http://doi.org/10.35784/iapgos.8206 *received: 07.08.2025 | revised: 04.09.2025 | accepted: 18.11.2025 | available online: 20.12.2025*

EVALUATION OF THE PERFORMANCE OF LLMs DEPLOYMENTS IN SELECTED CLOUD-BASED CONTAINER SERVICES

Mateusz Stęgierski, Piotr Szpak, Sławomir Przyłucki
Lublin University of Technology, Department of Computer Science, Lublin, Poland

Abstract. *The growing adoption of serverless container services has created challenges in selecting optimal cloud platforms for production LLM deployments, yet comparative performance evaluations remain limited. This study evaluates AWS Fargate and Azure Container Apps for LLM deployments, investigating whether architectural differences cause substantial performance variations under diverse load patterns. We conducted systematic experiments using containerized Llama 3.2:1b across multiple scenarios: baseline measurements, inference tests with varying prompt lengths, streaming API performance, and concurrent load testing with progressive scaling. Each scenario was executed on both standard and auto-scaled infrastructure with 10 runs per configuration to ensure statistical reliability. Key findings reveal distinct platform characteristics: AWS Fargate demonstrates superior baseline API response times and time-to-first-token performance, while Azure Container Apps consistently outperforms AWS in inference processing for short and medium prompts with better consistency across test runs. Streaming performance shows platform-specific trade-offs, with AWS achieving lower initial latency but Azure providing superior token generation consistency. Under concurrent loads, both platforms maintain full capacity at lower concurrency levels, but AWS exhibits exponential response time degradation at higher loads while Azure shows more linear, predictable scaling behavior. Statistical analysis confirms significant performance differences across all metrics, validating that platform architecture fundamentally impacts LLM deployment performance. These findings indicate platform selection should align with specific workload requirements: AWS Fargate for latency-critical applications with steady loads, and Azure Container Apps for inference-intensive workloads requiring robust scaling and consistency. This study offers crucial benchmarking data for businesses deploying production-grade AI services on serverless container platforms.*

Keywords: serverless containers, performance comparison, language models, auto-scaling, load testing

OCENA WYDAJNOŚCI WDROŻEŃ LLM W WYBRANYCH USŁUGACH KONTENEROWYCH OPARTYCH NA CHMURZE

Streszczenie. *Rosnąca popularność bezserwerowych usług kontenerowych stworzyła wyzwania związane z wyborem optymalnych platform chmurowych dla wdrożeń produkcyjnych LLM, jednak porównawcze oceny wydajności pozostają ograniczone. Niniejsze badanie ocenia AWS Fargate i Azure Container Apps pod kątem wdrożeń LLM, badając, czy różnice architektoniczne powodują znaczne różnice w wydajności przy różnych wzorcach obciążenia. Przeprowadziliśmy systematyczne eksperymenty przy użyciu skonteneryzowanej Llamy 3.2:1b w wielu scenariuszach: pomiary bazowe, testy wnioskowania z różnymi długościami podpowiedzi, wydajność API strumieniowego i jednoczesne testy obciążenia z progresywnym skalowaniem. Każdy scenariusz został wykonany zarówno na standardowej, jak i automatycznie skalowanej infrastrukturze z 10 przebiegami na konfigurację, aby zapewnić wiarygodność statystyczną. Kluczowe wnioski ujawniają wyraźne cechy platformy: AWS Fargate wykazuje lepsze bazowe czasy odpowiedzi API i wydajność time-to-first-token, podczas gdy Azure Container Apps konsekwentnie przewyższa AWS w przetwarzaniu wnioskowania dla krótkich i średnich monitów z lepszą spójnością we wszystkich przebiegach testowych. Wydajność przesyłania strumieniowego pokazuje kompromisy specyficzne dla platformy, przy czym AWS osiąga niższe opóźnienie początkowe, ale Azure zapewnia lepszą spójność generowania tokenów. Przy jednoczesnym obciążeniu obie platformy utrzymują pełną wydajność przy niższych poziomach współbieżności, ale AWS wykazuje wykładnicze pogorszenie czasu odpowiedzi przy wyższych obciążeniach, podczas gdy Azure wykazuje bardziej liniowe, przewidywalne zachowanie skalowania. Analiza statystyczna potwierdza znaczące różnice w wydajności we wszystkich metrykach, potwierdzając, że architektura platformy ma zasadniczy wpływ na wydajność wdrażania LLM. Wyniki te wskazują, że wybór platformy powinien być dostosowany do konkretnych wymagań dotyczących obciążenia: AWS Fargate dla aplikacji o krytycznym opóźnieniu i stałym obciążeniu oraz Azure Container Apps dla obciążeń intensywnie wykorzystujących wnioskowanie, wymagających solidnego skalowania i spójności. Badanie to oferuje kluczowe dane porównawcze dla firm wdrażających usługi AI klasy produkcyjnej na bezserwerowych platformach kontenerowych.*

Słowa kluczowe: kontenery bezserwerowe, porównanie wydajności, modele językowe, automatyczne skalowanie, testowanie obciążenia

Introduction

The rapid growth of cloud services has fundamentally changed the way modern IT solutions are designed, developed, and implemented. Cloud infrastructure has become the dominant platform for implementing new technological ideas, offering scalability, accessibility, and cost-effectiveness that traditional on-premises solutions cannot match. This paradigm shift enables organizations of all sizes to use enterprise-grade computing resources without managing complex physical infrastructure.

The transition from conceptual ideas to production-ready cloud deployments has been significantly facilitated by the adoption of versatile containerization technologies. Container-based virtualization provides a standardized approach to packaging applications and their dependencies, creating portable, consistent deployment units that run reliably across diverse environments. This technology bridges the gap between individual development environments and large-scale cloud infrastructure, enabling seamless application deployment regardless of underlying platform complexity.

Major cloud providers have developed comprehensive container-as-a-service offerings in recognition of the critical importance of containerization in modern software deployment. AWS (Amazon Web Services), Microsoft Azure, Google Cloud Platform, and other leading providers offer multiple solutions, ranging from basic container hosting to sophisticated orchestration platforms. This proliferation of options gives developers and research teams unprecedented flexibility in choosing deployment environments that best suit their needs.

The abundance of available container services naturally raises the fundamental question of how to select the optimal platform. It is not straightforward to determine which service provides the best performance, cost-effectiveness, and operational characteristics for a particular project, as this decision depends heavily on the specific nature and requirements of the application being deployed. This decision becomes even more complex when considering the unique computational demands of emerging technologies.

Recent years have witnessed an explosive growth in interest and adoption of Large Language Model (LLM) based systems. These AI (Artificial Inteligence) models, exemplified by systems like GPT, Claude, and Llama, have revolutionized natural language processing and generation capabilities across numerous industries. Due to their substantial computational requirements, including high memory usage for model loading, intensive CPU or GPU processing for inference operations, and significant network bandwidth needs for handling concurrent user requests, LLM-based applications present distinctive deployment challenges. These models also often require specialized runtime environments, careful resource allocation strategies, and optimized serving architectures to achieve acceptable response times and throughput.

Given the strategic importance of LLM deployment decisions and the complexity of the cloud container service landscape, there is a clear need for empirical research to guide the selection of platforms for containerized LLM applications. This study aims to comprehensively evaluate and compare the performance of leading cloud container services for LLM deployment scenarios. It focuses on AWS Fargate and Azure Container Apps, which are representative of the most prominent serverless container platforms.

Several key factors influenced the selection of AWS Fargate and Azure Container Apps for this comparative analysis: their market leadership positions (AWS and Microsoft Azure hold approximately 30% and 21% of the cloud market share, respectively [19]); their serverless container approach, which eliminates infrastructure management overhead; their architectural maturity and widespread enterprise adoption; and their representation of different cloud provider philosophies and technical implementations.

This study will conduct systematic benchmarking across multiple operational scenarios relevant to LLM deployment. These scenarios include baseline performance characteristics, inference processing capabilities under varying prompt complexities, streaming response performance, concurrent load handling, and system recovery patterns. The research will examine response latency, throughput metrics, resource utilization efficiency, scaling behavior, and performance consistency across multiple test iterations.

Several key propositions guide the investigation: architectural differences between cloud container platforms result in measurable performance variations for identical LLM workloads; one platform will demonstrate greater consistency in performance metrics across multiple test runs, indicating superior reliability for production deployments; and platform architecture influences the stability of performance characteristics during sustained LLM operations. Through comprehensive empirical analysis, this study aims to offer evidence-based recommendations for selecting platforms for production LLMs while contributing valuable insights to the emerging field of AI infrastructure optimization.

1. Literature review

1.1. Characteristics and performance evaluation of software container-based cloud services

The development of container technologies in the cloud has led to the emergence of specialized services that manage infrastructure without the need for direct administration by users. Serverless computing significantly simplifies system administration tasks, eliminating the need for developers to manage servers [1]. The CaaS (Container as a Service) concept provides an intermediate solution between IaaS (Infrastructure as a Service) and PaaS (Platform as a Service) in the cloud computing stack, offering a balance between infrastructure control and platform simplicity [2].

AWS Fargate, introduced in 2017 as a compute engine for containers, enables deployment and management of containers without administering AWS EC2 (Elastic Compute Cloud) infrastructure [9]. The platform represents an evolution toward cluster abstraction, where the AWS ECS (Elastic Container Service) layer is hidden from the user. A competing solution in the form of Azure Container Apps offers similar capabilities with built-in orchestration and auto-scaling mechanisms, with an architecture based on a simplified interface that eliminates the need for manual cluster configuration [5].

Performance evaluation methodologies for container services require a systematic approach to objectively compare platforms. Micro-benchmarking focuses on measuring single characteristics of server functions, covering CPU, memory, disk and network performance [15]. Application benchmarking takes a broader perspective, measuring end-to-end response times on serverless components in real-world usage scenarios, as demonstrated using an image processing application written in Node.js [16].

Key performance metrics include a number of parameters important for evaluating container platforms. Response time is a fundamental performance metric that measures the time from sending a request to receiving a response, with studies showing significant variations between platforms – for example, Google Cloud Run demonstrated an average request latency of 50 milliseconds compared to AWS App Runner's 91 milliseconds [1]. Cold start latency measures the time it takes for a new container to start up, with serverless platforms experiencing this problem initially but stabilizing after a few minutes, as demonstrated in backoff tests where execution latencies showed clear patterns after 15-minute idle periods [8]. A distribution of response times, including the median and the 90th, 95th and 99th percentiles, provides a more complete picture of performance than calculating the average value, with comprehensive benchmarking studies measuring response time distribution across different load scenarios [13].

Throughput, defined as the number of responses received per unit time, should increase with the level of concurrency. Scaling tests assume a gradual increase in the number of concurrent queries, which when comparing AWS ECS with AWS Lambda revealed significant differences in scaling mechanisms [11]. CPU utilization is a key indicator when studying application stability and the impact of scaling on resource consumption, especially in the context of long-running operations.

A comparative analysis of existing studies reveals a significant lack of direct comparisons between AWS Fargate and Azure Container Apps. Available studies focus on related platforms or are limited to single cloud providers, leaving a gap in direct performance analysis of these two key container platforms.

1.2. Specifics of implementing LLM models in cloud environments

Deploying LLMs in cloud environments introduces unique computational and infrastructure requirements that differ significantly from traditional web applications. These models are characterized by high memory requirements for model loading, CPU or GPU intensive processing for inference operations, and significant network bandwidth for handling concurrent user requests.

The performance challenges of deploying LLM in cloud environments include a number of technical aspects that require specialized approaches. Systems serving LLM have to deal with a large variance in processing times, where the length and complexity of prompts significantly affects response generation time [20]. Resource optimization for LLM inference in container environments requires consideration of specific memory and CPU usage patterns that differ from conventional web applications.

LLM application containerization offers a standardized approach to packaging models with their execution environment dependencies but introduces additional resource management challenges. The cold start problem is particularly relevant to LLM due to the long time it takes to load multi-gigabyte models into memory, which can significantly affect the initial latency of the system [9]. A systematic approach to minimizing cold start latency in the context of LLM requires advanced pre-warming and model caching techniques [4].

Scaling LLM in production environments introduces additional complexities associated with managing multiple model instances and load balancing between them [18]. Resource allocation strategies must take into account LLM load characteristics, which often exhibit significant variance depending on application usage patterns.

Optimizing infrastructure for LLM-based applications requires specialized approaches to managing memory and computing resources [17]. Interactive applications requiring progressive content generation (streaming) impose additional requirements on container platforms in terms of handling lengthy connections and efficient response caching.

Implementing LLM systems in the cloud also requires consideration of security and access management aspects of the models, further complicating the deployment architecture [6]. Containers offer resource isolation and access control but require appropriate configuration for the specific needs of LLM applications [14].

A review of the available literature reveals a significant research gap in the area of systematic performance analysis of container platforms for LLM workloads. Existing publications mainly focus on optimization of the models themselves, resource management in isolation or general aspects of cloud performance, while there is a lack of empirical comparative studies of the specifics of LLM performance on different container platforms under production conditions. This gap is particularly important from the perspective of organizations implementing LLM-based solutions, which require empirical data to make informed decisions about the choice of deployment platform for production systems supporting real user workloads.

2. Research methodology and scenarios

2.1. Research methodology

This study conducts a comparative analysis of AWS Fargate and Azure Container Apps as deployment platforms for LLMs. The investigation specifically examines performance characteristics when running identical Ollama instances across both environments. By maintaining strict equivalence in testing parameters, this research aims to isolate platform-specific performance attributes and provide actionable insights for organizations deploying LLM workloads in cloud environments.

The methodological approach utilizes scenario-based testing with precise temporal demarcation to establish clear correlations between test activities and resource utilization patterns. Each test scenario is executed independently with appropriate stabilization intervals between tests to prevent cross-contamination of results. Critically, to ensure statistical robustness, each scenario is executed multiple times (10 runs per configuration) to capture performance variability and enable comprehensive statistical analysis.

Data collection encompasses multiple dimensions of performance measurement including response latency, throughput capacity, token generation velocity, and resource utilization efficiency. The collection framework integrates directly with cloud-native monitoring services – AWS CloudWatch for Fargate deployments and Azure Monitor Software Development Kit (SDK) for Container Apps – providing high-fidelity resource utilization data throughout the testing process. This direct integration with platform metrics enables more sophisticated analysis than would be possible with application-level monitoring alone.

All performance data is captured in standardized JavaScript Object Notation (JSON) format with timestamps to facilitate temporal correlation across metrics sources. This provides complete transparency in the raw dataset while enabling sophisticated post-test analysis. An ensemble analysis approach is implemented through the python script, which aggregates data across multiple tests runs to generate statistical insights including:

- Mean, median, minimum, and maximum values for each performance metric.
- Standard deviation and CV (coefficient of variation) to quantify performance consistency.
- Statistical significance testing using Mann-Whitney U tests to verify platform differences.
- Visualization of performance distributions across test runs.

This ensemble approach provides significantly higher confidence in the findings by controlling for run-to-run variability and environmental fluctuations that might affect individual test executions.

2.2. Test scenarios

All of the presented test scenarios are executed on deployments of identical Ollama [10] instances with the llama3.2:1b [7] model loaded on both platforms: AWS Fargate and Azure Container Apps.

2.2.1. Scenario S0 – baseline performance

The baseline performance scenario establishes reference metrics for idle system behavior. During this phase, the testing framework collects 60 seconds of telemetry data while the container operates without active inference workloads. This provides critical baseline measurements for subsequent comparative analysis and helps isolate the impact of LLM inference activities from background system processes.

Measurement parameters:

- Baseline CPU utilization during idle state.
- Baseline memory allocation and consumption patterns.
- Application Programming Interface (API) responsiveness characteristics for basic health check operations.
- Background network activity patterns.

The baseline metrics establish a reference point against which subsequent test scenarios can be evaluated, enabling more precise quantification of the resource impact attributable to specific workload types.

2.2.2. Scenario S1 – basic inference performance

The basic inference scenario evaluates fundamental model performance characteristics under sequential request patterns. This testing phase employs prompts of varying complexity (short and medium) to assess how prompt length affects response generation across both platforms. Each request executes to completion before the subsequent request begins, preventing concurrent execution effects from influencing the results.

Measurement parameters:

- Total response time from request initiation to completion.
- Token generation velocity (tokens per second).
- Total token count produced per request.
- API success rate under normal operating conditions.
- CPU and memory utilization patterns during inference processing.
- Request-specific latency distribution statistics (mean, median, minimum, maximum).

This scenario provides insight into the base performance characteristics of each platform when processing standard, non-concurrent LLM inference requests. The collected metrics establish performance baselines for each prompt complexity level and serve as reference points for the subsequent concurrent testing scenarios.

2.2.3. Scenario S2 – streaming performance testing

The streaming performance scenario examines token-by-token response delivery capabilities, which are critical for interactive applications requiring progressive content generation. This test measures not only aggregate performance but also the user experience factors of initial latency and token delivery consistency.

Measurement parameters:

- Time to first token (TTFT) (initial response latency).
- Inter-token interval consistency (standard deviation).
- Total streaming duration from request to completion.
- Token delivery rate throughout the streaming process.
- Streaming API reliability under normal operating conditions.

- CPU and memory utilization patterns specific to streaming workloads.

Streaming performance metrics provide valuable insight into how each platform handles progressive content delivery, which directly impacts perceived responsiveness in interactive LLM applications. The consistency of token delivery intervals particularly affects the perceived quality of streaming text generation.

2.2.4. Scenario S3 – concurrent load testing

The concurrent load testing scenario evaluates performance characteristics under simultaneous user requests, providing critical insight into production scalability. This phase implements a gradual concurrency escalation pattern (1, 3, 5, 8 simultaneous requests) to identify how each platform responds to increasing demand.

The testing implementation incorporates several production-oriented reliability features:

- Exponential backoff retry logic with configurable attempt limits.
- Gradual request ramp-up to prevent artificial system overload,
- Extended timeout thresholds appropriate for LLM inference workloads.
- Comprehensive error trapping and categorization.

Measurement parameters:

- Response time degradation as concurrency increases.
- Request success rate across concurrency levels.
- System throughput (completed requests per second).
- Response time variability (standard deviation and 95th percentile).
- Resource utilization scaling patterns under concurrent load.
- Error distribution by category and concurrency level.

This scenario provides critical insights into each platform's request handling capabilities, queue management efficiency, and resource allocation strategies under concurrent workloads. The progressive increase in concurrency enables identification of performance inflection points where system behavior characteristics significantly change.

2.3. Cloud services configuration

To ensure a fair comparison, both AWS Fargate and Azure Container Apps were configured with equivalent parameters. For both platforms, containers were set with 1 vCPU (1024 units/1.0 cores) and 2 GiB (2048 MiB) of memory. Auto-scaling configurations on both AWS and Azure utilized CPU-based scaling, with a minimum capacity of 1 task/replica and a maximum of 5 tasks/replicas. The scale-up threshold was set at 70% CPU utilization with a 60-second cooldown, and the scale-down threshold at 30% CPU utilization with a 180-second cooldown. This consistent setup allowed for performance comparisons to be attributed directly to the platforms, rather than configuration differences, across both standard and auto-scaling infrastructure tests.

3. Results

3.1. Baseline performance test results

The baseline performance test (scenario S0) established reference metrics for system behavior under no load conditions. This scenario collected 60 seconds of telemetry data while the container operated without active inference workloads.

For standard infrastructure, analysis of baseline API response times revealed statistically significant differences between platforms ($p = 0.0002$). AWS Fargate demonstrated superior responsiveness with a median response time of 395.50 ms compared to Azure Container Apps at 588.37 ms, representing an 32.78% performance advantage for AWS.

For auto-scaling infrastructure, the baseline performance metrics maintained statistical significance ($p = 0.0004$) with AWS Fargate providing a median response time of 422.11 ms versus Azure Container Apps at 612.88 ms (31.13% advantage for AWS). Both platforms demonstrated excellent consistency in baseline measurements, though AWS showed notably better consistency with a coefficient of variation of 18.61% compared to Azure's 25.99%. The complete baseline performance metrics are summarized in Table 1.

Table 1. Baseline performance metrics summary

Platform	Infrastructure	API Response (ms)	CV (%)
AWS Fargate	Standard	395.50	8.25
Azure Container Apps	Standard	588.37	8.03
AWS Fargate	Auto-scaling	422.11	18.61
Azure Container Apps	Auto-scaling	612.88	25.99

These baseline measurements directly confirm that there are statistically significant variations in performance characteristics between cloud providers for identical workloads across both infrastructure types. The coefficient of variation for standard infrastructure measurements was notably low (AWS: 8.25%, Azure: 8.03%), providing initial insight into platform consistency differences.

3.2. Basic inference performance results

The basic inference scenarios (S1) evaluated model performance under sequential request patterns with varying prompt complexity, allowing each request to complete before initiating subsequent requests.

3.2.1. Short prompt inference results

For short prompt inference workloads in standard infrastructure, Azure Container Apps demonstrated a 27.47% performance advantage with a median response time of 64359.18 ms compared to AWS Fargate at 88740.50 ms. Statistical analysis revealed this difference was significant ($p = 0.0003$). Run-to-run consistency analysis showed AWS Fargate had a coefficient of variation of 10.24% compared to Azure's 6.72%, suggesting better consistency in the Azure environment for this workload type.

For the same inference workloads in auto-scaling infrastructure, Azure Container Apps showed a 24.86% advantage with a median response time of 61864.82 ms compared to AWS Fargate at 82330.44 ms. However, statistical analysis revealed this difference was not significant ($p = 0.0013$). Run-to-run consistency analysis showed Azure Container Apps had a remarkably better coefficient of variation of 8.04% compared to AWS's 15.34%, suggesting significantly better consistency in the Azure environment for this workload type in auto-scaling scenarios. All results are shown in Table 2.

Table 2. Short prompt inference performance statistics

Platform	Infrastructure	Response Time (ms)	Tokens per second	CV (%)
AWS Fargate	Standard	88740.50	3.19	10.24
Azure Container Apps	Standard	64359.18	4.14	6.72
AWS Fargate	Auto-scaling	82330.44	3.43	15.45
Azure Container Apps	Auto-scaling	61864.82	4.69	8.04

3.2.2. Medium prompt inference results

For medium-length prompts in standard infrastructure, Azure Container Apps again demonstrated a performance advantage with a median response time of 120183.95 ms versus Aws Fargate at 161486.51 ms (25.58% difference). This difference reached statistical significance ($p = 0.0002$). Both platforms exhibited similar consistency for this workload, with coefficients of variation of 8.14% for AWS and 9.70% for Azure (Fig. 1).

As for the auto-scaling infrastructure, Azure Container Apps demonstrated a substantial 23.51% performance advantage with

a median response time of 110997.95 ms versus AWS Fargate at 145122.27 ms. This difference also reached statistical significance (p = 0.0013). This time Azure also exhibited slightly better consistency with a coefficient of variation of 8.37% compared to 15.39% for AWS (Fig. 2).

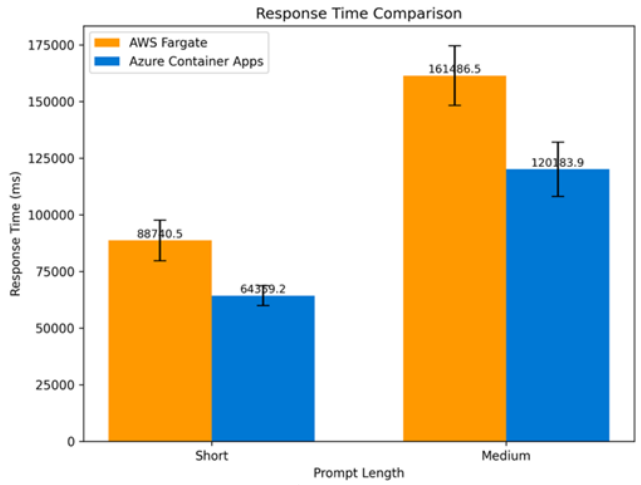

Fig. 1. Response time comparison for short and medium prompts – standard infrastructure

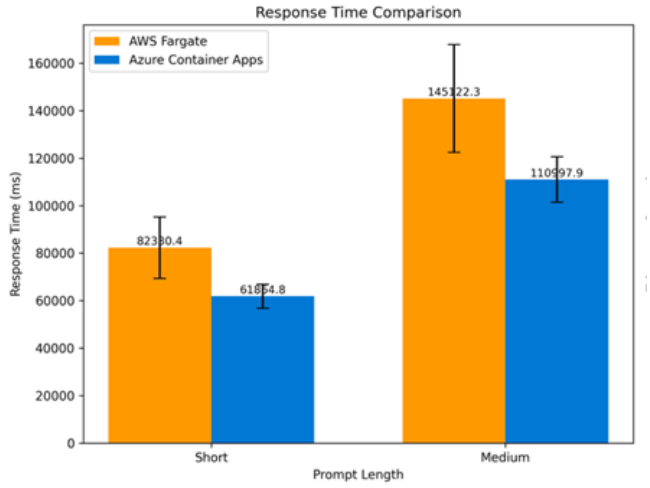

Fig. 2. Response time comparison for short and medium prompts – auto-scaling infrastructure

These results demonstrate that Azure Container Apps showed more consistent performance for short and medium prompts in both infrastructures, confirming the proposition about platform reliability differences. The statistical significance in performance differences for these tests also validates that cloud provider architectures fundamentally impact performance characteristics.

3.3. Streaming performance results

The streaming performance tests (S2) measured token-by-token response delivery capabilities, capturing metrics critical for interactive applications requiring progressive content generation.

For standard infrastructure, time to first token measurements showed a statistically significant difference (p = 0.0140) between platforms, with AWS Fargate delivering the first token in 693.5 ms versus Azure Container Apps at 897.3 ms (22.71% advantage for AWS). The consistency analysis revealed AWS had a coefficient of variation of 5.28% compared to Azure's 27.80%, indicating substantially more consistent initial response times on AWS. Token interval measurements showed small differences between platforms (AWS: 236.43 ms, Azure: 182.77 ms, p = 0.0002). However, Azure demonstrated better consistency in token interval with a coefficient of variation of 6.28% compared to AWS at 8.64%.

For auto-scaling infrastructure, AWS demonstrated significant advantage (p = 0.0002) in time to first token metrics with a median of 683.17 ms versus Azure at 1322.19 ms (48.33% advantage for AWS). For token interval, Azure showed a 29.39% advantage with 159.82 ms versus AWS at 226.32 ms, with statistical significancy (p = 0.0002). Azure demonstrated better token interval consistency (CV: 6.15% vs 15.62%) but showed worse time to first token consistency (CV: 34.42% vs 12.39%). Azure showed superior token generation rate at 6.2 tokens per second versus AWS at 4.4 tokens per second.

The complete streaming performance metrics are summarized in Table 3.

Table 3. Streaming performance metrics summary

Platform	Infrastructure	Time to First Token (ms)	Token Interval (ms)	Tokens per second	TTFT CV (%)	TI CV (%)
AWS Fargate	Standard	703.39	236.43	4.21	8.18	8.64
Azure Container Apps	Standard	773.59	182.77	5.43	26.26	6.28
AWS Fargate	Auto-scaling	683.17	226.32	4.4	12.39	15.62
Azure Container Apps	Auto-scaling	1322.19	159.82	6.2	24.42	6.15

These streaming results demonstrate statistically significant performance differences between platforms while also showing that each platform excels in consistency for different aspects of the streaming process across both infrastructure types, confirming both the performance variation and reliability difference propositions.

3.4. Concurrent load performance results

The concurrent load test (S3) evaluated performance characteristics under simultaneous user requests with gradually increasing concurrency levels (1, 3, 5, and 8 simultaneous requests).

For standard infrastructure, under single-request conditions, Azure Container Apps demonstrated faster response times (110617.01 ms vs. 129772 ms). As concurrency increased, the platforms exhibited divergent scaling behaviors. At concurrency level 8, Azure Container Apps demonstrated massive advantage in success rate (100% vs. 0% for AWS), indicating superior stability under high load conditions. Response time degradation patterns differed significantly between platforms. AWS Fargate showed a steep increase in response time as concurrency increased, while Azure Container Apps demonstrated more gradual degradation.

For auto-scaling infrastructure, the differences were less pronounced. Both platforms consistently maintained a 100% during 1, 3, 5 and 8 concurrent requests. Response times under load showed Azure maintaining consistent performance (around 125000 ms) across concurrency levels, while AWS response times escalated dramatically from 112920 ms at level 1 to 313280 ms at level 8. This demonstrates Azure's superior architecture for handling concurrent loads with less performance degradation (Table 4).

These results clearly demonstrate that platform architecture significantly influences stability under varying loads. Azure Container Apps showed less variation in success rates and more predictable scaling behavior as concurrency increased in both infrastructure types, confirming that different platforms exhibit varying degrees of performance degradation when performing similar tasks under increased load.

Statistical significance analysis was performed to validate the relationship between increasing concurrency levels and response time degradation. Pearson correlation coefficients

were calculated for each platform across both infrastructure types, followed by Fisher's Z transformation to enable statistical testing.

For standard infrastructure, AWS Fargate showed a strong positive correlation ($r = 0.913$) between concurrency level and response time, while Azure Container Apps demonstrated a moderate correlation ($r = 0.681$). Statistical testing using the t-distribution ($df = 9$) revealed both correlations were statistically significant (AWS: $p = 0.0033$, Azure: $p = 0.0334$), confirming that the observed performance degradation patterns were not due to random variation.

For auto-scaling infrastructure, both platforms exhibited very strong positive correlations between concurrency and response time (AWS: $r = 0.981$, $p = 0.0008$; Azure: $r = 0.988$, $p = 0.0005$). The extremely low p-values indicate highly significant relationships, with Azure showing a slightly stronger correlation despite maintaining more consistent absolute response times across concurrency levels.

These statistical analyses confirm that the observed scaling behaviors represent fundamental architectural differences between the platforms rather than measurement artifacts or random variations. The significant correlations validate that both platforms experience performance degradation under increasing load, but with markedly different patterns and magnitudes.

Table 4. Concurrent load test results by concurrency level

Platform	Infrastructure	Concurrency Level	Response Time (ms)	Success Rate (%)
AWS Fargate	Standard	1	133640	100.00
Azure Container Apps	Standard	1	93262	100.00
AWS Fargate	Standard	3	213550	100.00
Azure Container Apps	Standard	3	150358	100.00
AWS Fargate	Standard	5	239446	100.00
Azure Container Apps	Standard	5	161735	100.00
AWS Fargate	Standard	8	358235	100.00
Azure Container Apps	Standard	8	268319	100.00
AWS Fargate	Auto-scaling	1	112920	100.00
Azure Container Apps	Auto-scaling	1	99047	100.00
AWS Fargate	Auto-scaling	3	164049	100.00
Azure Container Apps	Auto-scaling	3	120200	100.00
AWS Fargate	Auto-scaling	5	189598	100.00
Azure Container Apps	Auto-scaling	5	126280	100.00
AWS Fargate	Auto-scaling	8	331280	100.00
Azure Container Apps	Auto-scaling	8	151069	100.00

4. Evaluation of key propositions

4.1. Platform consistency and reliability

The proposition that one platform would demonstrate more consistent performance metrics across multiple test runs, indicating greater reliability for production workloads, is partially supported with mixed results across different testing scenarios.

For baseline performance, AWS Fargate showed slightly better consistency in auto-scaling infrastructure with a CV of 18.61% compared to Azure's 25.99%, while both platforms had similar consistency for standard infrastructure.

For basic inference tests with short prompts, Azure Container Apps consistently demonstrated better run-to-run reliability with lower CVs in both infrastructure types (standard: 6.72% AWS vs 10.24% Azure; auto-scaling: 8.04% Azure vs 15.45% AWS). With medium prompts, the platforms showed similar consistency in standard infrastructure, but Azure performed better in auto-scaling scenarios (CV: 8.37% Azure vs 15.39%).

In streaming performance, the results were mixed. AWS showed significantly better consistency for time to first token in standard infrastructure (CV: 8.18% AWS vs 26.26% Azure), while Azure demonstrated superior consistency in token interval measurements across both infrastructure types (standard: 6.28% vs 8.64%; auto-scaling: 6.15% Azure vs 15.62% AWS).

Overall, Azure Container Apps demonstrated more consistent performance across a greater number of test scenarios, particularly in inference processing and under load conditions, making it generally more reliable for production workloads with sustained usage (Fig. 3 and Fig. 4).

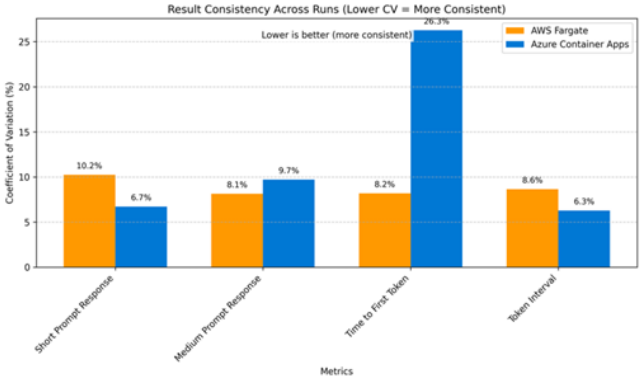

Fig. 3. Coefficient of variation comparison across all metrics – standard infrastructure

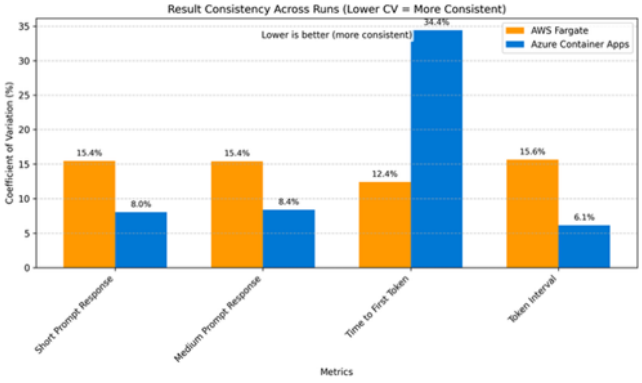

Fig. 4. Coefficient of variation comparison across all metrics – auto-scaling infrastructure

4.2. Performance variation between cloud providers

The proposition regarding statistically significant variations in performance characteristics of cloud-based container services across different providers for identical workloads is strongly supported by the data across all test scenarios.

Statistical analysis consistently showed significant differences ($p < 0.05$) between AWS Fargate and Azure Container Apps across multiple performance dimensions:

Baseline API response times showed statistically significant differences ($p = 0.0002$ for standard, $p = 0.0022$ for auto-scaling) with AWS demonstrating 32.78% and 31.13% advantages respectively.

Basic inference tests revealed significant differences in response times for both short and medium prompts ($p = 0.0003$ and $p = 0.0002$ for standard, $p = 0.0013$ for both length in auto-scaling), with Azure Container Apps consistently outperforming AWS Fargate.

Streaming performance metrics showed significant differences in both time to first token ($p = 0.0140$ for standard, $p = 0.0003$ for auto-scaling) and token interval ($p = 0.0002$ for standard, $p = 0.0004$).

These consistent, statistically significant variations confirm that cloud provider architecture fundamentally impacts container performance characteristics even when running identical workloads (Table 5).

Table 5. Summary of statistical significance testing

Metric	Infrastructure	p-value	Significant ($p < 0.05$)	Advantage To
Baseline API Response	Standard	0.0002	Yes	AWS Fargate
Baseline API Response	Auto-scaling	0.0022	Yes	AWS Fargate
Short Prompt Response	Standard	0.0003	Yes	Azure Container Apps
Short Prompt Response	Auto-scaling	0.0013	Yes	Azure Container Apps
Medium Prompt Response	Standard	0.0002	Yes	Azure Container Apps
Medium Prompt Response	Auto-scaling	0.0013	Yes	Azure Container Apps
Time to First Token	Standard	0.0140	Yes	AWS Fargate
Time to First Token	Auto-scaling	0.0003	Yes	AWS Fargate
Token Interval	Standard	0.0002	Yes	Azure Container Apps
Token Interval	Auto-scaling	0.0004	Yes	Azure Container Apps

4.3. Platform architecture impact on stability

The proposition that platform architecture influences the stability of performance metrics during operation of the AI model, where different platforms show varying degrees of response time variation when performing similar tasks, is strongly supported by the concurrent load test results.

The concurrent load tests demonstrated dramatically different behavior between platforms:

In standard infrastructure, Azure Container Apps and AWS Fargate maintained 100% success rates across all concurrency levels: 1, 3, 5, and 8 (Fig. 5).

In auto-scaling infrastructure, both platforms maintained 100% success rates at all concurrency levels (1, 3, 5, 8)

Azure showed more gradual performance degradation as concurrency increased, maintaining relatively consistent response times around 124000 ms across concurrency levels in auto-scaling infrastructure. In contrast, AWS response times escalated dramatically from 112000 ms at level 1 to 313000 ms at level 8 (Fig. 6).

These results clearly demonstrate that platform architecture substantially influences the stability of performance metrics during operation under varying loads, with Azure Container Apps exhibiting less performance degradation and more predictable scaling behavior, particularly in high-concurrency scenarios.

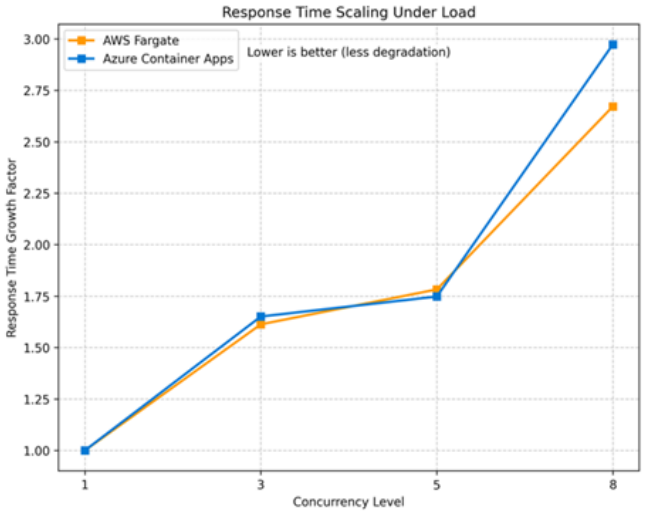

Fig. 5. Performance stability under increasing load – standard infrastructure

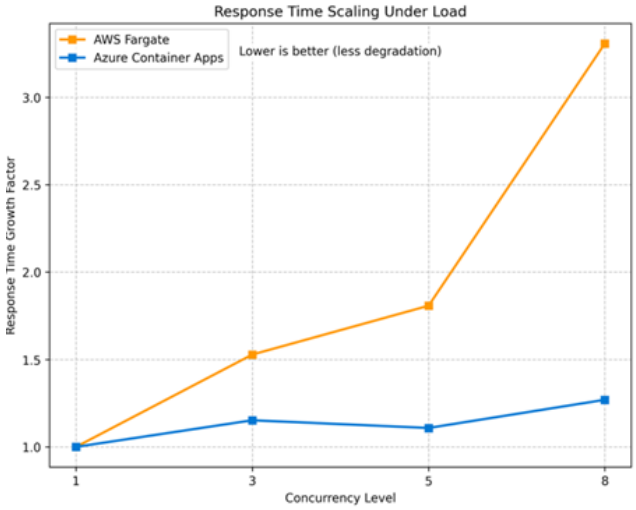

Fig. 6. Performance stability under increasing load – auto-scaling infrastructure

5. Discussion

5.1. Performance findings in context of existing literature

Our comprehensive evaluation of AWS Fargate and Azure Container Apps for LLM deployments reveals both convergent and divergent findings compared to existing research in cloud container performance studies.

Our baseline performance results, showing AWS Fargate's 32.78% advantage in API response time (395.50 ms vs 588.37 ms), partially contradict Abraham & Yang [1], who reported Azure Container Apps outperforming AWS App Runner. However, this discrepancy likely stems from architectural differences – AWS App Runner and Fargate represent distinct service models within the AWS ecosystem. Our findings align with Jain et al. [5], who emphasized that Fargate's ECS cluster abstraction provides efficient resource isolation with minimal overhead, which manifests in our superior baseline API responsiveness measurements.

The inference performance results present novel insights that directly address the unique computational challenges identified in section 2.2. Azure Container Apps' 27.47% advantage for short prompts and 25.58% for medium prompts validates Waseem et al.'s [20] observations about "large variance in processing times" where prompt length and complexity significantly affect response generation. Our findings demonstrate that Azure's architecture better handles these variable processing demands inherent to LLM workloads.

The superior consistency metrics (CV of 6.72% for Azure vs 10.24% for AWS) directly address the resource optimization challenges for LLM inference highlighted in the literature review. This aligns with Srivastava et al.'s [17] emphasis on "specialized approaches to managing memory and computing resources" for LLM applications, suggesting that Azure's Kubernetes-based foundation with KEDA integration provides superior resource allocation strategies for the high memory usage and intensive CPU processing characteristic of LLM inference operations.

Our results also validate the containerization benefits discussed by Dittakavi [9] and Golec et al. [4] regarding cold start mitigation. Both platforms showed consistent performance without the significant cold start latency issues that plague function-as-a-service platforms, confirming that container-based approaches successfully address the "long time it takes to load multi-gigabyte models into memory" problem identified in LLM deployment literature.

Our streaming performance analysis reveals architectural trade-offs crucial for interactive LLM applications requiring progressive content generation, directly addressing the requirements outlined in section 2.2. AWS Fargate's 22.71% advantage in time-to-first-token addresses the initial latency concerns highlighted by Stroh et al. [18] for "interactive applications requiring progressive content generation (streaming)," while Azure's superiority in sustained token generation (5.43 vs 4.21 tokens/second) aligns with the need for "efficient response caching" during lengthy connections.

The significant consistency differences (AWS CV: 5.28% vs Azure CV: 27.80% for time-to-first-token) provide empirical evidence for the infrastructure optimization challenges identified in LLM deployment literature. Azure's better token interval consistency (6.28% vs 8.64%) validates the importance of specialized container platform configuration for LLM needs, as emphasized by Sagi [14] regarding "appropriate configuration for the specific needs of LLM applications."

These findings extend beyond traditional web application performance metrics to address the unique "significant network bandwidth needs for handling concurrent user requests" and variable processing times inherent to LLM systems, as identified in the literature review.

Our concurrent load findings address critical challenges identified in section 2.2 regarding "scaling LLM in production environments" and the complexities of "managing multiple model instances and load balancing between them." Azure Container Apps' stable response times across concurrency levels (around 124,000 ms) versus AWS Fargate's dramatic escalation (112,920 ms to 313,280 ms) validates Stroh et al.'s [18] observations about LLM load characteristics that "often exhibit significant variance depending on application usage patterns."

This performance stability directly addresses the resource allocation strategies discussed in the literature review, confirming that Azure's Kubernetes foundation with KEDA scaling better handles the "specialized approaches to managing memory and computing resources" required for LLM applications, as emphasized by Srivastava et al. [17].

The results also validate concerns raised by Jonnakuti [6] about security and access management in LLM cloud deployments. Both platforms maintained 100% success rates across concurrency levels, but Azure's more predictable scaling behavior suggests better resource isolation and access control under varying loads, addressing the "security and access management aspects of the models" that "further complicate the deployment architecture" in LLM systems.

Our statistical analysis revealing significant differences ($p<0.05$) across multiple metrics addresses a methodological gap in previous research. While studies like Roloff et al. [12] provided valuable performance insights, they lacked rigorous statistical validation for modern container platforms. Our ensemble analysis approach with Mann-Whitney U tests provides the statistical rigor necessary for production decision-making.

The consistent statistical significance across diverse metrics (baseline API response, inference processing, streaming performance) demonstrates that architectural differences between cloud providers result in measurable, reliable performance variations even for identical workloads—a finding that extends theoretical architectural discussions into empirical validation.

5.2. Implications for production LLM deployments

Our findings align with Waseem et al.'s [20] emphasis on containerization's role in multi-cloud LLM environments, where "roles, strategies, challenges, and solutions for effective implementation" require careful platform evaluation. For latency-critical applications requiring fast initial response times, AWS Fargate's superior baseline API performance and time-to-first-token metrics make it preferable. This addresses Stroh et al.'s [18] observation about "themes of building LLM-based applications for production" where initial response latency significantly impacts user experience.

Conversely, for inference-intensive applications requiring sustained processing and predictable scaling, Azure Container Apps' superior consistency and load handling capabilities provide advantages. This finding supports Srivastava et al.'s [17] argument for "leveraging Kubernetes for Large Language Model deployment" in production environments, where Azure's Kubernetes-based architecture demonstrates measurable benefits for sustained LLM workloads.

The superior consistency of Azure Container Apps across most scenarios (particularly the CV of 6.15% for token interval in auto-scaling infrastructure) addresses Jonnakuti's [6] emphasis on "best practices for scaling generative AI in regulated industries," where predictable performance is crucial for compliance and reliability requirements in enterprise deployments.

This study directly addresses the significant research gap identified in section 2.2: "a significant research gap in the area of systematic performance analysis of container platforms for LLM workloads." Our comprehensive evaluation fills the void where "existing publications mainly focus on optimization of the models themselves, resource management in isolation or general aspects of cloud performance, while there is a lack of empirical comparative studies of the specifics of LLM performance on different container platforms under production conditions."

Our findings provide crucial empirical data that supports Waseem et al.'s [20] framework for "containerization in multi-cloud environment," demonstrating how platform architectural differences manifest in measurable performance variations for LLM workloads. The study validates their emphasis on understanding "roles, strategies, challenges, and solutions for effective implementation" by providing concrete performance benchmarks for platform selection decisions.

The streaming performance analysis addresses specific challenges highlighted by Stroh et al. [18] in their practitioner's view of "building LLM-based applications for production." Our findings on time-to-first-token vs. sustained token generation trade-offs provide empirical validation for their observations about the complexity of LLM production deployments and the need for platform-specific optimization strategies.

Additionally, our concurrent load testing results support Srivastava et al.'s [17] advocacy for Kubernetes-based deployments, with Azure Container Apps' superior scaling behavior under increasing concurrency validating their argument that "the future of AI in production" benefits from Kubernetes orchestration capabilities for LLM workloads.

Our results must be interpreted within the context of LLM-specific workloads using the Llama 3.2:1b model. The memory-intensive nature and streaming requirements of LLM inference create unique demands that may not generalize to other containerized applications, as emphasized by Waseem et al. [20] in their analysis of containerization challenges across different application types.

Future research should expand to include additional model sizes and different LLM architectures, addressing Stroh et al.'s [18] call for broader empirical studies of LLM production deployment patterns. The integration of specialized AI accelerators and emerging serverless GPU offerings will likely impact the performance landscape for LLM deployments, as suggested by Srivastava et al.'s [17] discussion of evolving Kubernetes capabilities for AI workloads.

Additionally, while our study provides insights into two major platforms, Jonnakuti's [6] analysis of "LLMs in the cloud" suggests that cost-effectiveness analysis alongside performance metrics represents a critical next step for comprehensive platform evaluation. The rapidly evolving cloud container landscape warrants ongoing evaluation as platforms introduce new features

specifically designed for AI workloads, particularly in regulated industries where predictable performance and compliance requirements intersect.

In conclusion, our findings demonstrate that platform architecture significantly influences performance characteristics for LLM workloads, with each platform offering distinct advantages depending on specific application requirements. The choice between AWS Fargate and Azure Container Apps should be guided by workload-specific performance priorities rather than general platform preferences, supporting the practitioner-oriented approach advocated by Stroh et al. [18] for LLM production deployments.

6. Conclusions

The purpose of this work was to conduct a comprehensive performance evaluation of AWS Fargate and Azure Container Apps as deployment platforms for containerized Large Language Model applications, specifically examining whether architectural differences between these platforms result in measurable performance variations under diverse operational conditions. The research accomplished systematic benchmarking across multiple scenarios including baseline performance, inference processing with varying prompt lengths, streaming capabilities, and concurrent load handling, using both standard and auto-scaling infrastructure configurations. The study established three key propositions: platform consistency differences were partially confirmed with Azure Container Apps showing more consistent performance across most test scenarios; statistically significant performance variations were strongly validated with significant differences ($p < 0.05$) found across multiple metrics; and architectural influence on stability was strongly demonstrated with Azure exhibiting less performance degradation under increasing loads. The research limitations include the focus on a single LLM model (Llama 3.2:1b), testing of two services and the memory-intensive nature of LLM workloads which may not generalize to other containerized applications. Future work is planned to expand the research to include additional LLM models of varying sizes, test across more platforms, evaluate cost-effectiveness alongside performance metrics, and investigate the behavior of these platforms under different workload patterns beyond inference tasks.

References

[1] Abraham A., Yang J.: Analyzing the system features, usability, and performance of a containerized application on serverless cloud computing systems. Research Square 2023 [https://doi.org/10.21203/rs.3.rs-3167840/v1].

[2] Agache A. et al.: Firecracker: Lightweight virtualization for serverless applications. 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI '20), 2020, 419–434.

[3] Dittakavi R. S. S.: Cold start latency in serverless computing: Current trends and mitigation techniques. EDUZONE: International Peer Reviewed/Refereed Multidisciplinary Journal 12(2), 2023, 134–138.

[4] Golec M. et al.: Cold start latency in serverless computing: A systematic review, taxonomy, and future directions (Appendix). Journal of ACM 37(4), 2024, 1–8.

[5] Jain P. et al.: Performance analysis of various server hosting techniques. Procedia Computer Science 173, 2020, 70–77 [https://doi.org/10.1016/j.procs.2020.06.010].

[6] Jonnakuti S.: LLMs in the cloud: Best practices for scaling generative AI in regulated industries. International Journal of Engineering Technology Research & Management 8(2), 2024, 105–112.

[7] Llama3.2:1b. Meta, 2024 [https://ollama.com/library/llama3.2:1b].

[8] McGrath G., Brenner P. R.: Serverless computing: Design, implementation, and performance. IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW), 2017, 405–410 [https://doi.org/10.1109/ICDCSW.2017.36].

[9] MSV J.: The evolution of serverless container platform on AWS Fargate. The New Stack, 2019 [https://thenewstack.io/the-evolution-of-serverless-container-platform-on-aws-fargate/].

[10] Ollama. GitHub - ollama/ollama: Get up and running with Llama 3.3, DeepSeek-R1, Phi-4, Gemma 3, Mistral Small 3.1 and other large language models. GitHub, 2025 [https://github.com/ollama/ollama].

[11] Petrovski T., Gusev M.: Container vs function as a service: Impact on cloud deployment for real-world applications. 47th MIPRO ICT and Electronics Convention (MIPRO), 2024, 869–874 [https://doi.org/10.1109/MIPRO60963.2024.10569811].

[12] Roloff E. et al.: High performance computing in the cloud: Deployment, performance and cost efficiency. IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom). Taipei, Taiwan, 2012, 371–378 [https://doi.org/10.1109/CloudCom.2012.6427549].

[13] Sadaqat M., Sánchez-Gordón M., Colomo-Palacios R.: Benchmarking serverless computing: Performance and usability. Journal of Information Technology Research 15(1), 2022, 1–17 [https://doi.org/10.4018/JITR.299374].

[14] Sagi S.: Overcoming challenges in deploying large language models for generative AI use cases: The role of containers and orchestration. International Journal of Computer Trends and Technology 72(2), 2024, 75–81 [https://doi.org/10.14445/22312803/IJCTT-V72I2P114].

[15] Seth D., Chintale P.: Performance benchmarking of serverless computing platforms. International Journal of Computer Trends and Technology 72(6), 2024, 160–167 [https://doi.org/10.14445/22312803/IJCTT-V72I6P121].

[16] Shrestha R., Nisha B.: Performance evaluation and comparison of microservices and serverless deployments in cloud. IEEE 8th International Conference on Smart Cloud (SmartCloud). Tokyo, Japan, 2023, 202–207 [https://doi.org/10.1109/SmartCloud58862.2023.00043].

[17] Srivastava S. et al.: The future of AI in production: Leveraging Kubernetes for Large Language Model deployment. International Journal for Multidisciplinary Research 7(1), 2025, 1–18.

[18] Stroh D., Mailach A., Siegmund N.: Themes of building LLM-based applications for production: A practitioner's view. arXiv preprint arXiv:2411.08574v2, 2024.

[19] Synergy Research Group: Cloud Market Jumped to $330 billion in 2024 – GenAI is Now Driving Half of the Growth. [https://www.srgresearch.com/articles/cloud-market-jumped-to-330-billion-in-2024-genai-is-now-driving-half-of-the-growth] (avaible: 7.06.2025).

[20] Waseem M. et al.: Containerization in multi-cloud environment: Roles, strategies, challenges, and solutions for effective implementation. arXiv preprint arXiv:2403.12980v2, 2024.

B.Eng. Mateusz Stęgierski
e-mail: s97221@pollub.edu.pl

He graduated with a degree in engineering from the Lublin University of Technology in 2024, majoring in computer science with a specialization in software engineering. His research interests include optimizing production processes with innovative technological solutions. His areas of expertise include process analysis, system integration design and implementation, and establishing robust CI/CD pipelines. Outside of his professional pursuits, he has a keen interest in discovering and creating music, exploring the intersection of technology and artistic expression.

https://orcid.org/0009-0007-1501-4156

B.Eng. Piotr Szpak
e-mail: s95585@pollub.edu.pl

Piotr Szpak graduated with a degree in engineering from the Lublin University of Technology in 2024, majoring in computer science and specializing in software engineering. His research interests include modern data stack architectures, cloud computing services for scalable data processing solutions, and contemporary approaches to data engineering workflows. He is also an aviation enthusiast with interest in gliding, skiing and cycling.

https://orcid.org/0009-0007-0517-7250

Ph.D. Sławomir Przylucki
e-mail: s.przylucki@pollub.pl

Slawomir Przylucki received his M.Sc. and Ph.D. degrees in electrotechnics from Lublin University of Technology (LUT), Poland, in 1991 and 1999, respectively. He has been employed at LUT since 1991, currently as an assistant professor at the Department of Computer Science. His scientific interests cover IP traffic engineering, wireless networks, sensor networks and video streaming systems. S. Przylucki is an author of more than 50 papers published in communication journals and presented at national and international conferences. He participated in several R&D projects funded by national authority as well as by European Union. Since 2013, he has been served as an expert in the field of IT projects for the government of the Republic of Polish and the Czech Republic.

https://orcid.org/0000-0001-9565-3802