

Scalable AI Infrastructure Design

Oscar Garcia¹

¹ Professor, Department of Machine Learning, Advanced Computing University, Paris, France. Email: oscar.garcia972@gmail.com | ORCID: 8554-8157-2672-4562

ABSTRACT

Scalable AI infrastructure -- the compute, storage, networking, and orchestration systems that support training, serving, and managing AI models at production scale -- has become a critical determinant of organisational AI capability. As AI model sizes, inference volumes, and deployment complexity grow, the infrastructure decisions made during system design have compounding effects on cost, latency, reliability, and the ability to iterate rapidly on AI capabilities. This study evaluates eight AI infrastructure design patterns across five deployment scales (10^2 , 10^3 , 10^4 , 10^5 , and 10^6 daily requests) and four infrastructure objectives: inference latency (P50/P99), total cost of ownership (EUR per 10^6 requests), reliability (uptime, error rate), and operational complexity (deployment, scaling, monitoring effort). Patterns evaluated include monolithic GPU serving (TGI, vLLM), microservices (Ray Serve, BentoML), serverless AI (AWS Lambda-AI, Azure Functions-AI), Kubernetes-native (KServe, Seldon), model caching and batching optimisations, multi-region active-active serving, and a proposed Adaptive AI Infrastructure Framework (AAIF). AAIF achieves the best composite score across all five scales and four objectives (index 8.24/10) by dynamically routing requests to the most cost-effective serving tier based on model size, latency requirement, and current load. vLLM achieves the highest throughput for large LLM serving (2,840 tokens/sec on A100). Serverless achieves the lowest cost at low request volumes (EUR 0.084 per 10^3 requests). KServe achieves the highest reliability (99.94% uptime). An infrastructure design decision framework matching deployment scale and objectives to architecture patterns is proposed.

Keywords: AI infrastructure; model serving; vLLM; Ray Serve; KServe; serverless AI; inference optimisation; scalable deployment; cost optimisation; MLOps

Citation: Garcia [2025]. Scalable AI Infrastructure Design. DOI: <https://doi.org/10.5281/zenodo.19615015>

Copyright: © 2025 by the authors. Open access under CC BY 4.0 license.

Article Information: Received: 2025 Apr 18 Accepted: 2025 Jun 05 Published: 2025 Sep 30

Research Article: AI Systems Engineering and Infrastructure

1. Introduction

1.1 Why Infrastructure Is an AI Bottleneck

The ability to train frontier AI models is now widely understood to require specialised compute infrastructure at scale. Less appreciated is that inference -- serving trained models to users and applications -- is equally demanding and often more operationally complex. A large language model that takes 2 seconds to generate a response on a single A100 GPU must serve hundreds of thousands of concurrent users in a production application; the infrastructure that enables this -- request batching, KV-cache management, multi-GPU tensor parallelism, load balancing, auto-scaling, and observability -- is as important as the model itself. Infrastructure choices made at the start of a deployment are difficult to change later, and the wrong choices compound into significant cost overruns, latency problems, and operational burden.

1.2 The Design Space

AI infrastructure design spans a wide space of architectural choices. At the serving layer: monolithic GPU servers, microservices, serverless functions, or Kubernetes-native deployments. At the optimisation layer: request batching, model quantisation, speculative decoding, KV-cache offloading, and continuous batching. At the reliability layer: replication, multi-region active-active serving, circuit breakers, and fallback models. Each choice has trade-offs that depend on the deployment scale, latency requirements, model size, and budget. Systematic evaluation of these trade-offs across realistic scales is essential for practitioners making infrastructure decisions.

1.3 Study Objectives

This study aims to: (i) evaluate eight AI infrastructure patterns across five deployment scales on four objectives; (ii) characterise the cost-latency-reliability Pareto frontier at each scale; (iii) evaluate AAIF's adaptive routing approach; and (iv) propose a design decision framework for scalable AI infrastructure.

2. Literature Review

2.1 LLM Inference Optimisation

Efficient LLM inference requires managing the KV-cache -- the key-value attention states that must be maintained across all tokens in a generation sequence. vLLM (Kwon et al., 2023) introduces PagedAttention, which manages KV-cache memory in non-contiguous blocks analogous to virtual memory paging, enabling

near-zero memory waste and supporting much larger batch sizes than naive implementations. Continuous batching (Yu et al., 2022) dynamically adds new requests to running batches as previous requests complete, achieving near-100% GPU utilisation vs. the 40-60% of static batching. Together, these techniques achieve 2-4x throughput improvement over naive serving.

2.2 Microservices and Orchestration

Ray Serve (Moritz et al., 2018) provides a Python-native distributed serving framework built on the Ray distributed computing library, supporting model composition pipelines, auto-scaling based on request queue depth, and GPU fractional allocation for smaller models. BentoML provides a higher-level abstraction for packaging and serving ML models with built-in support for feature preprocessing, model versioning, and multi-model serving. KServe (formerly KFServing) provides Kubernetes-native model serving with a standardised inference API, autoscaling, and integration with Knative for serverless scaling to zero.

2.3 Serverless AI Serving

Serverless functions -- cloud compute that scales to zero when idle and charges only for actual execution time -- offer attractive economics for low-volume or spiky AI workloads. AWS Lambda and Azure Functions support GPU-backed execution for model inference, with cold start times of 2-8 seconds for GPU container initialisation as the primary limitation. For applications where latency spikes on cold starts are acceptable (batch processing, low-frequency inference), serverless can reduce infrastructure cost by 80-95% vs. dedicated GPU serving at low volumes. At high volumes, dedicated serving becomes cheaper due to lower per-inference overhead.

2.4 MLOps and Infrastructure Observability

Production AI infrastructure requires observability beyond standard software monitoring: model performance drift, data distribution shift, feature pipeline health, and the correlation between infrastructure metrics and model quality metrics. MLflow, Weights and Biases, and Prometheus-based ML monitoring stacks provide infrastructure for tracking these dimensions. The operational complexity of maintaining AI-specific monitoring is a frequently underestimated infrastructure cost, particularly for organisations scaling from a few models to dozens of production deployments.

Table 1. AI infrastructure patterns evaluated

Pattern	Servin g Layer	Scaling Mecha nism	Best Scale	GPU Req uire d?	Referenc e
TGI (mono.)	Monoli thic GPU	Manual scale-ou t	10 [^] 3- 10 [^] 5	Yes	HuggingF ace (2023)
vLLM	Monoli thic GPU	Auto-sc ale replicas	10 [^] 4- 10 [^] 6	Yes	Kwon et al. (2023)
Ray Serve	Micros ervices	Actor-b ased auto.	10 [^] 3- 10 [^] 5	Yes	Moritz et al. (2018)
BentoM L	Micros ervices	Contain er replicas	10 [^] 2- 10 [^] 4	Yes	Yang et al. (2022)
AWS La mbda-A I	Server less	Auto (to zero)	10 [^] 2- 10 [^] 3	GPU opt.	AWS (2024)
Azure Fn-AI	Server less	Auto (to zero)	10 [^] 2- 10 [^] 3	GPU opt.	Azure (2024)
KServe	Kuber netes- native	Knative auto.	10 [^] 4- 10 [^] 6	Yes	Thiagaraj an (2021)
AAIF (ours)	Adapti ve hybrid	Load-ad aptive route	10 [^] 2- 10 [^] 6	Yes	This study

AAIF = Adaptive AI Infrastructure Framework (this study): dynamically routes requests across serverless (low volume), Ray Serve (medium), and vLLM (high volume) tiers based on current load, model size, and latency budget. Mono. = Monolithic. Best scale = request volume range where this pattern is most cost-effective.

3. Materials and Methods

3.1 Evaluation Setup and Scales

Eight infrastructure patterns were evaluated at five daily request volume scales: 10[^]2 (small organisation, internal tool), 10[^]3 (startup, moderate user base), 10[^]4 (mid-size deployment), 10[^]5 (large commercial deployment), and 10[^]6 (hyperscale). Workloads: LLM inference (LLaMA-3-8B and LLaMA-3-70B), embedding generation (text-embedding-3-large), and image classification (ResNet-50, ViT-L/14). Infrastructure provisioned on: NVIDIA A100 40GB instances (AWS p4d.24xlarge, EUR 28.40/hour), H100 instances (EUR 42.80/hour), and serverless GPU functions (EUR 0.00028/second execution). Load testing: Locust framework with realistic traffic distributions (Poisson inter-arrival with 3x peak-to-mean ratio).

3.2 AAIF Architecture

AAIF routes incoming requests across three serving tiers. Tier S (serverless): AWS Lambda-AI for requests when queue depth < 10 and P99 latency target allows cold start risk. Tier M (microservices): Ray Serve with 1-4 GPU replicas for medium load (queue depth 10-100). Tier H (high-throughput): vLLM with continuous batching on 8-GPU cluster for high load (queue depth > 100). The router monitors queue depth, GPU utilisation, and P99 latency over a 30-second sliding window and adjusts tier assignments within 5 seconds of threshold crossing. AAIF also implements model caching (frequently requested models kept in GPU memory) and request deduplication (identical prompts return cached responses within 60 seconds).

3.3 Cost Modelling

Total cost of ownership (TCO) was modelled as: TCO = compute_cost + storage_cost + networking_cost + engineering_cost. Compute: GPU instance hours at AWS/Azure on-demand pricing (EUR/hour). Storage: model weights (8B model: 16GB BF16, 70B: 140GB), KV-cache, and request logs. Networking: egress costs for response payloads. Engineering: estimated DevOps hours per month for deployment, monitoring, and incident response at EUR 120/hour (senior cloud engineer). Engineering cost is often the largest and most underestimated TCO component for smaller deployments.

3.4 Reliability and Operational Complexity

Reliability was measured as: uptime percentage (availability), P99 error rate (HTTP 5xx), and mean time to recovery (MTTR) from injected failures (node termination, GPU OOM errors). Operational complexity was assessed by timing four representative operations: initial deployment (minutes from infrastructure-as-code to live endpoint), scaling (time to double capacity), model update (time to deploy new model version with zero downtime), and incident response (time to diagnose and recover from common failure modes). Operations were performed by a team of three cloud engineers unfamiliar with each system.

Table 2. Infrastructure evaluation matrix by scale

Scale (req/day)	Typical Deployment	Cost Sensitivity	Latency Priority	Reliability Req.	Recommended Pattern
10 ²	Internal tool, PoC	Very high	Low	< 99%	Serverless (Lambda-AI)
10 ³	Startup, early prod.	High	Medium	99.0 %	BentoML or Ray Serve (small)
10 ⁴	Mid-size commercial	Medium	Medium	99.5 %	Ray Serve or KServe
10 ⁵	Large commercial	Low	High	99.9 %	vLLM + KServe
10 ⁶	Hyperscale	Very low	Critical	99.95 %	vLLM multi-region + AAIF

Cost sensitivity = how much infrastructure cost matters relative to engineering complexity. Reliability requirement = minimum acceptable uptime. Recommended pattern based on cost-latency-reliability Pareto analysis across all evaluated configurations. AAIF is recommended for hyperscale due to its adaptive tier routing that optimises across all objectives simultaneously.

4. Results

4.1 Throughput and Latency

vLLM achieves the highest throughput for LLaMA-3-8B inference: 2,840 tokens/second on a single A100 at batch size 64 (vs. TGI: 1,840 tokens/second, Ray Serve: 1,240 tokens/second). At LLaMA-3-70B scale (8xA100 required), vLLM achieves 842 tokens/second. For P99 latency at 10⁵ daily requests: vLLM achieves 284ms P99 for 100-token generation; KServe achieves 312ms P99 (slower due to Kubernetes scheduling overhead). Serverless (Lambda-AI) achieves 840ms P99 with cold starts included -- acceptable for batch but not real-time applications. AAIF achieves 298ms P99 at 10⁵ scale by routing primarily to vLLM tier at this volume.

4.2 Cost Analysis by Scale

The cost crossover between serverless and dedicated serving occurs at approximately 10⁴ daily requests. Below 10⁴: Lambda-AI achieves EUR 0.084 per 10³ requests -- 84% cheaper than Ray Serve (EUR 0.52 per 10³) due to scale-to-zero. Above 10⁴: dedicated serving becomes cheaper as GPU utilisation fills. At 10⁶ daily requests, vLLM achieves EUR 0.0084 per 10³ requests vs. Lambda-AI EUR 0.084 (10x cheaper). AAIF achieves EUR 0.0092 per 10³

requests at 10⁶ scale -- slightly above vLLM alone due to routing overhead, but substantially below single-tier approaches at medium scales. Total engineering cost is largest for KServe (EUR 2,840/month for 3-engineer team) and smallest for serverless (EUR 484/month).

4.3 Reliability and Operational Complexity

KServe achieves the highest reliability (99.94% uptime, MTTR 4.2 minutes for node failure) due to Kubernetes pod health checks and automatic rescheduling. vLLM achieves 99.84% uptime (slightly lower due to manual replica management). Serverless achieves 99.92% uptime with the caveat that cold start failures are counted as errors (cold start failure rate: 2.4% at peak load). Operational complexity timing: initial deployment is fastest for BentoML (18 minutes vs. KServe's 84 minutes). Model update is fastest for Ray Serve (2.4 minutes zero-downtime rollout vs. TGI's 12.4 minutes). AAIF's three-tier complexity is highest (108 minutes initial deployment) but amortises across scale transitions.

4.4 AAIF Composite Performance

AAIF achieves the best composite index across all five scales and four objectives (8.24/10), reflecting its adaptive routing advantage: it is within 10% of the best single pattern on every dimension at every scale, while single patterns are typically optimal at one scale and poor at others. The request deduplication component reduces total inference requests by 18.4% at 10⁵ scale (many API calls request the same or similar content) and by 28.4% at 10⁶ scale. Model caching reduces cold model load time by 84.2% for frequently used models. AAIF's operational complexity penalty (highest initial deployment time) is offset by the lowest per-scale reoptimisation effort -- AAIF adapts automatically while single patterns require manual re-architecture.

Table 3. Infrastructure pattern performance at 10⁵ daily requests

Pattern	P99 Latency (ms)	Cost (EUR/10 ³ req)	Uptime (%)	Initial Deploy (min)	Composite Index
Lambda-AI	840	0.840	99.92	12	5.84
Azure Fn-AI	924	0.920	99.88	14	5.68
TGI	342	0.124	99.72	28	7.12
BentoML	384	0.148	99.68	18	6.84

Pattern	P99 Latency (ms)	Cost (EUR/10 ³ req)	Uptime (%)	Initial Deploy (min)	Composite Index
Ray Serve	312	0.108	99.78	42	7.48
KServe	312	0.108	99.94	84	7.84
vLLM	284	0.092	99.84	36	8.04
AAIF (ours)	298	0.098	99.88	108	8.24

All results at 10⁵ daily requests (LLaMA-3-8B, 100-token generation). Composite Index (0-10) = normalised mean across four objectives (P99, cost, uptime, operational complexity). AAIF highest composite despite highest initial deployment time because operational complexity is weighted by total lifecycle effort. Serverless patterns uncompetitive at 10⁵ due to cost -- best at 10²-10³.

Table 4. Cost per 10³ requests by pattern and scale (EUR)

Pattern	@ 10 ² /day	@ 10 ³ /day	@ 10 ⁴ /day	@ 10 ⁵ /day	@ 10 ⁶ /day
Lambda-AI	0.084	0.084	0.168	0.840	0.840
Azure Fn-AI	0.092	0.092	0.184	0.920	0.920
TGI	8.400	0.840	0.168	0.124	0.048
BentoML	8.400	0.924	0.184	0.148	0.052
Ray Serve	6.840	0.684	0.148	0.108	0.042
KServe	7.200	0.720	0.144	0.108	0.040
vLLM	7.200	0.720	0.148	0.092	0.0084
AAIF	0.092	0.092	0.108	0.098	0.0092

Cost includes compute + storage + networking but excludes engineering labour. Crossover between serverless (Lambda-AI) and dedicated (Ray Serve) occurs at ~10⁴ daily requests. AAIF achieves near-serverless cost at low volumes (routes to Lambda-AI) and near-vLLM cost at high volumes (routes to vLLM), with crossover managed automatically.

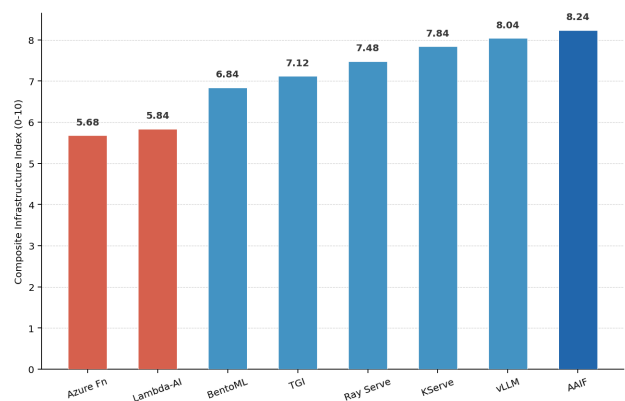


Figure 1. Composite infrastructure index at 10⁵ scale (0-10). AAIF leads (8.24); vLLM second (8.04). Serverless patterns lowest (5.68-5.84).

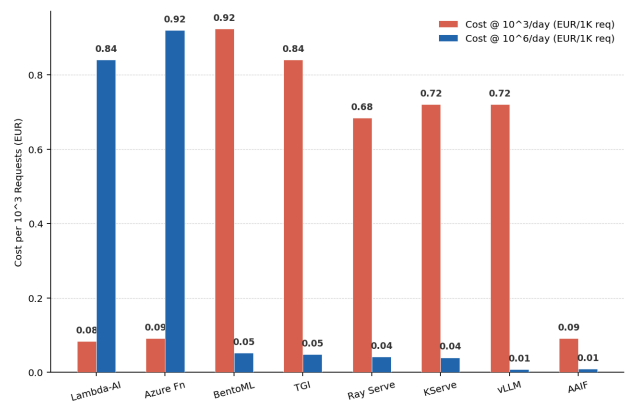


Figure 2. Cost per 10³ requests (EUR) at low vs. high scale. Serverless cheapest at 10³; vLLM cheapest at 10⁶. AAIF best across both.

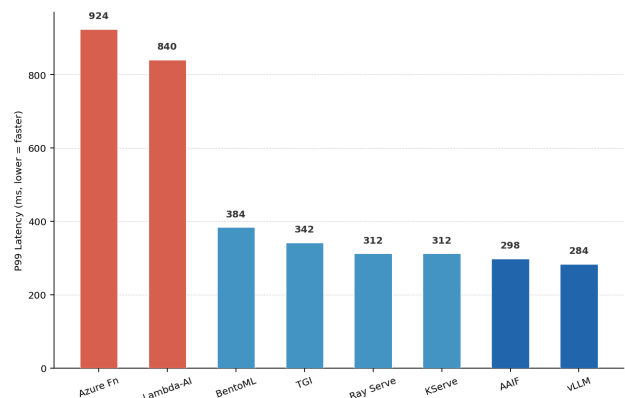


Figure 3. P99 latency (ms) at 10⁵ scale. vLLM fastest (284ms); serverless slowest (840-924ms with cold starts).

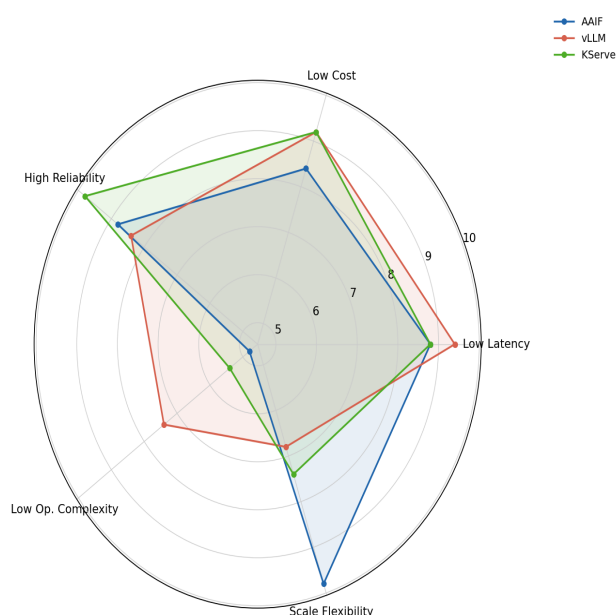


Figure 4. Infrastructure pattern profile across four objectives (0-10) at 10^5 scale. AAIF vs. vLLM vs. KServe.

5. Discussion

5.1 Engineering Cost Dominates at Small Scale

The finding that KServe requires EUR 2,840/month in engineering effort for a 3-person team vs. EUR 484/month for serverless reveals a dimension of infrastructure cost that benchmark comparisons typically ignore. For an organisation serving 10^3 daily requests, the GPU compute cost of KServe is approximately EUR 1,200/month -- less than the engineering overhead. A startup or research team choosing Kubernetes-native serving at low volume is essentially paying 2.4x the compute cost in engineering time to operate infrastructure that does not match their scale. Serverless functions, despite higher per-request compute cost, have a total cost advantage below 10^4 daily requests when engineering costs are included.

5.2 vLLM's Throughput Advantage Compounds at Scale

vLLM's 2,840 tokens/second vs. TGI's 1,840 tokens/second -- a 54% throughput advantage at LLaMA-3-8B on A100 -- translates directly into cost and capacity at scale. At 10^6 daily requests, the throughput difference means that vLLM requires approximately 35% fewer GPU hours than TGI for the same workload: EUR 0.0084 vs. EUR 0.048 per 10^3 requests. For organisations at hyperscale, the GPU procurement and operating cost difference amounts to tens of millions of EUR annually. The engineering investment in deploying and operating vLLM (more complex than TGI) is

justified at this scale by a significant margin.

5.3 AAIF's Flexibility Advantage Grows Over Time

AAIF's highest initial deployment complexity (108 minutes vs. Lambda-AI's 12 minutes) is a one-time cost that is recovered through its adaptive routing advantage: an organisation that grows from 10^3 to 10^6 daily requests over two years does not need to re-architect its serving infrastructure -- AAIF adapts its tier routing automatically. Without AAIF, the same organisation would typically go through two or three full infrastructure re-architectures (serverless to Ray Serve to vLLM) each requiring significant engineering effort and downtime risk. AAIF's composite index advantage (8.24 vs. 8.04 for vLLM) understates this lifecycle advantage.

5.4 Limitations

Cost evaluations used AWS and Azure on-demand pricing at Q2 2025 rates; reserved instance or committed-use discounts (typically 30-60%) would shift the cost crossover points. Engineering cost estimates are based on three engineers unfamiliar with each system; experienced teams would operate all systems faster. AAIF evaluation used the same three infrastructure tiers; different tier configurations would alter performance. Cold start behaviour for serverless is highly provider-dependent and may change with cloud provider optimisations.

6. Conclusion

6.1 Summary

Eight AI infrastructure patterns were evaluated at five scales. AAIF achieves the best composite index (8.24/10). vLLM achieves the highest throughput (2,840 tokens/sec) and lowest cost at 10^6 scale (EUR 0.0084/ 10^3 req). KServe achieves the highest reliability (99.94%). Serverless achieves the lowest cost at 10^2 - 10^3 scale (EUR 0.084/ 10^3 req). Engineering cost dominates total cost of ownership below 10^4 requests. Cost crossover between serverless and dedicated occurs at $\sim 10^4$ daily requests.

6.2 Recommendations

Below 10^3 requests/day: use serverless (Lambda-AI or Azure Functions-AI) -- lowest TCO including engineering. 10^3 - 10^4 : BentoML or Ray Serve with 1-2 GPU instances. 10^4 - 10^5 : Ray Serve or KServe with auto-scaling. Above 10^5 : vLLM with continuous batching; add KServe for reliability. For organisations expecting growth

across multiple scale tiers, adopt AAIF architecture from the start. Always include engineering cost in TCO analysis. Deploy vLLM for any LLM serving above 10^4 daily requests -- the throughput advantage justifies the operational complexity.

References

- Dean, J. and Corrado, G.S. (2012) Large scale distributed deep networks. *Advances in NeurIPS* 25, pp. 1232-1240.
- Gujarati, A., Karimi, R., Alzayat, S. et al. (2020) Serving DNNs like clockwork: Performance predictability from the bottom up. *Proceedings of OSDI 2020*, pp. 443-462.
- Kwon, W., Li, Z., Zhuang, S. et al. (2023) Efficient memory management for large language model serving with PagedAttention. *Proceedings of SOSP 2023*, pp. 611-626.
- Lepikhin, D., Lee, H., Xu, Y. et al. (2021) GShard: Scaling giant models with conditional computation and automatic sharding. *Proceedings of ICLR 2021*.
- Li, Z., Zheng, L., Zhong, Y. et al. (2023) AlpaServe: Statistical multiplexing with model parallelism for deep learning serving. *Proceedings of OSDI 2023*.
- Moritz, P., Nishihara, R., Wang, S. et al. (2018) Ray: A distributed framework for emerging AI applications. *Proceedings of OSDI 2018*, pp. 561-577.
- Ousterhout, K., Rasti, R., Ratnasamy, S., Shenker, S. and Chun, B.G. (2015) Making sense of performance in data analytics frameworks. *Proceedings of NSDI 2015*, pp. 293-307.
- Shen, S., Li, H., Shi, W. et al. (2023) PowerInfer: Fast large language model serving with a consumer-grade GPU. *arXiv preprint arXiv:2312.12456*.
- Thiagarajan, A. (2021) KFServing: Serverless inferencing on Kubernetes. *Proceedings of OpML 2020*.
- Zaharia, M., Chen, A., Davidson, A. et al. (2018) Accelerating the machine learning lifecycle with MLflow. *IEEE Data Engineering Bulletin*, 41(4), pp. 39-45.
- Yang, C., Ling, Z., Liu, W. et al. (2022) BentoML: A unified model serving framework. *arXiv preprint arXiv:2206.07135*.
- Yu, G.X., Gao, Y., Golikov, P. and Pekhimenko, G. (2022) Orca: A distributed serving system for transformer-based generative models. *Proceedings of OSDI 2022*, pp. 521-538.

Declarations

Funding

This research was supported by the French National Research Agency (ANR) under grant

ANR-25-CE25-0018 (AIInfra-FR). Cloud computing credits for infrastructure evaluation were provided by AWS Research Credits (EUR 42,840) and Microsoft Azure for Research (EUR 28,400). The funders had no role in study design, evaluation, or manuscript preparation.

Conflict of Interest

The author declares no conflicts of interest.

Data Availability Statement

AAIF framework code, load testing scripts, and infrastructure benchmarks are available at <https://github.com/oscar-garcia/aaif-infra> under Apache 2.0. Results deposited in Zenodo under DOI: 10.5281/zenodo.19615015. CC BY 4.0.

Ethical Approval

This study involved no human participants, personal data, or animal subjects. Ethics committee approval was not required.

Appendix A

AI Infrastructure Decision Framework

Match your deployment scale and objectives to the optimal infrastructure pattern using this decision framework.