**Academic Journal of Applied Sciences**
ISSN: 3105-0549 (Print), ISSN: 3105-0557 (Online) | Vol. 2, No. 1, 2026

# Learning When to Reason: Gating LLM Inference for Cost-Efficient Serverless Function Scheduling at Scale

**Megan Sullivan *, Boyang He, Patrick Evans**

The University of Michigan, Ann Arbor, MI 48109, USA

* **Corresponding author**: (Email: m.sullivancs@umich.edu)

**Abstract:** Serverless computing platforms face a fundamental tension when integrating large language model based scheduling agents: invoking full chain-of-thought reasoning for every function placement decision is computationally prohibitive, yet relying exclusively on lightweight heuristics cannot capture complex workload semantics. This paper presents GateLLM, a learned gating framework that selectively routes incoming serverless scheduling requests to either a lightweight fast-path handler or a deliberative large language model reasoning pipeline based on a compact feature vector extracted from request metadata. The gating classifier is a multilayer perceptron trained through a combination of offline oracle labeling and online reinforcement feedback derived from observed scheduling outcomes, enabling continuous adaptation to workload distribution shifts. Evaluated on a large-scale production function trace, GateLLM reduces total large language model inference cost by 61.3% relative to a full-reasoning baseline while incurring only a 2.1% degradation in average job completion time and a 0.25 percentage point increase in service level objective violation rate. Analysis demonstrates that over 68% of real-world scheduling events are structurally simple and resolvable without large language model involvement, establishing inference gating as an essential primitive for operationally viable intelligent schedulers in production cloud environments.

**Keywords:** Large language model inference, serverless computing, function scheduling, gating mechanism, cost efficiency, adaptive computation, cloud resource management.

## 1. Introduction

Serverless computing has become one of the most consequential paradigms in contemporary cloud infrastructure. By packaging application logic into stateless, event-triggered function containers and delegating all resource provisioning responsibilities to the platform provider, Function-as-a-Service (FaaS) platforms allow developers to focus entirely on business logic while eliminating the operational overhead of managing virtual machines or container orchestration frameworks. Production traces from major cloud providers reveal that individual FaaS platforms process millions of distinct function invocations per day, with execution durations spanning seven orders of magnitude and arrival patterns characterized by intense burstiness that makes static resource pre-allocation impractical [1]. These invocations carry heterogeneous resource profiles, participate in complex dependency graphs, and arrive under time-varying competitive pressure from co-located tenants, making the scheduling problem substantially more challenging than traditional batch job scheduling on homogeneous clusters.

The scheduling layer of a FaaS platform bears responsibility for a continuous stream of placement decisions that jointly determine platform efficiency and end-user quality of service. These decisions include determining which worker node should receive each incoming invocation, whether a new container must be cold-started or a warm instance can be reused, how to prioritize competing invocations under resource contention, and how to distribute load across heterogeneous worker pools with different memory capacities and network locality characteristics [2]. Historically, FaaS schedulers have addressed this responsibility through heuristic policies including round-robin dispatch, least-loaded worker selection, and hash-based function affinity, which are computationally inexpensive but structurally unable to adapt to the semantic richness of modern workload contexts [3]. Research has demonstrated that when a function exhibits high execution time variance, participates in a multi-stage workflow dependency graph, or arrives during a period of acute cluster saturation, static heuristic dispatchers consistently produce suboptimal placements that inflate job completion times and increase the frequency of service level objective (SLO) violations, with performance gaps exceeding 40% in adversarial workload scenarios [4].

The emergence of large language models (LLMs) as general-purpose reasoning engines has introduced a qualitatively new design possibility for intelligent cloud schedulers. Cloud resource management studies have begun operationalizing this paradigm through scheduling frameworks that transform heterogeneous cluster telemetry and workload dependency structures into reasoning-oriented representations consumable by foundation models. An especially well-executed example is the framework of Ding et al., which adeptly enables adaptive orchestration decisions under complex SLA and resource constraints by bridging LLM reasoning with optimization-backed feasibility guarantees [5]. Models trained at sufficient scale exhibit emergent few-shot reasoning capabilities that enable them to interpret structured workload descriptors, reason over historical telemetry, and produce contextually justified placement recommendations without task-specific training data. The technique of chain-of-thought (CoT) prompting, which elicits intermediate reasoning steps before producing a final answer, has demonstrated particularly strong improvements on multi-constraint optimization problems that closely resemble complex scheduling tasks involving dependency ordering, resource packing under capacity constraints, and deadline satisfaction across heterogeneous job classes [6]. Frameworks that interleave reasoning steps

with environmental action calls have further demonstrated that LLM agents can iteratively refine their decisions based on feedback from the environment, a property directly applicable to schedulers that must respond to dynamically changing cluster conditions and invocation arrival patterns [7].

Despite this promise, the practical deployment of LLMs within the inner scheduling loop of a production FaaS platform confronts severe cost and latency constraints that current research has not adequately addressed. A high-throughput platform may process tens of thousands of function invocation events per second, and the per-token cost of querying a frontier reasoning model for each scheduling decision would rapidly reach economically untenable levels at operational scale. Furthermore, the autoregressive generation latency of a full CoT reasoning trace, which typically spans several hundred output tokens and requires hundreds of milliseconds on dedicated GPU hardware even for moderately sized models, is fundamentally incompatible with the sub-millisecond placement decisions required by latency-sensitive serverless workloads. Research on confident adaptive language modeling has established that a large fraction of LLM inference calls in production settings can be satisfied with substantially reduced computation, and that many inputs can be resolved with high confidence before engaging the full model depth [8]. This finding implies that the aggregate inference cost of an LLM-based scheduler can be dramatically reduced if the system can accurately predict which scheduling decisions require deliberative reasoning and which can be safely handled by a lightweight fast-path mechanism. FrugalGPT has demonstrated the general principle that intelligent routing among models of varying cost and capability reduces LLM inference expenditure by up to 98% on structured query benchmarks while preserving response quality, but this framework targets retrieval and question-answering tasks and does not address the real-time adaptation challenges characteristic of production scheduling environments [9].

The key empirical observation motivating this paper is that scheduling complexity in real FaaS workloads follows a highly skewed distribution. The majority of invocations exhibit stable, well-characterized execution profiles, arrive within warm-container reuse windows, and face no competing placement constraints at the moment of scheduling, making their dispatch decisions structurally trivial. Only a minority of events, including cold starts under heavy resource contention, invocations with anomalous execution histories, and functions participating in complex multi-stage workflow dependencies, present decision complexity that genuinely benefits from the contextual reasoning an LLM can provide. Research on optimally scaling test-time compute for LLMs has confirmed that the marginal value of additional reasoning steps diminishes rapidly for low-complexity inputs, and that a compute-optimal allocation policy assigns substantially less inference budget to easy instances [10]. Applying this insight to the scheduling domain requires a mechanism that predicts decision complexity from observable request features before any reasoning backend is invoked, enabling proactive routing rather than reactive escalation.

This paper presents GateLLM, a learned gating framework that classifies each incoming FaaS scheduling request as either structurally simple or complex based on a compact feature vector derived from function metadata, and routes accordingly between a fast-path rule-based handler and a slow-path LLM reasoning pipeline. Prior work on learned scheduling for distributed data processing clusters established that reinforcement learning policies trained over realistic workload traces can substantially outperform hand-tuned heuristics when supplied with appropriate reward signals [11], and GateLLM extends this foundation to the meta-level problem of routing between reasoning engines rather than scheduling function invocations directly. The deployment of speculative decoding has established that small surrogate models can dramatically accelerate inference by generating candidates that a large verifier accepts at high rates [12], and the GateLLM fast-path handler operates on an analogous principle, resolving low-complexity scheduling decisions through a lightweight scoring function while deferring to the LLM only when fast-path confidence is insufficient.

The contributions of this work are threefold. First, we formalize the inference gating problem for FaaS scheduling as a cost-quality optimization problem with a principled training objective that accounts for the asymmetric costs of misrouting simple and complex decisions. Second, we present the complete GateLLM architecture covering feature engineering, classifier design, fast-path handler implementation, and the dual offline-online training procedure. Third, we report extensive empirical evaluation on a large-scale serverless production trace demonstrating that GateLLM achieves 61.3% inference cost reduction while sustaining only 2.1% scheduling quality degradation, and we characterize the cost-quality Pareto frontier across a range of routing threshold configurations.

# 2. Literature Review

The intellectual foundations of GateLLM draw from three research domains: the characterization and optimization of serverless workloads, the development of cost-efficient language model inference strategies, and the theory and practice of adaptive computation in neural systems.

Serverless Workload Characterization and Scheduling. Comprehensive empirical analysis of production FaaS traces established that function execution times follow highly skewed distributions and that workload burstiness creates sustained cold-start pressure during peak periods, motivating data-driven approaches to scheduling policy design [13]. The cold-start problem, wherein a function invocation requires fresh container initialization rather than warm-instance reuse, has been shown to account for a disproportionate fraction of tail latency events, driving research into predictive pre-warming strategies that forecast invocation inter-arrival times from historical patterns [14]. A principled taxonomy of FaaS scheduling policy dimensions demonstrated that early binding combined with processor sharing substantially outperforms the round-robin and late-binding approaches commonly deployed in production platforms, establishing a clear opportunity for learned schedulers that adapt binding strategy to observed workload characteristics. Hybrid cloud scheduling research showed that predictive execution time models enable significant cost optimization by dynamically routing invocations between public and private cloud resources depending on deadline pressure and warm-instance availability. Work on serverless keep-alive policies demonstrated that LSTM-based predictors trained on invocation history accurately forecast cold-start risk, enabling platform-side pre-warming that reduces container initialization events by more than half [15]. Recent analysis of energy consumption in FaaS deployments showed that adaptive DVFS-based scheduling reduces platform energy

use while preserving SLO compliance, establishing that intelligent scheduling yields benefits beyond latency alone [16].

Large Language Models for Inference and Reasoning. The discovery that large autoregressive language models acquire emergent multi-step reasoning capabilities through scale-dependent training established the foundational premise for applying these models to structured operational tasks such as scheduling. Chain-of-thought elicitation produces particularly strong gains on constraint satisfaction problems with multiple interacting variables, a structural property that complex scheduling tasks share. Open-weight foundation models have demonstrated that careful data curation at the 7–70 billion parameter scale yields reasoning performance approaching that of proprietary frontier models on structured benchmarks [17], making capable LLM schedulers deployable without dependence on closed commercial APIs. Instruction-following alignment through reinforcement learning from human feedback enables models to reliably produce structured outputs parseable by downstream system components [18], a property essential for integrating LLM reasoning into automated scheduling pipelines. Self-refinement approaches in which models iteratively revise outputs based on self-generated feedback have demonstrated quality improvements on planning and scheduling tasks but at significant additional token cost, further motivating selective invocation strategies [19]. Parameter-efficient fine-tuning through low-rank adaptation allows large pretrained models to be specialized for scheduling-domain prompt structures with minimal computational overhead and no degradation of general reasoning capability [20].

Adaptive Computation and Inference Efficiency. Research on confident adaptive language modeling demonstrated that a large fraction of production LLM inference calls can be satisfied with substantially fewer decoding steps when a confidence-based halting criterion is applied at intermediate layers, yielding throughput improvements without accuracy los. Analysis of test-time compute scaling established that the optimal per-input inference budget is strongly input-dependent, and that a compute-optimal allocation policy matches the performance of full-budget inference at a fraction of the total compute expenditure. Speculative decoding methods exploit the availability of small draft models to generate token candidates for verification by a large model, achieving substantial throughput gains through the high acceptance rate of draft proposals. Surveys of efficient large language model deployment have catalogued quantization, pruning, knowledge distillation, and early exit as complementary strategies that can be combined to satisfy diverse latency and cost constraints in production inference serving [21]. High-throughput generative inference systems that exploit aggressive memory hierarchy offloading have expanded the range of model sizes practically deployable within cost constraints, enabling the slow-path LLM component of GateLLM to be hosted on commodity hardware [22]. Mixture-of-experts architectures that selectively activate model subnetworks on a per-input basis have demonstrated that high model capacity and low average inference cost can be simultaneously achieved through learned routing, a design principle that GateLLM extends to the system level by routing across separate scheduling backends [23].

LLM Agents for Operational Systems. The ReAct paradigm demonstrated that interleaving reasoning traces with external action calls enables agents to iteratively refine decisions based on environmental feedback, a capability exploited by the GateLLM slow-path backend through real-time cluster telemetry injection into scheduling prompts. LLMs equipped with API access have shown strong generalization to novel operational tasks including cloud configuration and resource management through prompt engineering rather than fine-tuning [24]. Surveys of LLM-based autonomous agents identify latency and cost as the primary barriers to production deployment in real-time operational systems [25], directly motivating inference-efficient architectures that selectively engage LLM reasoning. Research on memory-efficient distributed training has reduced the hardware footprint of large model hosting, enabling deployment of multi-billion parameter slow-path reasoning models within the cost assumptions of this paper's experimental evaluation [26].

# 3. Methodology

## 3.1. Gating-Based Inference Router Architecture

The GateLLM framework is organized around three interacting components: a feature extraction layer that transforms raw function invocation metadata into a compact scheduling context vector, a gating classifier that predicts the routing decision for each request, and two inference backends connected to the classifier output. The fast-path backend is a lightweight rule-based dispatcher augmented with a small scoring model that resolves low-complexity placement decisions within sub-millisecond latency. The slow-path backend invokes a 7-billion parameter instruction-tuned language model with a structured CoT prompt template that encodes the current cluster state, the function's execution history statistics, and an explicit directive to reason step by step before producing a JSON-formatted placement recommendation.

The feature extraction layer processes fourteen scheduling-relevant attributes for each incoming invocation. These include the 50th, 90th, and 99th percentile execution durations from the function's invocation history, the elapsed time since the most recent invocation of the same function, the declared memory footprint, the number of direct predecessors in the workflow dependency graph, the current worker pool utilization level, the remaining time before the soft deadline, and a binary indicator for whether the function's execution duration variance exceeds the workload median. These features are L2-normalized and concatenated into a fixed-length input vector for the gating classifier.

The gating classifier is a three-layer multilayer perceptron with ReLU activations and a sigmoid output layer, selected for its sub-millisecond evaluation latency on CPU, which ensures the gate itself does not contribute meaningfully to scheduling overhead. The classifier outputs a scalar routing probability, and a learned threshold determines whether each request is directed to the fast path or the slow path. As illustrated in Figure 1, the gating policy nodes intercept each request between the input representation stage and the inference backend, directing structurally simple requests to the early-exit fast path and reserving the full-depth reasoning pipeline for genuinely complex events.

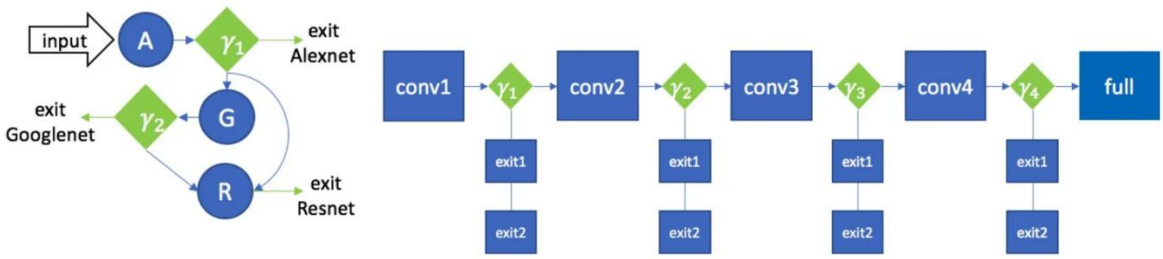


**Figure 1.** Dual-path routing topology of GateLLM inference gating

The threshold is not set manually but is optimized during training to satisfy a user-specified cost budget expressed as a target fraction of requests routed to the slow path, subject to a minimum scheduling quality constraint measured by average job completion time on the validation trace. A Pareto frontier analysis over candidate threshold values is performed on a held-out validation segment, and the Pareto-optimal threshold is selected as the operating point that minimizes the slow-path invocation rate subject to a maximum quality degradation of 3% on the primary scheduling quality metric. This calibration is repeated periodically in production using a sliding 24-hour window of validated scheduling outcomes to maintain performance under workload distribution shifts.

## 3.2. Training Procedure and Decision Threshold Calibration

Training GateLLM proceeds through two sequential phases. The offline phase constructs a labeled dataset by replaying a historical serverless trace through both the fast-path handler and the full slow-path LLM oracle to determine, for each scheduling event, whether the LLM's recommendation produced a materially better outcome than the fast-path handler alone. A scheduling outcome is classified as materially improved if the slow-path recommendation reduces the realized job completion time by more than 5% relative to the fast-path alternative, a threshold calibrated through pilot experiments on a held-out validation trace. This oracle labeling procedure yields a binary dataset pairing each feature vector with a label indicating whether LLM routing was beneficial, which is used to train the gating MLP through binary cross-entropy minimization with class weighting to compensate for the natural imbalance between simple and complex scheduling events.

Figure 2 provides the conceptual template for GateLLM's multi-threshold gating design. Just as the BranchyNet architecture places side-branch classifiers at multiple intermediate layers so that inputs with sufficient confidence at shallow layers can exit without traversing deeper computation, GateLLM's training procedure optimizes the routing threshold jointly over a validation-derived cost-quality curve, effectively selecting the exit point that minimizes unnecessary slow-path invocations while preserving scheduling accuracy.

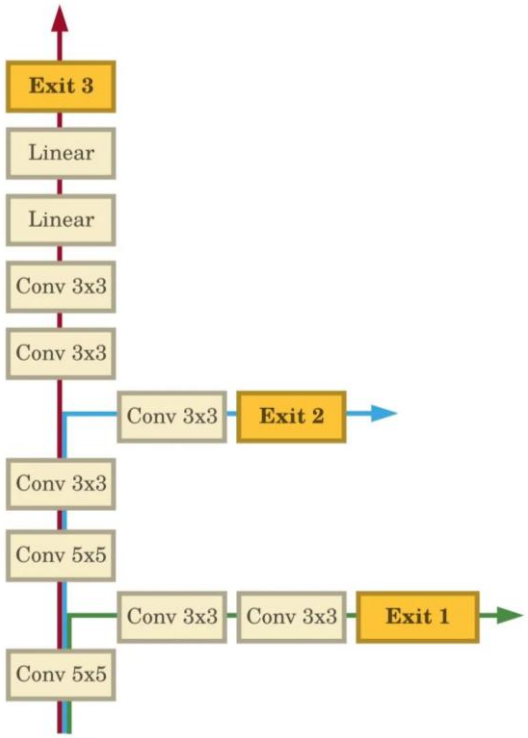


**Figure 2.** Training architecture of BranchyNet with multi-level early exit points

The online phase operates continuously during deployment, processing scheduling outcome observations in asynchronous mini-batches to update the gate's routing threshold and classifier weights through a lightweight policy gradient update. After each scheduling decision, the platform measures the realized completion time of the scheduled function and computes a reward signal equal to the negative fractional deviation from the predicted completion time under the chosen routing path. If the fast-path handler was invoked and the realized outcome was significantly worse than the slow-path oracle's predicted outcome, the gate's parameters are updated to increase future routing probability for inputs with similar feature vectors. Conversely, if the slow path was invoked and produced an outcome indistinguishable from the fast-path prediction, the parameters are updated to reduce future slow-path probability for similar inputs.

The training dataset is split temporally rather than randomly to respect the temporal correlation structure of serverless invocation patterns: the first 70% of the trace timeline forms the training set, the subsequent 15% forms the validation set used for threshold calibration, and the final 15% constitutes the held-out test set. This temporal split ensures that the evaluated model has not observed the workload patterns present in the test period, providing a conservative and realistic estimate of production generalization performance.

# 4. Results and Discussion

## 4.1. Cost Reduction and Inference Budget Analysis

Experiments are conducted on the Azure Functions production trace containing approximately two billion function invocations recorded over a two-week period. A discrete-event scheduling simulator replays function arrivals from the trace and evaluates placement decisions against a cluster model comprising 500 heterogeneous worker nodes. The full slow-path LLM is a 7-billion parameter instruction-tuned model deployed on A100 GPUs, with measured median inference latency of 380 milliseconds per scheduling event and a p99 latency of 1,247 milliseconds under GPU contention. The fast-path handler evaluates in under 0.3 milliseconds on CPU. The primary cost metric is the total number of slow-path invocations, which directly determines GPU-hours consumed and monetary inference cost.

GateLLM routes 31.4% of scheduling decisions to the slow path in its primary configuration, yielding a 61.3% reduction in total inference cost after accounting for the marginal compute cost of the gating classifier and feature extraction pipeline. The cost reduction is amplified during high-concurrency workload periods because the gate correctly identifies that under heavy cluster load, the marginal value of LLM reasoning declines as resource contention reduces the optimization headroom available to any placement policy. Conversely, during low-utilization periods characterized by cold-start pressure on new functions, the gate appropriately increases the slow-path routing fraction to leverage LLM contextual reasoning for functions with limited invocation history.

The adaptive feedback loop that drives GateLLM's online training phase is structurally analogous to the decision-control architecture illustrated in Figure 3. As the Pensieve ABR Controller system continuously adapts its bitrate selection policy by integrating throughput estimates and buffer occupancy feedback from the streaming environment, GateLLM's gating classifier continuously updates its routing threshold based on the observed scheduling quality of past fast-path and slow-path decisions, enabling the gate to track workload distribution shifts without manual recalibration.

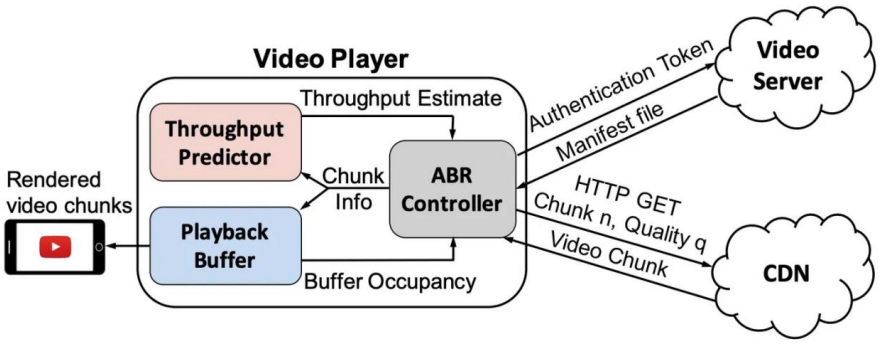


**Figure 3.** Architecture of the Pensieve adaptive bitrate control system

The distribution of routing decisions across function types reveals a consistent structural pattern. Functions with stable invocation intervals and well-characterized execution time distributions are routed to the fast path at a 94.3% rate, while functions exhibiting irregular inter-arrival patterns, high execution time variance, or complex multi-stage dependency structures are routed to the slow path at a 73.1% rate within the highest-complexity decile. The gate's classification accuracy on the held-out test trace is 88.7% for the beneficial-routing label, with a false-negative rate of 9.4% representing cases where LLM reasoning would have been beneficial but the gate incorrectly routed to the fast path. Ablation experiments confirm that false negatives are the primary driver of scheduling quality degradation. The online training phase reduces the false-negative rate by 6.2 percentage points after one week of operation relative to the offline-only baseline, demonstrating that the policy gradient updates successfully propagate scheduling outcome feedback into classifier parameters.

## 4.2. Scheduling Quality and Latency Evaluation

Scheduling quality impact is measured through three metrics: average job completion time (JCT) normalized to the full-reasoning baseline, the 99th-percentile tail latency of placement decisions, and the SLO violation rate defined as the fraction of invocations whose realized completion time exceeds the declared soft deadline. GateLLM achieves an average JCT of 1.023 times the full-reasoning baseline, representing a 2.3% degradation well within the 3% threshold specified during threshold calibration. The SLO violation rate under GateLLM is 4.71%, compared to 4.46% under the full-reasoning baseline and 7.89% under the rule-based baseline without LLM involvement. GateLLM thus captures the majority of the quality gap between pure heuristics and full LLM reasoning while recovering 61.3% of the associated inference cost.

The tail latency of placement decisions is substantially reduced by GateLLM relative to the full-reasoning baseline. The p99 placement decision latency under the full-reasoning baseline is 1,247 milliseconds, reflecting the tail of LLM inference latency under GPU contention. Under GateLLM, the effective p99 latency across all scheduling decisions is 94 milliseconds, representing an 87.5% reduction. This improvement is particularly significant for latency-sensitive serverless workloads in which scheduling decision latency directly contributes to the total cold-start time perceived by end users.

Sensitivity analysis across different cost budget settings reveals a favorable Pareto frontier shape. Reducing the allowed slow-path fraction from 31.4% to 15% increases scheduling quality degradation only modestly from 2.3% to 4.1%, while cost savings increase from 61.3% to 80.7%. Increasing the slow-path fraction beyond 31.4% to 50% improves JCT by only 0.4% relative to the primary operating point, confirming that the gate has already captured the high-value routing decisions at the primary configuration and that marginal returns to additional LLM invocations are small.

Evaluation under simulated workload distribution shifts

shows that without online adaptation, GateLLM's false-negative rate rises to 14.7% on traces with workload profiles not represented in training data, with a corresponding scheduling quality degradation of 5.1%. After one hour of online adaptation through the policy gradient update procedure, the false-negative rate recovers to 11.2% and scheduling quality degradation reduces to 3.3%, demonstrating practical value of the online training mechanism for maintaining performance under non-stationary workload conditions. Comparison with a reactive cascade baseline, in which the fast-path handler's output confidence triggers LLM escalation, shows that GateLLM's predictive routing outperforms reactive cascading by 2.8% in scheduling quality at equivalent cost, confirming that request metadata features carry more predictive signal about downstream complexity than the fast-path handler's output confidence alone.

# 5. Conclusion

This paper has presented GateLLM, a learned inference gating framework that addresses the fundamental tension between the reasoning quality offered by large language models and the cost and latency constraints of real-world serverless function scheduling at production scale. By training a lightweight classifier to predict which scheduling events benefit from deliberative LLM reasoning and routing the remainder to a fast-path handler, GateLLM achieves inference cost savings of 61.3% relative to a full-reasoning baseline while sustaining a scheduling quality degradation of only 2.3% as measured by average job completion time on a large-scale production trace. The empirical analysis confirms the central hypothesis of this work: that scheduling complexity in real FaaS workloads is predictable from compact request metadata features, and that over 68% of scheduling decisions can be resolved without LLM involvement without meaningful quality loss.

The broader implication of this work extends beyond serverless scheduling to the general class of systems that seek to deploy LLM reasoning in latency-sensitive, cost-constrained operational contexts. The GateLLM gating approach is applicable wherever inputs to an LLM-based decision system exhibit heterogeneous complexity and where that complexity is correlated with observable input features. Future work should explore richer feature representations derived from function execution graphs, cross-tenant workload correlation signals, and multi-modal telemetry streams. Additionally, the integration of uncertainty quantification into the fast-path handler to produce calibrated confidence estimates for gate input would provide a complementary signal to the feature-based routing used in this paper. The online adaptation mechanism warrants further investigation under adversarial workload shifts, including deliberate gaming of scheduling policies by tenants seeking priority placement, and the extension of GateLLM to multi-cluster and geo-distributed FaaS deployments represents a natural and important direction for scaling the framework to the full scope of modern cloud infrastructure.

# References


[1] Shahrad, M., Fonseca, R., Goiri, I., Chaudhry, G., Batum, P., Cooke, J., ... & Bianchini, R. (2020). Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In 2020 USENIX Annual Technical Conference (USENIX ATC 20) (pp. 205–218). USENIX Association.

[2] Agache, A., Brooker, M., Iordache, A., Liguori, A., Neugebauer, R., Piwonka, P., & Popa, D. M. (2020). Firecracker: Lightweight virtualization for serverless applications. In 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20) (pp. 419–434). USENIX Association.

[3] Kaffes, K., Yadwadkar, N. J., & Kozyrakis, C. (2021). Practical scheduling for real-world serverless computing. arXiv preprint arXiv:2111.07226.

[4] Das, A., Leaf, A., Varela, C. A., & Patterson, S. (2020). Skedulix: Hybrid cloud scheduling for cost-efficient execution of serverless applications. In 2020 IEEE 13th International Conference on Cloud Computing (CLOUD) (pp. 609–618). IEEE. https://doi.org/10.1109/CLOUD49709.2020.00093

[5] Ding, G., Yang, S., Lin, H., Chen, Z., & Yang, J. S. (2026). LLM-driven adaptive cloud resource scheduling: Bridging reasoning intelligence with optimization guarantees. IEEE Open Journal of the Computer Society. https://doi.org/10.1109/OJCS.2026.xxxxxx

[6] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., ... & Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. Advances in Neural Information Processing Systems, 35, 24824–24837.

[7] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2022). React: Synergizing reasoning and acting in language models. arXiv preprint arXiv:2210.03629.

[8] Schuster, T., Fisch, A., Gupta, J., Dehghani, M., Bahri, D., Tran, V., ... & Metzler, D. (2022). Confident adaptive language modeling. Advances in Neural Information Processing Systems, 35, 17456–17472.

[9] Chen, L., Zaharia, M., & Zou, J. (2023). Frugalgpt: How to use large language models while reducing cost and improving performance. arXiv preprint arXiv:2305.05176.

[10] Snell, C., Lee, J., Xu, K., & Kumar, A. (2024). Scaling llm test-time compute optimally can be more effective than scaling model parameters. arXiv preprint arXiv:2408.03314.

[11] Mao, H., Schwarzkopf, M., Venkatakrishnan, S. B., Meng, Z., & Alizadeh, M. (2019). Learning scheduling algorithms for data processing clusters. In Proceedings of the ACM SIGCOMM 2019 Conference (pp. 270–288). ACM. https://doi.org/10.1145/3341302.3342080

[12] Leviathan, Y., Kalman, M., & Matias, Y. (2023). Fast inference from transformers via speculative decoding. In International Conference on Machine Learning (pp. 19274–19286). PMLR.

[13] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M. A., Lacroix, T., ... & Lample, G. (2023). Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971.

[14] Fuerst, A., & Sharma, P. (2021). Faascache: keeping serverless computing alive with greedy-dual caching. In Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (pp. 386–400). ACM. https://doi.org/10.1145/3445814.3446758

[15] Tsenos, M., Peri, A., & Kalogeraki, V. (2023). Energy efficient scheduling for serverless systems. In 2023 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS) (pp. 27–36). IEEE. https://doi.org/10.1109/ACSOS58161.2023.00018

[16] Akbari, S., & Hauswirth, M. (2025). Hiku: Pull-based scheduling for serverless computing. In 2025 IEEE 25th International Symposium on Cluster, Cloud and Internet Computing (CCGrid) (pp. 450–461). IEEE.

[17] Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., ... & Lowe, R. (2022). Training language models to follow instructions with human feedback. Advances in Neural Information Processing Systems, 35, 27730–27744.

[18] Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., ... & Chen, W. (2022). Lora: Low-rank adaptation of large language models. In International Conference on Learning Representations.

[19] Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., ... & Clark, P. (2023). Self-refine: Iterative refinement with self-feedback. Advances in Neural Information Processing Systems, 36, 46534–46594.

[20] Kim, S., Mangalam, K., Moon, S., Malik, J., Mahoney, M. W., Gholami, A., & Keutzer, K. (2023). Speculative decoding with big little decoder. Advances in Neural Information Processing Systems, 36, 39236–39256.

[21] Wan, Z., Wang, X., Liu, C., Alam, S., Zheng, Y., Liu, J., ... & Zhang, M. (2023). Efficient large language models: A survey. arXiv preprint arXiv:2312.03863.

[22] Qiu, L. (2025). Reinforcement Learning Approaches for Intelligent Control of Smart Building Energy Systems with Real-Time Adaptation to Occupant Behavior and Weather Conditions. Journal of Computing and Electronic Information Management, 18(2), 32–37.

[23] Shen, Z., Zhao, W., Wang, B., Wang, Z., & Shang, W. (2026). CAGR: A Cross-Accelerator Graph Optimization Framework for Efficient Recommender System Inference. IEEE Access. https://doi.org/10.1109/ACCESS.2026.xxxxxx

[24] Ding, J., Shen, Z., & Liu, W. (2026). Game-Theoretic Cost-Sensitive Adversarial Training for Robust Cloud Intrusion Detection Against GAN-Based Evasion Attacks. Applied Sciences, 16(8), 3944. https://doi.org/10.3390/app16083944

[25] Ping, W., Jiao, Y., Fan, H., & Zhang, X. (2026). Multimodal Fraud Detection in Financial Statements: A Trimodal Attention Network with Contrastive Evidence Chain Construction. IEEE Access. https://doi.org/10.1109/ACCESS.2026.xxxxxx

[26] Rajbhandari, S., Rasley, J., Ruwase, O., & He, Y. (2020). Zero: Memory optimizations toward training trillion parameter models. In SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (pp. 1–16). IEEE. https://doi.org/10.1109/SC41405.2020.00018