

# Auto-Tuning Framework for Efficient Large Language Model Inference on Cloud Accelerators

Cornelis de Plantard

Department of Computer Science and Engineering, University of Nevada, Reno, Reno, NV,  
USA.

deplantard1986@unr.edu

Baghyawati M. Nayak

Department of Computer Science, Binghamton University, Binghamton, NY, USA.

baghyawati.nayak227@binghamton.edu

Cohen Boniface

Department of Computer Science, Colorado State University, Fort Collins, CO, USA.

bonifacecohen@colostate.edu

## Abstract

Large language model inference on cloud accelerators has become a central systems problem because it couples model architecture, hardware heterogeneity, serving policies, energy constraints, and organizational governance. This paper develops a system-level account of an auto-tuning framework for efficient inference in cloud environments. The framework is not treated merely as a collection of parameter search routines but as a socio-technical infrastructure that continuously observes workloads, predicts latency and cost behavior, selects execution configurations, and subjects those decisions to governance and policy constraints. The analysis begins with the architectural foundations of transformer-based inference and accelerator-oriented serving, including attention optimization, memory management, quantization, adapters, speculative decoding, and distributed inference. It then examines trade-offs among throughput, latency, accuracy, robustness, fairness, and sustainability. Particular attention is given to telemetry and machine-learning-based delay prediction as an input to tuning decisions, while avoiding the reduction of auto-tuning to prediction alone. The paper argues that efficient inference requires co-design across accelerator scheduling, model compression, request routing, multi-tenant isolation, observability, and carbon-aware deployment. It also considers deployment risks such as performance instability, unfair resource allocation, vendor lock-in, and hidden environmental costs. Case illustrations from inference serving systems, adaptive model serving, and distributed transformer serving are used to show how auto-tuning frameworks can be evaluated beyond benchmark accuracy. The conclusion proposes that future auto-tuning infrastructures should be auditable, accountable, and adaptive, with explicit metrics for robustness, fairness, energy intensity, and operational resilience rather than latency alone.

## Keywords

large language model inference; cloud accelerators; auto-tuning; systems architecture; governance; sustainability; robustness; fairness; infrastructure policy.

## 1. Introduction

The rapid adoption of large language models has transformed cloud infrastructure from a general-purpose compute substrate into a specialized inference environment where accelerator memory, interconnect bandwidth, batching policy, and request arrival patterns determine both service quality and operating cost. Transformer-based architectures [1] and scaling laws for few-shot learning [2] established that model capability can improve with parameter count and data scale, but they also created deployment regimes in which inference is no longer a secondary workload. Instead, inference has become a first-order systems concern that shapes data center design, procurement, scheduling, and sustainability reporting. An auto-tuning framework for inference must therefore be understood as more than a performance optimization layer; it is an institutional mechanism that translates service-level objectives into hardware actions under uncertainty.

The central problem is that efficient inference is not a single objective. A configuration that maximizes throughput may increase tail latency, degrade fairness across tenants, or concentrate load on a small number of accelerator types. A configuration that minimizes latency may waste memory bandwidth or energy. A configuration that reduces model precision may improve cost but introduce accuracy regressions for specific languages, domains, or user groups. Auto-tuning frameworks must navigate these tensions while operating in cloud environments where workloads are dynamic, hardware is heterogeneous, and policy constraints are imposed by both providers and customers. The literature on efficient transformer inference [3] and memory-efficient serving [5] shows that system-level decisions about attention, caching, and batching can dominate raw model-level improvements. This paper argues that an effective auto-tuning framework must be designed as a governed control plane rather than a standalone optimizer.

The socio-technical dimension is equally important. Cloud accelerators are not neutral resources; they are embedded in contractual relationships, regional energy markets, data sovereignty rules, and organizational risk cultures. A framework that tunes only for latency may unintentionally externalize costs onto users through unpredictable performance or onto the environment through carbon-intensive scheduling. Conversely, a framework that enforces strict sustainability constraints without considering service quality may fail operationally and be bypassed by engineering teams. The analysis that follows therefore integrates architecture, deployment, governance, robustness, fairness, and policy. It draws on inference serving systems, quantization and adaptation techniques, speculative decoding, distributed execution, and green computing research to outline a research agenda for auto-tuning as an accountable infrastructure.

## **2. Conceptual Foundations of Auto-Tuning for Cloud Accelerator Inference**

Auto-tuning for large language model inference begins with a distinction between static optimization and runtime adaptation. Static optimization occurs during model preparation, compiler selection, kernel generation, and deployment packaging. Runtime adaptation occurs when request patterns, tenant mixes, thermal conditions, or cost signals change. A cloud inference service may deploy a model that was compiled for a particular accelerator family, but the optimal batch size, sequence length distribution, cache policy, and parallelism strategy may shift across hours or minutes. The framework must therefore maintain a representation of the workload, a model of the execution environment, and a policy layer that determines when reconfiguration is worth the disruption. This is why inference delay prediction and performance modeling are relevant, but they are inputs to a broader control loop rather than substitutes for system design.

The conceptual foundation also includes the distinction between model-level efficiency and service-level efficiency. Model-level techniques such as quantization [7], post-training quantization [8], low-rank adaptation [9], quantized fine-tuning [10], and smooth activation scaling [11] reduce memory footprint and compute demand. Service-level techniques such as paged memory management [5], speculative decoding [12], and optimized attention kernels [4] improve the way requests are scheduled and executed. A framework that tunes only one layer can produce misleading gains. For example, aggressive quantization may reduce memory pressure but increase decoding variance, while speculative decoding may improve interactive latency but consume additional compute under high concurrency. Auto-tuning must therefore evaluate configurations as compositions, not as isolated switches.

Cloud accelerators add another layer of complexity because the available hardware is rarely uniform. A provider may offer graphics processing units, tensor processing units, inference accelerators, and custom application-specific integrated circuits across regions and generations. Each device family has different memory hierarchies, interconnect topologies, numerical formats, and scheduling constraints. Distributed inference frameworks such as DeepSpeed-Inference [14] demonstrate that model partitioning and communication overlap are essential for large models, but the best partition depends on the accelerator topology and the request workload. An auto-tuning framework must abstract these details without hiding them completely, because governance requires visibility into which resources are used, why they are selected, and what trade-offs are being accepted.

### **3. Architectural and Resource Trade-Offs**

The architectural core of auto-tuning is the management of memory, compute, and communication. Transformer inference is often memory-bound during decoding because each generated token requires access to model weights and key-value caches. PagedAttention [5] addresses this by managing cache memory in blocks, reducing fragmentation and enabling more effective batching. Attention kernel improvements such as FlashAttention [4] reduce memory traffic by reorganizing computation around input and output awareness. These techniques show that auto-tuning cannot be reduced to selecting a model size or a batch size; it must reason about data movement, cache residency, and kernel-level scheduling. A framework that ignores memory hierarchy may select configurations that look efficient in arithmetic operations but perform poorly in wall-clock latency.

Quantization and adaptation introduce distinct trade-offs. Eight-bit matrix multiplication [7] and post-training quantization [8] can reduce memory and compute costs, but they may require calibration data and careful handling of outliers. Low-rank adaptation [9] enables parameter-efficient specialization, while quantized fine-tuning [10] allows larger models to be adapted under memory constraints. SmoothQuant [11] shifts quantization difficulty from activations to weights, improving robustness for large models. In an auto-tuning framework, these techniques are not merely model compression options; they are policy variables that affect accuracy, fairness, and update velocity. A tenant serving many languages may need different quantization profiles than a tenant serving a narrow domain. The framework must therefore support per-workload configuration while preventing configuration sprawl that becomes unauditabile.

Decoding strategies create additional architectural trade-offs. Speculative decoding [12] and fast transformer decoding [13] can reduce latency by predicting multiple tokens or reorganizing attention computation. However, speculative methods may increase compute utilization and energy consumption when acceptance rates are low. Distributed serving

systems such as Orca [19] and AlpaServe [18] show that continuous batching, model parallelism, and statistical multiplexing can improve aggregate throughput, but they also introduce scheduling complexity and potential unfairness across tenants. An auto-tuning framework must evaluate these techniques together. A configuration that improves average latency may worsen tail latency for small tenants or increase preemption risk for long-running requests. The framework should therefore expose multi-objective controls rather than a single efficiency score.

#### **4. Telemetry, Prediction, and Governance in Auto-Tuning Pipelines**

Telemetry is the connective tissue of auto-tuning. Without accurate observation of request arrival rates, sequence lengths, cache hit behavior, accelerator utilization, memory pressure, and error rates, the framework cannot distinguish between a good configuration and a temporarily lucky one. Machine-learning-based delay prediction [6] can improve scheduling by anticipating latency under different load conditions, but prediction models must be integrated with governance mechanisms. A prediction that a configuration will be fast is not sufficient if it violates tenant isolation, exceeds a carbon budget, or depends on hardware that is unavailable in a required jurisdiction. The framework should treat prediction as one signal among many, with explicit uncertainty estimates and fallback policies.

Governance enters through the definition of objectives, constraints, and audit trails. Who decides whether energy efficiency outweighs latency? How are fairness guarantees encoded when multiple tenants share accelerators? What happens when a tuning action degrades performance for a minority workload? These questions cannot be resolved by optimization alone. They require policy layers that specify acceptable ranges, escalation paths, and transparency requirements. For example, a cloud provider may allow aggressive batching for best-effort workloads but require strict latency isolation for interactive applications. An auto-tuning framework should record the rationale for each configuration change, the observed outcome, and the affected tenants. This record becomes essential for incident review, regulatory compliance, and continuous improvement.

The governance challenge is amplified by multi-tenancy. Inference serving systems such as Clipper [15], Clockwork [16], and INFaaS [17] demonstrate that serving infrastructure must handle variation in model type, latency targets, and resource requirements. When auto-tuning operates across tenants, it can inadvertently create feedback loops in which popular workloads receive more optimization attention while rare workloads are underserved. A fair framework should include workload-aware budgeting, minimum performance guarantees, and periodic audits for disparity. It should also distinguish between tuning for provider profit and tuning for user welfare. Without such distinctions, auto-tuning becomes a hidden policy instrument that shapes access to computational capability.

#### **5. Deployment, Robustness, and Fairness Across Cloud Infrastructures**

Deployment is where auto-tuning meets operational reality. A configuration that performs well in a controlled benchmark may fail under production traffic because of burstiness, noisy neighbors, thermal throttling, or software version drift. Robustness requires redundancy in tuning decisions. The framework should maintain multiple candidate configurations, monitor for distribution shift, and revert quickly when performance or accuracy degrades. It should also test configurations under adversarial or unusual conditions, such as very long prompts, multilingual inputs, repeated prefixes, or sudden changes in tenant mix. Robustness is not

merely an engineering property; it is a governance commitment to predictable service under stress.

Fairness in auto-tuning is often overlooked because efficiency is treated as a universal good. In practice, efficiency gains may be unevenly distributed. A framework that optimizes for average throughput may allocate more memory to large tenants and reduce cache availability for small tenants. A framework that routes requests to the lowest-cost accelerator may send users in certain regions to older hardware with higher latency. A framework that compresses models aggressively may reduce accuracy for dialects or domains with less representation in calibration data. Fairness requires explicit metrics, such as performance parity across tenant classes, accuracy parity across language groups, and error rate parity across request types. Auto-tuning should not be allowed to silently trade fairness for aggregate efficiency.

Cloud infrastructure also introduces dependency and lock-in risks. Auto-tuning frameworks often rely on vendor-specific profilers, compilers, and accelerator libraries. If the framework encodes too much hardware-specific knowledge, it may become brittle when the provider introduces new instances or changes pricing. A more resilient design separates portable policy from device-specific implementation. The policy layer defines objectives, constraints, and acceptable trade-offs, while device adapters implement tuning actions for particular accelerators. This separation supports portability, auditability, and competition among hardware providers. It also makes it easier to incorporate new accelerator types without rewriting the entire control plane.

## **6. Sustainability, Economics, and Policy Implications**

Sustainability is a first-class concern for large language model inference because inference occurs continuously and at scale. Training has received substantial attention for its energy and policy implications [20], and green computing research has argued for efficiency metrics that go beyond raw performance [21]. Inference auto-tuning can contribute to sustainability by reducing wasted computation, improving accelerator utilization, and selecting configurations that match workload demand. However, it can also increase energy consumption if it encourages speculative decoding, redundant model replicas, or always-on maximum-performance settings. A sustainable framework should incorporate energy intensity, carbon intensity of the regional grid, and embodied hardware costs into its objective functions. It should also distinguish between energy saved per request and total energy consumed by the service.

The economics of auto-tuning are similarly multi-dimensional. Cloud providers charge for accelerator time, memory, storage, and data transfer, while users experience cost through latency, reliability, and accuracy. A framework that minimizes provider cost may increase user cost if it shifts computation to client-side preprocessing or reduces output quality. A framework that minimizes user latency may increase provider cost by overprovisioning. Economic governance therefore requires transparent accounting. The framework should expose cost attribution by tenant, workload, and configuration, and it should allow policy constraints such as maximum cost per thousand tokens or maximum carbon per request. These constraints turn auto-tuning into a negotiated infrastructure rather than an opaque optimization.

Policy implications extend beyond internal governance. Regulators and standards bodies may increasingly require disclosure of computational intensity, energy use, and performance discrimination in automated systems. An auto-tuning framework that can document its

decisions, constraints, and outcomes is better positioned for compliance. It can also support public-interest research by providing anonymized telemetry and configuration performance data. At the same time, transparency must be balanced against security and proprietary information. The policy design should therefore specify what must be disclosed, what may be aggregated, and what must remain protected. The goal is not radical transparency for its own sake but accountable efficiency that can be examined by affected parties.

## **7. Case Illustrations and Cross-Domain Comparisons**

Several inference serving systems illustrate the value of auto-tuning as a control plane. Clipper [15] showed that a general-purpose prediction serving system can abstract model execution and apply latency-aware policies. Clockwork [16] emphasized performance predictability from the bottom up, arguing that serving systems should provide stable latency rather than only high average throughput. INFaaS [17] automated model selection and configuration, demonstrating that inference serving can benefit from model-less abstractions that hide implementation details from users. These systems predate the current scale of large language models, but their lessons about abstraction, predictability, and automation remain directly relevant. An auto-tuning framework for large models should inherit their emphasis on service-level objectives rather than model-level benchmarks.

More recent systems address transformer-specific challenges. Orca [19] introduced continuous batching and iteration-level scheduling for generative models, improving accelerator utilization under dynamic request patterns. AlpaServe [18] used statistical multiplexing and model parallelism to serve many models efficiently, showing that placement and parallelism decisions can be automated. DeepSpeed-Inference [14] demonstrated large-scale distributed inference with optimized kernels and communication overlap. These systems illustrate that auto-tuning must span scheduling, partitioning, and kernel selection. A framework that tunes only one of these layers may miss cross-layer interactions that dominate end-to-end performance. The case for an integrated framework is therefore empirical as well as conceptual.

Cross-domain comparisons with database query optimization and network traffic engineering are instructive. Database optimizers select execution plans based on statistics and cost models, but they also provide hints, plan stability, and workload management. Network traffic engineering balances utilization and fairness through admission control, pacing, and routing. Auto-tuning for inference can borrow these ideas by treating configuration selection as plan optimization, tenant isolation as admission control, and model routing as traffic steering. However, inference differs because model accuracy, language coverage, and safety behavior can change with configuration. The framework must therefore include quality-of-service dimensions that are not present in traditional query optimization or packet routing.

## **8. Future Directions and Research Agenda**

Future research should advance three interconnected areas. The first is multi-objective optimization under uncertainty. Auto-tuning frameworks need methods that can reason about latency, throughput, accuracy, fairness, energy, and cost without collapsing them into a single scalar. They also need uncertainty quantification so that decisions are robust to noisy telemetry and unpredictable workloads. The second area is governance-aware automation. Frameworks should support policy specification, audit trails, and contestability, allowing affected users or administrators to understand and challenge tuning decisions. The third area

is sustainable infrastructure co-design. Accelerators, data center cooling, power markets, and inference schedulers should be studied together rather than as separate layers.

A research agenda should also address evaluation. Benchmarks for auto-tuning must move beyond static datasets and average latency. They should include bursty traffic, mixed tenants, multilingual workloads, long-context requests, hardware failures, and carbon-aware scheduling. They should measure tail latency, fairness gaps, energy per request, and accuracy regressions under compression. Comparisons between frameworks should report not only best-case performance but also stability, recovery time, and operational complexity. Without such evaluation, auto-tuning risks becoming a collection of impressive demonstrations rather than a dependable infrastructure capability.

## 9. Conclusion

Auto-tuning for large language model inference on cloud accelerators is fundamentally a systems and governance problem. The technical layers include attention optimization, memory management, quantization, adaptation, speculative decoding, distributed execution, and request scheduling. The socio-technical layers include tenant fairness, robustness, sustainability, cost attribution, policy compliance, and public accountability. Efficient inference cannot be achieved by optimizing one layer in isolation. It requires a control plane that observes workloads, predicts performance, selects configurations, enforces constraints, and records outcomes. The framework proposed in this paper treats auto-tuning as an accountable infrastructure rather than a hidden optimizer.

The path forward requires collaboration among systems researchers, machine learning engineers, cloud providers, policymakers, and affected communities. Auto-tuning frameworks should be designed to expose trade-offs, support contestability, and align efficiency with broader public values. They should reduce waste, improve access, and protect against unfair performance degradation. They should also remain adaptable as models, accelerators, energy markets, and regulations change. If designed with these principles, auto-tuning can become a foundational capability for responsible and efficient large language model inference in cloud environments.

## References

1. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 5998–6008.
2. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... Amodei, D. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33, 1877–1901.
3. Pope, R., Douglas, S., Chowdhery, A., Devlin, J., Bradbury, J., Heek, J., ... Dean, J. (2023). Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5, 606–624.
4. Dao, T., Fu, D. Y., Ermon, S., Rudra, A., & Ré, C. (2022). FlashAttention: Fast and memory-efficient exact attention with IO-awareness. *Advances in Neural Information Processing Systems*, 35, 16344–16359.

5. Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., ... Zhang, H. (2023). Efficient memory management for large language model serving with PagedAttention. *Proceedings of the 29th Symposium on Operating Systems Principles*, 611–626.
6. Zhang, J., & Gao, H. (2026, March). LLM inference delay prediction based on machine learning algorithms. In *2026 International Conference on Generative Artificial Intelligence and Information Security (GAIIS)* (pp. 37-41). IEEE.
7. Dettmers, T., Lewis, M., Belkada, Y., & Zettlemoyer, L. (2022). LLM.int8(): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35, 30318–30332.
8. Frantar, E., Ashkboos, S., Hoefler, T., & Alistarh, D. (2023). GPTQ: Accurate post-training quantization for generative pre-trained transformers. *The Eleventh International Conference on Learning Representations*.
9. 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. *International Conference on Learning Representations*.
10. Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). QLoRA: Efficient finetuning of quantized LLMs. *Advances in Neural Information Processing Systems*, 36, 10088–10115.
11. Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., & Han, S. (2023). SmoothQuant: Accurate and efficient post-training quantization for large language models. *Proceedings of the 40th International Conference on Machine Learning*, 202, 38087–38099.
12. Leviathan, Y., Kalman, M., & Matias, Y. (2023). Fast inference from transformers via speculative decoding. *Proceedings of the 40th International Conference on Machine Learning*, 202, 19274–19286.
13. Shazeer, N. (2019). Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*.
14. Aminabadi, R. Y., Rajbhandari, S., Awan, A. A., Li, C., Li, D., Zheng, E., ... He, Y. (2022). DeepSpeed-Inference: Enabling efficient inference of transformer models at unprecedented scale. *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–15.
15. Crankshaw, D., Wang, X., Zhou, G., Franklin, M. J., Gonzalez, J. E., & Stoica, I. (2017). Clipper: A low-latency online prediction serving system. *14th USENIX Symposium on Networked Systems Design and Implementation*, 613–627.
16. Gujarati, A., Karimi, R., Alzayat, S., Hao, W., Kaufmann, A., Vigfusson, Y., & Mace, J. (2020). Serving DNNs like clockwork: Performance predictability from the bottom up. *14th USENIX Symposium on Operating Systems Design and Implementation*, 443–462.
17. Romero, F., Li, Q., Yadwadkar, N. J., & Kozyrakis, C. (2021). INFaaS: Automated model-less inference serving. *2021 USENIX Annual Technical Conference*, 397–411.
18. Li, Z., Zheng, L., Zhong, Y., Liu, V., Sheng, Y., Jin, X., ... Gonzalez, J. E. (2023). AlpaServe: Statistical multiplexing with model parallelism for deep learning serving. *17th USENIX Symposium on Operating Systems Design and Implementation*, 663–679.

19. Yu, G.-I., Jeong, J. S., Kim, G.-W., Kim, S., & Chun, B.-G. (2022). Orca: A distributed serving system for transformer-based generative models. 16th USENIX Symposium on Operating Systems Design and Implementation, 521–538.
20. Strubell, E., Ganesh, A., & McCallum, A. (2019). Energy and policy considerations for deep learning in NLP. Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, 3645–3650.
21. Schwartz, R., Dodge, J., Smith, N. A., & Etzioni, O. (2020). Green AI. Communications of the ACM, 63(12), 54–63.
22. Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., ... Sifre, L. (2022). Training compute-optimal large language models. Advances in Neural Information Processing Systems, 35, 30016–30030.