric	Description	SageMaker	Bedrock	Possible related research scope for metric
Latency	time to first token (TTFT)	Time from request to first output token generation	FirstChunkLatency	API response only	Leviathan et al. (2023); Agrawal et al. (2024a)
	model latency	Model container processing time excluding infrastructure overhead	ModelLatency (μs)	InvocationLatency (ms)	Dao et al. (2022)
	overhead latency	Routing, authentication, and preprocessing time	OverheadLatency	Not available	Yu et al. (2022)
	inter-token latency (ITL)	Average time between consecutive token generations	Custom required	Not available	Agrawal et al. (2024b)
	end-to-end latency	Total time from request submission to final token	ModelLatency + Overhead	InvocationLatency	Kwon et al. (2023)
	P50/P95/P99 latencies	Percentile latencies for SLO compliance	Built-in statistics	Calculated	Zhang & Matta (2025)
	Cold start time	Time to launch compute and load model	ModelSetupTime	Not exposed	Fu et al. (2024)

Category	Metric	Description	SageMaker	Bedrock	Possible related research scope for metric
Throughput	Invocations	Total request count	Invocations	Invocations	Romero et al. (2022)
	Tokens per second (TPS)	Output token generation rate	Custom required	Derived from OutputTokenCount	Zheng et al. (2024)
	Input/Output token count	Tokens processed/generated per request	Custom logging	InputTokenCount, OutputTokenCount	Kwon et al. (2023)
	Concurrent invocations	Simultaneous requests being processed	ConcurrentRequestsPerModel	Not exposed	Agrawal et al. (2024b)
	Invocations per Instance	Normalized load for autoscaling	InvocationsPerInstance	N/A (serverless)	Fu et al. (2024)
	Queue backlog	Pending requests awaiting processing	ApproximateBacklogSize	BatchRecordsPending	Fu et al. (2024)
Resource	GPU utilization	Percentage of GPU compute cycles used	GPUUtilization (0-400%)	Not available	Dao (2024)
	GPU memory utilization	VRAM consumption for KV cache	GPUMemoryUtilization	Not available	Kwon et al. (2023)
	CPU/memory utilization	Host CPU and RAM usage	CPUUtilization, MemoryUtilization	Not available	Li et al. (2024b)
	KV cache utilization	Attention cache memory usage	Custom (vLLM/TGI)	Not available	Zhou et al. (2024)
Cost	Cost per token	Direct cost for token processing	Derived: instance-hrs/tokens	Token count × price	Various industry analyses (2024–2025), i.e. industry white papers and technical reports from cloud providers (AWS, Azure, GCP).
	Cost per request	Total cost for single inference request	Derived: instance-hrs/invocations	(input×price)+(output×price)	Zhang & Matta (2025)
	Idle Cost	Cost during zero-utilization periods	Full instance cost	Zero (on-demand)	Fu et al. (2024)
Reliability	Client errors (4XX)	Request validation and auth failures	Invocation4XXErrors	InvocationClientErrors	Romero et al. (2022)
	Server errors (5XX)	Infrastructure and model failures	Invocation5XXErrors	InvocationServerErrors	Fu et al. (2024)
	Mid-stream errors	Failures during streaming response	MidStreamErrors	Not available	Agrawal et al. (2024b)
	Throttling	Rate limit exceeded events	Via error codes	InvocationThrottles	Zhang & Matta (2025)
Model	Prefill/decode latency	Time for prompt processing vs token generation	Custom logging	Not available	Agrawal et al. (2024a)
	Context window utilization	Percentage of max context used	Custom calculation	Derived	—
	Model cache hit rate	Cache efficiency for multi-model endpoints	ModelCacheHit	N/A	Cheng et al. (2025)
	Model loading time	Time to load model into memory	ModelLoadingTime	Not available (always hot)	Fu et al. (2024)

Note: SageMaker provides 42+ built-in metrics with full infrastructure visibility. Bedrock exposes approximately 12 core metrics with serverless abstraction. "Custom required" indicates metrics available via custom instrumentation (e.g., vLLM, TGI). "Not available" indicates metrics not exposed due to platform architecture constraints.

The analysis reveals several critical findings. SageMaker provides granular latency decomposition, enabling precise identification of bottlenecks through metrics like OverheadLatency (infrastructure routing time) and ModelLatency (pure inference time). In contrast, Bedrock consolidates these into a single InvocationLatency metric, sacrificing

diagnostic granularity for operational simplicity. The most pronounced difference emerges in resource utilization metrics, where SageMaker exposes comprehensive GPU metrics including utilization (0–400% for multi-GPU instances) and memory consumption, while Bedrock's serverless abstraction completely obscures infrastructure metrics. Both platforms enable cost-per-token calculations through different mechanisms: SageMaker requires deriving costs from instance-hours and throughput metrics, while Bedrock provides direct token-level pricing transparency. Critically, SageMaker users incur idle costs during low-utilization periods, whereas Bedrock's on-demand model eliminates this waste. Several critical metrics for LLM optimization—including Tokens Per Second (TPS), prefill/decode latency separation, and KV cache utilization—require custom logging in both platforms, suggesting that current cloud provider abstractions have not yet converged on standardized metrics for LLM-specific performance characteristics.

Table 12 provides a consolidated summary of the key architectural differences between serverless and BYOC deployment approaches across the six operational dimensions examined in this study, serving as a practical decision-support reference for researchers selecting cloud infrastructure for LLM-based workloads.

Table 12. *Comparative summary of serverless and BYOC deployment architectures across six operational dimensions.*

Dimension	Serverless (Bedrock, Azure Serverless)	BYOC (Amazon SageMaker)
Cost model	Per-token, no idle cost	Per-hour, idle cost applies
Scaling	Automatic, scale-to-zero	Manual or custom autoscaling
Availability	34.4% of examined models	All 32 examined models
Observability	~12 core metrics	42+ built-in metrics
GPU metrics	Not exposed	Full visibility (utilization, memory)
Cross-platform	6.2% cross-platform redundancy	Provider-independent deployment
Management	None (fully abstracted)	Full infrastructure responsibility

5 DISCUSSION

This study examined the performance, availability, and operational characteristics of LLMs in cloud computing environments through three complementary analyses. Our findings reveal fundamental scaling laws governing LLM performance and cost, significant disparities in model accessibility across cloud platforms, and critical trade-offs between deployment architectures that inform both theoretical understanding and practical implementation strategies.

5.1 Synthesis of key findings

The empirical analysis of 32 LLMs reveals three central insights that characterize cloud-based deployment. First, model performance follows a logarithmic relationship with both parameter count and deployment cost, indicating diminishing returns as model size increases. This challenges the assumption that larger models provide proportional performance gains and underscores the importance of efficiency-oriented model selection.

Second, serverless LLM availability remains uneven across platforms, limiting deployment flexibility and increasing the risk of vendor lock-in. This fragmentation is particularly pronounced across model families and parameter ranges, where access to serverless endpoints is inconsistent.

Third, deployment architecture introduces a fundamental trade-off between operational simplicity and observability. Serverless platforms reduce infrastructure complexity and eliminate idle costs, but at the expense of reduced visibility into system-level performance, constraining opportunities for optimization.

5.2 Theoretical implications of scaling laws

The logarithmic relationship between model parameters and accuracy (Accuracy = $0.329 + 0.126 \times \log_{10}(\text{Parameters})$) represents a critical empirical validation of neural scaling laws in production deployment contexts. This finding extends theoretical work on scaling behavior by demonstrating that the diminishing returns predicted by Kaplan et al. (2020) manifest not only in training dynamics but also in deployed model performance on discriminative tasks. The 71% reduction in marginal efficiency when transitioning from the 0.6-3B to 3-10B parameter range, escalating to 99% reduction in the 150B-1000B range, quantifies the severity of these diminishing returns and suggests a performance plateau that contradicts the prevailing industry emphasis on ever-larger models.

The moderately weaker predictive power of cost-based models ($R^2 = 0.639$) compared to parameter-based models ($R^2 = 0.727$) reveals that pricing strategies do not perfectly align with computational requirements. This discrepancy likely reflects competitive market dynamics, vendor-specific optimizations, and the incorporation of non-computational factors such as support costs and profit margins into pricing structures. The implications for researchers are significant: parameter count serves as a more reliable predictor of performance than deployment cost, suggesting that cost-performance optimization requires explicit analysis rather than reliance on market pricing as a proxy for model capability.

5.3 Practical implications for LLM deployment

For practitioners deploying LLMs in production environments, our findings yield several actionable recommendations. Organizations should prioritize mid-sized models for applications requiring high absolute accuracy, as they capture most achievable performance while maintaining reasonable cost efficiency. The identification of Llama-3.2-3B as the most cost-efficient model (0.528 accuracy per dollar) suggests that smaller models merit consideration for applications tolerant of moderate accuracy reductions, potentially achieving 98-fold cost improvements over trillion-parameter alternatives. Deployment architecture selection should explicitly account for the operational simplicity versus diagnostic depth trade-off: serverless platforms suit rapid deployment and automatic scaling requirements, while BYOC approaches enable fine-grained performance optimization and cost attribution in multi-tenant environments.

The observed fragmentation in serverless availability creates strategic planning challenges for organizations seeking vendor independence. The minimal cross-platform redundancy indicates that multi-cloud strategies cannot simply replicate deployments across providers but must instead implement provider-specific integrations. Organizations requiring specific model families—particularly Mistral (66.7% serverless coverage), Qwen (20.0%), and Falcon (0%)—face constrained deployment options that may necessitate BYOC implementations despite preferring serverless simplicity. The complete absence of small models (< 10B parameters) from serverless platforms forces organizations deploying cost-efficient applications to adopt infrastructure management responsibilities that serverless architectures aim to eliminate.

5.4 Observability and operational maturity

The 71% reduction in exposed metrics when transitioning from BYOC to serverless architectures (42+ metrics versus ~12 metrics) reflects an immature standardization landscape for LLM-specific observability. Critical metrics for production optimization—including inter-token latency, prefill/decode latency separation, KV cache utilization, and tokens per second—require custom instrumentation in both deployment models, indicating that cloud providers have not yet converged on standardized abstractions for LLM performance characteristics. This gap mirrors early cloud computing challenges where infrastructure providers gradually developed standardized metrics for conventional workloads. The absence of GPU utilization and memory consumption metrics in serverless platforms eliminates crucial optimization levers, forcing organizations to choose between operational simplicity and performance tuning capability. Future platform evolution must address this observability gap, potentially through standardized LLM performance APIs that expose essential metrics while preserving serverless abstraction benefits.

5.5 Limitations and boundary conditions

Several limitations constrain the generalizability of our findings. First, the performance evaluation employed a single benchmark dataset (Belebele) focused on discriminative question-answering tasks, which may not generalize to

generative, reasoning-intensive, or domain-specific applications where model scaling behaviors could differ substantially. Second, the analysis captured a temporal snapshot of cloud platform offerings as of early 2025, while rapid platform evolution and continuous model releases may alter availability patterns and pricing structures within months. Third, the study focused exclusively on inference deployment, excluding training costs and fine-tuning considerations that significantly influence total cost of ownership for organizations developing custom models. Fourth, the metrics analysis examined only two cloud providers (AWS and Azure), potentially missing platform-specific optimizations or metric exposures from Google Cloud Platform, Oracle Cloud Infrastructure, or specialized AI platform providers. Fifth, the cost analysis utilized list pricing without accounting for volume discounts, reserved capacity pricing, or enterprise agreements that substantially modify cost structures for large-scale deployments.

5.6 Future research directions

The findings suggest several promising research directions. First, longitudinal studies tracking scaling law evolution across model generations would clarify whether diminishing returns represent fundamental computational limits or artifacts of current architectures potentially addressable through architectural innovations. Second, comparative analysis across diverse task categories (generative, reasoning, domain-specific) would establish task-dependent scaling behaviors and identify applications where smaller models achieve competitive performance. Third, investigation of hybrid deployment strategies combining serverless and BYOC approaches could identify optimal architectures for organizations requiring both operational simplicity and performance optimization capabilities. Fourth, development of standardized LLM observability frameworks would accelerate platform maturation and enable cross-provider performance comparisons essential for informed deployment decisions. Fifth, analysis of fine-tuning cost structures and their interaction with base model selection would provide comprehensive total cost of ownership models guiding build-versus-buy decisions. Sixth, examination of emerging quantization and compression techniques (e.g., GPTQ, AWQ) would quantify their impact on the accuracy-to-cost ratio and potentially shift sweet spot identification toward smaller effective parameter counts.

5.7 Implications for cloud platform evolution

Cloud providers face strategic decisions regarding platform design based on observed market dynamics. The serverless availability gap for small models (0% coverage for <10B parameters) represents a market opportunity for providers targeting cost-conscious deployments, as our findings demonstrate that 3-10B models deliver substantial cost efficiency improvements. The metric exposure asymmetry between serverless and BYOC platforms suggests competitive differentiation opportunities through enhanced observability APIs that preserve serverless simplicity while exposing critical performance metrics. Platform providers could address vendor lock-in concerns by expanding cross-platform model availability, currently limited to 6.2% of serverless-accessible models. The identification of the 30-50B parameter sweet spot indicates that platform optimization efforts should prioritize this range rather than focusing exclusively on trillion-parameter frontier models. Finally, standardization efforts around LLM-specific metrics—potentially coordinated through industry consortia or open standards bodies—would reduce fragmentation and accelerate platform maturity, benefiting both providers and users through improved interoperability and performance transparency.

6 CONCLUSION

This study provides an empirical evaluation of cloud-based LLM deployment strategies across serverless and BYOC architectures. The results demonstrate that model performance scales logarithmically with size and cost, leading to diminishing returns beyond mid-sized models. These findings challenge the prevailing emphasis on ever-larger models and highlight the importance of efficiency-driven deployment decisions.

Serverless platforms offer operational simplicity and cost advantages for certain workloads, but remain constrained by limited model availability and reduced observability. These limitations restrict deployment flexibility and reduce opportunities for detailed performance optimization.

From a practical perspective, the study provides clear guidance for infrastructure selection. Smaller models offer superior cost efficiency, while mid-sized models provide a strong balance between performance and cost for more

demanding applications. Deployment architecture should therefore be selected based on the trade-off between simplicity and control, aligned with workload requirements.

Overall, the findings emphasize that effective LLM deployment is not solely a function of model scale, but of informed trade-offs between performance, cost, and operational constraints.

ADDITIONAL INFORMATION AND DECLARATIONS

Conflict of Interests: The author declares no conflict of interest.

Author Contributions: The author confirms being the sole contributor of this work.

Statement on the Use of Artificial Intelligence Tools: The author declares that he didn't use artificial intelligence tools for text or other media generation in this article.

Data Availability Statement: The evaluation scripts, benchmark prompts, and aggregated results supporting this study are publicly available at https://github.com/matpl2/Cloud-Based-Large-Language-Model-Deployment.

REFERENCES

Agrawal, A., Kedia, N., Panwar, A., Mohan, J., Kwatra, N., Gulavani, B. S., Tumanov, A., & Ramjee, R. (2024a). Taming throughput-latency tradeoff in LLM inference with Sarathi-Serve. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation* (pp. 117–134). USENIX Association.

Agrawal, A., Panwar, A., Mohan, J., Kwatra, N., Gulavani, B. S., & Ramjee, R. (2024b). Sarathi-Serve: Efficient LLM inference by piggybacking decodes with chunked prefills. arXiv. https://doi.org/10.48550/arXiv.2403.02310

Appenzeller, M. (2024). Welcome to LLMflation - LLM inference cost is going down fast. Andreessen Horowitz. https://a16z.com/llmflation-llm-inference-cost

Bandarkar, L., Liang, D., Muller, B., Artetxe, M., Shukla, S. N., Husa, D., Goyal, N., Krishnan, A., Zettlemoyer, L., & Khabsa, M. (2024). The Belebele benchmark: A parallel reading comprehension dataset in 122 language variants. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics* (pp. 749–775). ACL. https://doi.org/10.18653/v1/2024.acl-long.44

Cheng, Y., Liu, Y., Yao, J., An, Y., Chen, X., Feng, S., Huang, Y., Shen, S., Zhang, R., Du, K., & Jiang, J. (2025). LMCache: An efficient KV cache layer for enterprise-scale LLM inference. arXiv. https://doi.org/10.48550/arXiv.2510.09665

Collabnix. (2024). Kubernetes autoscaling for LLM inference: Complete guide (2024). https://collabnix.com/kubernetes-autoscaling-for-llm-inference-complete-guide-2024/

Dao, T. (2024). FlashAttention-2: Faster attention with better parallelism and work partitioning. In *12th International Conference on Learning Representations*. ICLR.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., & Ré, C. (2022). FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems* (pp. 16344–16359). NeurIPS.

Xiang, Y., Li, X., Qian, K., Yang, Y., Zhu, D., Yu, W., Zhai, E., Liu, X., Jin, X., & Zhou, J. (2025). Aegaeon: Effective GPU pooling for concurrent LLM serving on the market. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles* (pp. 1030–1045). ACM. https://doi.org/10.1145/3731569.3764815

Frantar, E., Ashkboos, S., Hoefler, T., & Alistarh, D. (2023). GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *11th International Conference on Learning Representations*. ICLR.

FriendliAI. (2024). Which quantization to use to reduce the size of LLMs? https://friendli.ai/blog/quantization-reduce-llm-size

Fu, Y., Xue, L., Huang, Y., Brabete, A.-O., Ustiugov, D., Patel, Y., & Mai, L. (2024). ServerlessLLM: Low-latency serverless inference for large language models. In *18th USENIX Symposium on Operating Systems Design and Implementation* (pp. 135–153). USENIX Association.

GMI Cloud. (2025). How much do GPU cloud platforms cost for AI startups in 2025. https://www.gmicloud.ai/blog/how-much-do-gpu-cloud-platforms-cost-for-ai-startups-in-2025

Gun.io. (2025). Scaling AI infrastructure for LLMs: Best practices for mid-sized companies. https://gun.io/news/2025/04/scaling-ai-infrastructure-for-llms/

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., & Amodei, D. (2020). Scaling laws for neural language models. arXiv. https://doi.org/10.48550/arXiv.2001.08361

Cheng, X., Zhang, Z., Zhou, Y., Ji, J., Jiang, J., Zhao, Z., Xiao, Z., Ye, Z., Huang, Y., Lai, R., Jin, H., Hou, B., Wu, M., Dong, Y., Yip, A., Wang, S., Yang, W., Miao, X., Chen, T., & Jia, Z. (2025). Mirage persistent kernel: A compiler and runtime for mega-kernelizing tensor programs. arXiv. https://doi.org/10.48550/arXiv.2512.22219

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., & Stoica, I. (2023). Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles,* (pp. 611–626). ACM. https://doi.org/10.1145/3600006.3613165

Lai, R., Liu, H., Lu, C., Liu, Z., Cao, S., Shao, S., Zhang, Y., Mai, L., & Ustiugov, D. (2025). TokenScale: Timely and accurate autoscaling for disaggregated LLM serving with token velocity. arXiv. https://doi.org/10.48550/arXiv.2512.03416

Leviathan, Y., Kalman, M., & Matias, Y. (2023). Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning* (pp. 19274–19286). PMLR.

Li, L., Dinh, L., Hu, S., & Hemphill, L. (2024a). Academic collaboration on large language model studies increases overall but varies across disciplines. arXiv. https://doi.org/10.48550/arXiv.2408.04163

Li, B., Jiang, Y., Gadepally, V., & Tiwari, D. (2024b). LLM inference serving: Survey of recent advances and opportunities. In *2024 IEEE High Performance Extreme Computing Conference.* IEEE. https://doi.org/10.1109/HPEC62836.2024.10938426

Lin, J., Tang, J., Tang, H., Yang, S., Dang, X., & Han, S. (2025). AWQ: Activation-aware weight quantization for LLM compression and acceleration. *GetMobile: Mobile Computing and Communications*, 28(4), 12–17. https://doi.org/10.1145/3714983.371498

Lin, Y., Peng, S., Lu, C., Xu, C., & Ye, K. (2026). FlexPipe: Adapting dynamic LLM serving through inflight pipeline refactoring in fragmented serverless clusters. In *Proceedings of the 21st European Conference on Computer Systems*. ACM. https://doi.org/10.1145/3767295.3769316

Meta AI. (2024). Llama 3.3 70B Instruct. Hugging Face. https://huggingface.co/meta-llama/Llama-3.3-70B-Instruct

Microsoft Azure. (2023). Cost-effective private large language model inference on Azure Kubernetes Service. https://msazure.club/cost-effective-private-large-language-model-inference-on-azure-kubernetes-service/

NVIDIA. (2023). NVIDIA TensorRT-LLM supercharges large language model inference on NVIDIA H100 GPUs. NVIDIA Technical Blog. https://developer.nvidia.com/blog/nvidia-tensorrt-llm-supercharges-large-language-model-inference-on-nvidia-h100-gpus

NVIDIA. (2025). NVIDIA TensorRT-LLM now supports recurrent drafting for optimizing LLM inference. NVIDIA Technical Blog. https://developer.nvidia.com/blog/nvidia-tensorrt-llm-now-supports-recurrent-drafting-for-optimizing-llm-inference/

Oracle. (2024). Achieve cost-efficient LLM serving with production-ready quantization solution. Oracle Cloud Infrastructure Blog. https://blogs.oracle.com/cloud-infrastructure/cost-efficient-llm-serving-with-quantization

Paleyes, A., Urma, R.-G., & Lawrence, N. D. (2022). Challenges in deploying machine learning: A survey of case studies. *ACM Computing Surveys*, 55(6), Article 124. https://doi.org/10.1145/3533378

Romero, F., Souza, M., Watcharapichat, P., Zhang, Q., Zhao, N. J., & Li, A. (2022). INFless: A native serverless system for low-latency, high-throughput inference. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (pp. 768–781). ACM.

Saleh, Y., Abu Talib, M., Nasir, Q., & Dakalbab, F. (2025). Evaluating large language models: A systematic review of efficiency, applications, and future directions. *Frontiers in Computer Science*, 7, Article 1523699. https://doi.org/10.3389/fcomp.2025.1523699

Schmid, L., Hey, T., Armbruster, M., Corallo, S., Fuchß, D., Keim, J., Liu, H., & Koziolek, A. (2025). Software architecture meets LLMs: A systematic literature review. arXiv. https://doi.org/10.48550/arXiv.2505.16697

Semerikov, S. O., Vakaliuk, T. A., Kanevska, O. B., Ostroushko, O. A., & Kolhatin, A. O. (2025). Edge intelligence unleashed: A survey on deploying large language models in resource-constrained environments. *Journal of Edge Computing*, 4(2), 179–233. https://doi.org/10.55056/jec.1000

Xiao, C., Cai, J., Zhao, W., Zeng, G., Lin, B., Zhou, J., Zheng, Z., Han, X., Liu, Z., & Sun, M. (2025). Densing law of LLMs. *Nature Machine Intelligence*, 7, 1823–1833. https://doi.org/10.1038/s42256-025-01137-0

Xu, M., Liao, J., Wu, J., He, Y., Ye, K., & Xu, C. (2025). Cloud native system for LLM inference serving. arXiv. https://doi.org/10.48550/arXiv.2507.18007

Yadav, R. (2025). Deploying scalable serverless LLM workloads on Kubernetes + Knative + GPU support + validating admission webhook. Medium. https://medium.com/@daydreamingguy941/deploying-scalable-serverless-llm-workloads-on-kubernetes-knative-gpu-support-validating-1e7cab5b0cf8

Yu, G.-I., Jeong, J. S., Kim, G.-W., Kim, S., & Chun, B.-G. (2022). Orca: A distributed serving system for transformer-based generative models. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation* (pp. 521–538). USENIX Association.

Zhang, Y., & Matta, I. (2025). SERFLOW: Serverless LLM serving with cost-efficient autoscaling. In *Proceedings of the 8th International Workshop on Edge Systems, Analytics and Networking* (pp. 25–30). ACM. https://doi.org/10.1145/3721323.3724838

Zheng, L., Yin, L., Xie, Z., Huang, J., Sun, C., Yu, C. H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J. E., Barrett, C., & Sheng, Y. (2024). SGLang: Efficient execution of structured language model programs. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, (pp. 62557–62583). ACM.

Zhou, Z., Ning, X., Hong, K., Fu, T., Xu, J., Li, S., Lou, Y., Wang, L., Yuan, Z., Li, X., Yan, S., Dai, G., Zhang, X.-P., Dong, Y., & Wang, Y. (2024). A survey on efficient inference for large language models. arXiv. https://doi.org/10.48550/arXiv.2404.14294

Acta Informatica Pragensia is published by the Prague University of Economics and Business, Czech Republic | eISSN: 1805-4951