

Queueing-Aware Latency Prediction for Scalable Large Language Model Applications

Gianfranco Interiano

School of Electrical Engineering and Computer Science, Oregon State University, Corvallis,
OR, USA.

contactgianfranco@oregonstate.edu

Abstract

Large language model applications increasingly operate as shared, multi-tenant services in which response quality depends not only on model accuracy but on the timing of computation, memory movement, scheduling, and network exchange. Latency prediction for such systems cannot be reduced to isolated accelerator benchmarks, because the observed response time emerges from dynamic queues that form across gateways, batching schedulers, key-value caches, inference workers, autoscalers, and downstream tools. This paper develops a queueing-aware perspective on latency prediction for scalable large language model applications. It synthesizes classical queueing theory with contemporary inference serving architectures, discussing how batching, continuous scheduling, paged attention, disaggregated execution, and autoscaling create structural trade-offs among throughput, tail latency, cost, and fairness. The paper argues that prediction systems must be treated as socio-technical infrastructure rather than purely statistical components. Accurate forecasts require observability across layers, governance over service-level objectives, robust treatment of workload drift, and explicit attention to sustainability and equitable access. It also examines how queueing-aware prediction can improve admission control, routing, capacity planning, and user-facing communication while introducing risks of feedback loops, manipulation, and uneven service quality. The conclusion outlines research directions for hybrid mechanistic and machine-learning models, policy-aware service management, and resilient deployment in cloud-edge environments.

Keywords

large language models, queueing theory, latency prediction, inference serving, scalability, tail latency, governance, sustainability.

1. Introduction

Large language model applications have moved from experimental demonstrations to production infrastructure, where they mediate search, software development, customer service, education, scientific analysis, and public administration. In these settings, the practical utility of a model is shaped not only by its generative quality but by whether a response arrives within a time window that users, workflows, and downstream systems can tolerate. A conversational assistant may require interactive responsiveness, an automated code review system may operate under batch deadlines, and a real-time decision-support tool may require bounded delay to preserve safety and situational awareness. This diversity means that latency is not a single metric but a socio-technical property that connects infrastructure capacity, service-level agreements, user expectations, and organizational accountability. Classical queueing theory offers a foundational vocabulary for this problem, because it explains how delay arises when demand arrives randomly and service capacity is finite [1]. Early stochastic

treatments of queues established that waiting time and system occupancy are not incidental byproducts of computation but structural consequences of variability and scheduling [2]. Performance modeling for computer systems later translated these insights into practical methods for capacity planning, contention analysis, and service-level design [3].

The central problem addressed in this paper is that large language model latency prediction is often treated as a purely predictive modeling task, in which historical traces are fed into a regression or machine learning pipeline to estimate future response times. Such approaches can be useful, but they risk missing the generative mechanisms that produce latency. Modern inference systems create queues at many points, including API gateways, request routers, tokenization stages, batching queues, memory managers, accelerator schedulers, network transports, and external tool endpoints. When these queues interact, small changes in arrival patterns, model configuration, sequence length, cache hit rate, or tenant priority can produce large changes in tail latency. The tail at scale problem is especially relevant, because rare but severe delays can dominate user experience and can propagate through distributed systems [4]. A queueing-aware approach therefore asks not only what latency has been observed but why delay emerges from the architecture and how operational decisions alter the distribution of waiting times.

This paper argues that scalable large language model applications require latency prediction systems that are mechanistically informed, operationally integrated, and governance-aware. Mechanistic awareness means that statistical models should be conditioned on structural features such as queue depth, batch composition, memory pressure, token generation phase, and autoscaling state. Operational integration means that predictions must be connected to admission control, routing, batching, and capacity management rather than serving as passive dashboards. Governance awareness means that latency management distributes access, cost, and inconvenience across users and institutions, and therefore raises questions about fairness, transparency, sustainability, and policy. By connecting queueing theory, inference serving architecture, machine learning prediction, and socio-technical governance, the paper develops a systems-level account of latency prediction for scalable language model services.

2. Foundations of Queueing-Aware Latency Prediction

Queueing theory begins from the observation that service systems rarely operate under perfectly smooth demand. Requests arrive with variability, service times vary across tasks, and limited resources must be shared. Even when average utilization appears reasonable, variability can create congestion and long waits. Foundational work in stochastic processes and queueing analysis formalized how arrival patterns, service discipline, and server count influence waiting time and queue length [2]. Later computer systems research adapted these ideas to interactive and distributed environments, showing that performance modeling can guide architectural choices before deployment and can diagnose bottlenecks after deployment [3]. For large language models, these classical insights remain relevant because inference is not a uniform service. A short prompt with a small output may complete quickly, while a long-context request with many generated tokens may occupy memory and accelerator time for a much longer interval. The service time distribution is therefore heavy and context-dependent, which makes average-case planning insufficient.

A queueing-aware latency prediction system must distinguish between queueing delay, service delay, and propagation delay. Queueing delay is the time a request waits before receiving resources. Service delay is the time spent actively processing the request, including prefill, decode, memory movement, and sampling. Propagation delay includes network

transmission, gateway overhead, and communication with external tools or databases. In large language model services, these categories overlap because continuous batching can interleave prefill and decode phases, and because token generation may be paused or resumed depending on scheduler decisions. Memory management also affects service delay: if key-value cache blocks are fragmented or evicted, the system may need to recompute or reload state, increasing service time and creating additional queue pressure. Thus, latency prediction cannot be separated from the resource management policies that determine when requests run, how long they run, and what data remains resident.

Classical queueing models often assume stationary arrival processes and stable service distributions. Production language model workloads violate these assumptions in important ways. Traffic may spike after a product announcement, during business hours, or in response to an external event. Workloads may shift from short chat prompts to long document analysis tasks. Model updates may alter service times, and new safety filters may add preprocessing or postprocessing stages. Autoscaling introduces additional dynamics, because capacity changes occur with delay and may lag behind demand. A prediction model that ignores these non-stationarities may perform well on historical averages while failing during the periods when prediction matters most. Queueing-aware prediction therefore needs to represent time-varying demand, transient congestion, and feedback loops between prediction, admission control, and routing. If the system rejects or reroutes requests based on predicted latency, it changes the very workload that future predictions must estimate.

The value of queueing awareness also lies in its ability to support counterfactual reasoning. A purely observational predictor may estimate that a request will take a certain time under current conditions, but it may not explain how latency would change if the request were routed to another region, delayed until a batch completes, assigned a smaller model, or given lower priority. Queueing models provide a language for such counterfactuals by linking arrivals, service capacity, scheduling discipline, and wait times. In large language model serving, this counterfactual capacity is essential for autoscaling, load shedding, and quality-of-service differentiation. It also supports governance, because operators can ask whether a latency target is achievable for all users or only for privileged tenants. Without a queueing-aware frame, latency prediction risks becoming a narrow forecasting exercise that cannot guide architectural or policy decisions.

3. Architectural Sources of Latency Variation in Large Language Model Serving

Contemporary large language model serving systems are complex pipelines rather than monolithic models. Requests pass through tokenization, scheduling, prefill, decode, sampling, detokenization, safety filtering, and often external tool calls. Each stage can introduce variability, and the interactions among stages determine end-to-end latency. Systems such as Orca introduced continuous batching and iteration-level scheduling to improve accelerator utilization by mixing requests at different generation stages [6]. This design improves throughput but also complicates latency prediction, because a request's completion time depends on which other requests share its batch and how many tokens each will generate. PagedAttention and related memory management techniques further improve efficiency by allowing key-value cache blocks to be allocated non-contiguously, but they also create new sources of variation related to block allocation, eviction, and memory pressure [5]. A queueing-aware predictor must therefore treat batching and memory management as first-class explanatory variables rather than hidden implementation details.

Scaling transformer inference introduces additional structural trade-offs. Efficient inference depends on parallelism across layers, attention heads, sequence positions, and batch elements, and the optimal strategy changes with model size, sequence length, hardware topology, and network bandwidth [7]. DeepSpeed-Inference and similar systems demonstrate that inference can be distributed across devices to reduce per-request latency or increase throughput, but distribution also adds communication overhead and synchronization delays [8]. When requests are replicated across regions or clusters, the routing layer becomes a queueing network in its own right. Some replicas may be warm and lightly loaded, while others may be cold, overloaded, or experiencing degraded network conditions. A latency predictor that ignores replica-level queue state may route requests poorly, creating hotspots that worsen tail latency. Queueing-aware prediction can help balance load by estimating not only average service time but also the probability that a request will encounter a long queue or a slow path.

Programmatic and structured use of language models adds another layer of complexity. Modern applications often execute chains of prompts, tool calls, validations, and conditional branches. Systems such as SGLang aim to optimize structured language model programs by improving execution efficiency and reducing redundant computation [9]. From a latency perspective, these programs resemble workflows with multiple queueing stages, where the output of one stage becomes the input to another. A delay in an early stage can propagate through the entire workflow, and a rare slow tool call can dominate end-to-end response time. Prediction systems must therefore model not only individual model invocations but also dependency graphs and retry behavior. If a downstream tool times out and the request is retried, the effective arrival rate at the model service increases, producing congestion that may not have been anticipated by per-request benchmarks.

Serving infrastructure also shapes latency through resource abstraction and scheduling. Inference serving platforms such as Clipper explored low-latency prediction serving with caching, batching, and model selection [10]. Clockwork emphasized performance predictability by improving scheduling and reducing interference at the bottom of the stack [11]. INFaaS demonstrated automated model selection and provisioning as a way to meet latency and cost objectives across heterogeneous models [13]. These systems show that latency is produced by policy choices about when to batch, when to cache, when to scale, and which model variant to use. For large language models, similar choices arise with quantized models, adapter modules, speculative decoding, and retrieval augmentation. A queueing-aware predictor should therefore include configuration and policy features, because the same request can have very different latency profiles depending on the serving plan selected by the platform.

4. Prediction Models and Their Operational Integration

Machine learning approaches to latency prediction can capture complex relationships that are difficult to specify manually. They can learn from traces, metrics, logs, and system state to estimate response times under changing conditions. Recent work has explored machine learning algorithms for large language model inference delay prediction, showing that data-driven models can provide useful forecasts when engineered features reflect workload and system behavior [12]. Such approaches are valuable because they can adapt to nonlinear interactions among prompt length, output length, batch size, hardware type, and queue depth. However, their operational value depends on how they are integrated into control loops. A prediction that is accurate in offline evaluation but unavailable during overload is not useful. A prediction that is used to reject requests without transparency can create fairness and

accountability problems. A prediction that creates feedback loops may destabilize the system if it causes synchronized routing or autoscaling decisions.

Operational integration begins with observability. Queueing-aware prediction requires telemetry from multiple layers: request arrival rates, queue lengths, batch sizes, token generation rates, cache hit ratios, memory utilization, accelerator utilization, network latency, error rates, and autoscaling events. It also requires labels or outcomes that can be used for training, such as actual completion times and timeout events. Because large language model services are often distributed across cloud regions and edge locations, telemetry must be aggregated without introducing unacceptable monitoring overhead. Federated or hierarchical monitoring architectures may be appropriate when data cannot leave a jurisdiction or when tenants require isolation. The design challenge is to collect enough information for prediction while preserving privacy, security, and tenant confidentiality.

Once predictions are available, they can support several control functions. Admission control can compare predicted latency against service-level objectives and decide whether to accept, delay, or reject a request. Routing can send requests to replicas, regions, or model variants that are likely to meet latency targets. Batching can adjust batch size and composition to balance throughput against tail latency. Autoscaling can pre-provision capacity before predicted demand materializes. Caching and retrieval systems can be tuned based on predicted reuse. These functions are not independent. Admission control changes arrival rates, routing changes queue lengths, batching changes service times, and autoscaling changes capacity. A queueing-aware prediction system must therefore be evaluated as part of a control system, not as an isolated forecasting model. Stability, convergence, and robustness become central concerns.

There is also a distinction between descriptive, predictive, and prescriptive latency analytics. Descriptive analytics report what happened, predictive analytics estimate what will happen, and prescriptive analytics recommend what should be done. Queueing-aware prediction can bridge these categories by combining mechanistic models with learned components. For example, a mechanistic queueing model can represent fundamental relationships among arrival rate, service capacity, and waiting time, while a machine learning component can estimate service-time distributions from prompt and system features. This hybrid approach can improve extrapolation during unusual conditions and can make predictions more interpretable to operators. It can also support scenario analysis, allowing decision makers to ask how latency would change under different capacity, routing, or pricing policies. The goal is not to replace queueing theory with machine learning or machine learning with queueing theory, but to use each where it is strongest.

5. Trade-offs in Scalability, Cost, and Quality of Service

Scalable large language model applications face a fundamental tension among throughput, latency, cost, and quality. Larger batches improve accelerator utilization and reduce cost per token, but they can increase waiting time for individual requests. Larger models may produce higher-quality outputs, but they require more memory and computation, which increases service time and reduces the number of concurrent requests a replicator can handle. Longer context windows improve task performance for document analysis and retrieval, but they increase prefill time and key-value cache pressure. Quantization and distillation reduce resource requirements, but they may affect output quality or require additional validation. Queueing-aware latency prediction helps make these trade-offs explicit, because it connects

configuration choices to the distribution of user-visible delays rather than to isolated benchmark averages.

The tail latency problem is particularly important in multi-tenant services. A small fraction of requests may experience very long waits because they arrive during a burst, require long generation, or encounter a slow downstream tool. In a queueing network, these slow requests can occupy resources and delay subsequent requests, creating cascading congestion. The tail at scale literature emphasizes that rare delays can be amplified by fan-out and retries [4]. In language model applications, fan-out may occur when a single user request triggers multiple model calls, retrieval queries, or tool invocations. Retries may occur when timeouts are exceeded or when safety filters require additional review. A queueing-aware predictor should therefore estimate not only median latency but also high-percentile latency and timeout probability. Operators can then design admission control and routing policies that protect interactive users while allowing batch workloads to use spare capacity.

Cost management is closely related to latency. Cloud inference is often priced by compute time, token count, or reserved capacity. Overprovisioning reduces queueing delay but increases idle cost. Underprovisioning reduces cost but increases waiting time and timeout risk. Autoscaling seeks a middle ground, but it is constrained by startup time, model loading, warm-up effects, and regional capacity limits. Queueing-aware prediction can improve autoscaling by forecasting demand and by estimating how much capacity is needed to meet a latency objective at a given percentile. It can also support differentiated service levels, where some users pay for lower latency and others accept slower batch processing. However, such differentiation raises fairness concerns. If premium users receive priority, ordinary users may experience systematically worse queues, even when total capacity is adequate. Governance must therefore consider whether latency is treated as a public service quality, a market good, or a combination of both.

Quality of service also depends on robustness. Language model services may experience hardware failures, network partitions, model regressions, adversarial inputs, and unexpected workload shifts. A prediction system trained on stable conditions may fail during such events. Queueing-aware design can improve robustness by monitoring queue stability and by using fallback strategies such as smaller models, cached responses, or degraded modes. These fallbacks change the service-time distribution and may alter user expectations. Transparency is important: users should know when they are receiving a degraded response or when latency is unusually high because of capacity constraints. Hidden degradation can erode trust, especially in high-stakes domains such as healthcare, finance, or public safety. Thus, scalability is not only a matter of adding servers; it is a matter of designing predictable, governable, and communicable service behavior.

6. Governance, Fairness, and Policy Implications

Latency prediction and management are governance issues because they determine who waits, who receives priority, and which applications remain feasible. In many organizations, service-level objectives are treated as technical targets, but they also encode policy choices. A target of sub-second response time may be essential for interactive accessibility, while a multi-second target may be acceptable for batch analytics. If these targets differ across user groups, then latency becomes a dimension of equity. Queueing-aware prediction can reveal whether certain tenants, regions, languages, or request types systematically experience longer delays. For example, long-context requests in underrepresented languages may require more computation if tokenization is less efficient, and users on slower networks may experience

higher propagation delay. Without disaggregated latency analysis, such disparities can remain invisible.

Governance also concerns transparency and contestability. If an application rejects or delays a request based on predicted latency, users and auditors may need to understand why. A prediction system that relies on opaque machine learning models may be difficult to explain, especially when predictions interact with admission control or pricing. Hybrid models that expose queueing relationships can support explanation by showing how arrival rate, capacity, and priority affect expected delay. Even then, explanation must be careful not to reveal sensitive infrastructure details or enable adversarial manipulation. Policy frameworks for artificial intelligence increasingly emphasize accountability, fairness, and human oversight [19]. Latency governance should be integrated into these frameworks rather than treated as a purely operational concern. Service-level agreements, incident reports, and audit logs should include latency distributions and priority decisions, not only average performance.

Sustainability adds another policy dimension. Training and serving large language models consume significant energy and water, and latency optimization can either reduce or increase environmental impact depending on how it is pursued [16]. Aggressive overprovisioning may lower latency but waste resources during idle periods. Energy-aware scheduling may shift batch workloads to times and regions with cleaner electricity, potentially increasing latency for non-urgent tasks. Green AI research argues for efficiency metrics that go beyond accuracy and consider computational cost [17]. Queueing-aware prediction can support sustainability by matching capacity to demand, reducing unnecessary replication, and enabling flexible service tiers. However, it can also enable rebound effects if lower latency encourages more usage and higher total consumption. Sustainable design therefore requires system-level accounting that includes energy, carbon, water, hardware lifecycle, and user value.

Fairness and sustainability intersect with concerns about concentration of computational power. If low-latency large language model services require expensive accelerators and global network infrastructure, then only well-resourced organizations may be able to provide them reliably. Edge and federated approaches can distribute inference and reduce dependence on centralized clouds, but they introduce new challenges in model consistency, privacy, and coordination [20]. Federated learning research has explored ways to train models across distributed devices without centralizing data, and similar ideas can inform inference architectures that place computation closer to users. Yet edge deployment may also produce uneven quality if devices have different capabilities. Policy responses might include public compute infrastructure, open model standards, interoperability requirements, and procurement rules that prioritize accessibility. Queueing-aware latency prediction cannot solve these issues alone, but it can make their technical manifestations visible and measurable.

7. Deployment, Robustness, and Sustainability

Deployment of queueing-aware latency prediction requires careful engineering across the software lifecycle. Models must be trained on representative data, validated under stress, deployed with monitoring, and updated as workloads and architectures change. Hidden technical debt in machine learning systems often arises from entangled data dependencies, configuration drift, and feedback loops [15]. In latency prediction, technical debt can appear when features are computed differently in training and serving, when telemetry pipelines drop data during overload, or when prediction models are not retrained after hardware or model changes. Robust deployment therefore requires versioning, canary testing, and fallback policies. If the prediction service itself becomes a bottleneck or fails, the system must be able

to operate with conservative defaults. A queueing-aware design should treat the predictor as part of the critical path only when its reliability and value justify that placement.

Robustness also depends on handling distribution shift. Workloads change as users adopt new features, as model providers release updates, and as external events alter demand. A prediction model trained on last month's traffic may underestimate latency for a new long-context summarization feature or overestimate latency after an optimization update. Continuous monitoring and drift detection are therefore essential. Mechanistic components can help because relationships such as the effect of queue length on waiting time are more stable than raw statistical correlations. Machine learning components can help because they capture complex interactions that mechanistic models omit. A hybrid architecture can combine both, using queueing models as a prior and learned models to correct for residuals. This approach aligns with broader trends in performance modeling and machine learning for systems, where interpretability and adaptability are both valued.

Sustainability in deployment involves both operational efficiency and lifecycle planning. Inference efficiency can be improved through better batching, memory management, quantization, and compiler optimization [14]. These techniques reduce energy per request, but they must be evaluated in conjunction with latency and quality. A quantized model may reduce service time but require additional retries or human review if quality degrades. A compiler optimization may improve average throughput but worsen tail latency for certain sequence lengths. Queueing-aware prediction can help evaluate such trade-offs by estimating how changes affect the full latency distribution and how they interact with capacity. Sustainability also requires planning for hardware refresh, e-waste, and regional energy constraints. Latency objectives should not be pursued in ways that externalize environmental costs onto communities or future users.

Finally, deployment must consider resilience and graceful degradation. Large language model services are increasingly embedded in critical workflows, and outages can have significant consequences. Queueing-aware prediction can support resilience by identifying impending congestion before it becomes failure, by enabling load shedding that preserves essential functions, and by routing around degraded regions. However, resilience policies must be governed by clear priorities. In a hospital or emergency response setting, preserving low latency for critical queries may justify delaying non-urgent requests. In a commercial setting, priorities may be defined by contracts or fairness commitments. The technical design of admission control and routing is therefore inseparable from policy design. Sustainable and fair deployment requires that these choices be made explicitly, documented, and reviewed rather than emerging accidentally from short-term optimization.

8. Conclusion

Queueing-aware latency prediction offers a systems-level framework for understanding and managing delay in scalable large language model applications. By treating latency as an emergent property of queues, service distributions, scheduling policies, and infrastructure constraints, it moves beyond isolated benchmarking and average-case reporting. Classical queueing theory provides the conceptual foundation, while modern inference serving systems reveal the specific mechanisms that generate variation in large language model workloads. Continuous batching, paged memory management, distributed inference, structured program execution, model selection, and autoscaling all shape the distribution of response times. Prediction systems that ignore these mechanisms may be accurate in narrow conditions but fragile during bursts, model changes, or infrastructure degradation.

Operationally, queueing-aware prediction can improve admission control, routing, batching, autoscaling, and user communication. It can help operators meet service-level objectives while controlling cost and energy use. Yet these benefits depend on integration into stable control loops and on careful treatment of feedback effects. Predictions that change routing or admission can alter future workloads, creating dynamics that must be modeled and governed. Hybrid approaches that combine mechanistic queueing relationships with machine learning can offer both interpretability and adaptability. They can also support counterfactual reasoning, allowing decision makers to ask how latency would change under different capacity, priority, or deployment policies.

Governance, fairness, and sustainability are not peripheral to latency prediction. Latency determines who can use a service, under what conditions, and at what cost. It shapes access to education, healthcare, public services, and economic opportunity when language models mediate those domains. Policy frameworks should therefore treat latency as a service quality with distributional consequences. Transparency, contestability, disaggregated reporting, and sustainability accounting are necessary complements to technical optimization. Edge and federated architectures may broaden access, but they also introduce coordination and equity challenges. The next generation of queueing-aware latency prediction should therefore be evaluated not only by accuracy but by its contribution to resilient, fair, and environmentally responsible infrastructure.

References

1. Gross, D., Shortle, J. F., Thompson, J. M., & Harris, C. M. (2008). *Fundamentals of queueing theory* (4th ed.). Wiley.
2. Kendall, D. G. (1953). Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded Markov chain. *The Annals of Mathematical Statistics*, 24(3), 338–354.
3. Harchol-Balter, M. (2013). *Performance modeling and design of computer systems: Queueing theory in action*. Cambridge University Press.
4. Dean, J., & Barroso, L. A. (2013). The tail at scale. *Communications of the ACM*, 56(2), 74–80.
5. Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., & Stoica, I. (2023). Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles* (pp. 611–626). ACM.
6. Yu, G. I., Jeong, J. S., Kim, G. W., Kim, S., & Chun, B. G. (2022). Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation* (pp. 521–538). USENIX Association.
7. Pope, R., Douglas, S., Chowdhery, A., Devlin, J., Bradbury, J., Heek, J., Xiao, K., Agrawal, S., & Dean, J. (2023). Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5, 606–624.
8. Aminabadi, R. Y., Rajbhandari, S., Awan, A. A., Li, C., Li, D., Zheng, E., Ruwase, O., Smith, S., Zhang, M., Rasley, J., & He, Y. (2022). DeepSpeed-Inference: Enabling efficient inference of transformer models at unprecedented scale. In *SC22: International*

Conference for High Performance Computing, Networking, Storage and Analysis (pp. 1–15). IEEE.

9. Zheng, L., Yin, L., Xie, Z., Sun, C., Huang, J., Yu, C. H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J. E., Barrett, C., & Sheng, Y. (2023). SGLang: Efficient execution of structured language model programs. arXiv. <https://arxiv.org/abs/2312.07104>
10. Crankshaw, D., Wang, X., Zhou, G., Franklin, M. J., Gonzalez, J. E., & Stoica, I. (2017). Clipper: A low-latency online prediction serving system. In 14th USENIX Symposium on Networked Systems Design and Implementation (pp. 613–627). USENIX Association.
11. 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. In 14th USENIX Symposium on Operating Systems Design and Implementation (pp. 443–462). USENIX Association.
12. 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.
13. Romero, F., Li, Q., Yadwadkar, N. J., & Kozyrakis, C. (2021). INFaaS: Automated model-less inference serving. In 2021 USENIX Annual Technical Conference (pp. 397–411). USENIX Association.
14. Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Cowan, M., Shen, H., Wang, L., Hu, Y., Ceze, L., Guestrin, C., & Krishnamurthy, A. (2018). TVM: An automated end-to-end optimizing compiler for deep learning. In 13th USENIX Symposium on Operating Systems Design and Implementation (pp. 578–594). USENIX Association.
15. Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J. F., & Dennison, D. (2015). Hidden technical debt in machine learning systems. In Advances in Neural Information Processing Systems 28 (pp. 2503–2511). Curran Associates.
16. Strubell, E., Ganesh, A., & McCallum, A. (2019). Energy and policy considerations for deep learning in NLP. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (pp. 3645–3650). Association for Computational Linguistics.
17. Schwartz, R., Dodge, J., Smith, N. A., & Etzioni, O. (2020). Green AI. Communications of the ACM, 63(12), 54–63.
18. Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021). On the dangers of stochastic parrots: Can language models be too big? In Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency (pp. 610–623). ACM.
19. Mittelstadt, B. (2019). Principles alone cannot guarantee ethical AI. Nature Machine Intelligence, 1(11), 501–507.
20. Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., D’Oliveira, R. G. L., Eichner, H., El Rouayheb, S., Evans, D., Gardner, J., Garrett, Z., Gascón, A., Ghazi, B., Gibbons, P. B., ... Zhao, S. (2021). Advances and open problems in federated learning. Foundations and Trends in Machine Learning, 14(1–2), 1–210.