

Distributed Cloud Performance Benchmarking on Google Cloud Platform Using MobileNetV2 Inference Workloads

Yogi Saputra¹, Muhammad Saifurridwani 'Ijazi², Wisnu Uriawan³

^{1,2,3}Informatics, UIN Sunan Gunung Djati Bandung, Indonesia
yogi.saputra@uinsgd.ac.id

Article Info

Article history:

Received September 08, 2026

Revised September 30, 2026

Accepted September 30, 2026

Published October 15, 2026

ABSTRACT

This study benchmarked MobileNetV2 inference on Google Kubernetes Engine to examine how cluster configuration and concurrent-user load affected service performance. A 3×3 factorial design compared static two-node, static four-node, and autoscaling two-to-four-node deployments at 10, 50, and 100 concurrent users. Five 15-minute runs per treatment yielded 45 experimental units. Client-side throughput, average latency, and error rate were interpreted alongside CPU and pod telemetry. At 100 users, the two-node configuration recorded 14.40 requests/s, 7,121.81 ms latency, and 70.67% errors. Four static nodes recorded 19.62 requests/s, 4,636.13 ms latency, and 50.53% errors. The 36.25% throughput increase was less than proportional to the doubled node count at that load. Autoscaling recorded 19.82 requests/s and 69.79% errors at 100 users, while its 50-user treatment coincided with pod thrashing. The reported two-way analysis of variance indicated significant configuration, load, and interaction effects at the 0.05 level; exact assumption-test outputs and sums of squares were unavailable for independent verification. The results showed that the tested configurations did not ensure reliable service under high concurrency. Resource limits, timeouts, and autoscaling settings remained plausible contributors requiring controlled follow-up experiments.

Keywords: Autoscaling; CPU Throttling; Distributed Cloud; Inference Benchmarking; Service Reliability.

INTRODUCTION

Cloud-based artificial intelligence inference requires measurements of throughput, latency, and service reliability under clearly specified workloads. MLPerf Inference provides a benchmark methodology for evaluating inference systems across deployment scenarios [1]. Google Kubernetes Engine (GKE) offers a managed Kubernetes environment with a control plane and worker nodes [2]. MobileNetV2 uses inverted residuals and linear bottlenecks to support efficient image processing [3], making it a useful workload for examining how a lightweight model behaves under concurrent requests.

Horizontal Pod Autoscaling (HPA) adjusts replica counts in response to observed metrics and supports configurable scaling behavior [4]. However, replica expansion alone does not establish proportional improvements in service performance. CPU limits can restrict execution through throttling, while memory limits can lead to out-of-memory termination [5]. Consequently, the observed response to user load depends on both provisioned resources and deployment

settings. Node-level utilization should be interpreted alongside container-level constraints and client-side failures.

Prior studies have addressed several components of this problem. K-Serve based infrastructure automation has been investigated for scalable machine learning inference [6], and production workloads and scheduling in heterogeneous GPU clusters [7], and transparent GPU sharing for deep-learning workloads serverless and virtual-machine container deployments have also been evaluated in high-performance computing environments [8]. FaST-GShare specifically addresses spatio-temporal GPU sharing for serverless inference [9]. These accelerator-oriented studies inform resource-management research but do not directly establish the performance of CPU-based MobileNetV2 serving on the configurations tested here. Research on automated evaluation of Kubernetes container network interfaces also supports measuring network behavior explicitly when investigating distributed-service bottlenecks [10]. hybrid task scheduling [11] and network-overhead mitigation [12] further demonstrate that observed performance is jointly determined by compute, orchestration, and communication layers.

Related work has also explored scalability limits in distributed deep-learning systems [13], predictive resource allocation [14], heuristic scheduling [15], edge-to-cloud orchestration [16], quality-of-service-aware Kubernetes inference [17], and shared GPU frameworks [18]. These studies largely emphasize custom schedulers, accelerators, training workloads, or cross-platform comparison. Broader analytics research confirms the value of measurement-driven evaluation and documents the diversity of machine-learning frameworks used at scale [19], image-data practices such as augmentation remain important for training robustness [19], but do not directly characterize production serving behavior. A controlled native-GKE baseline for CPU-based MobileNetV2 inference therefore remains useful.

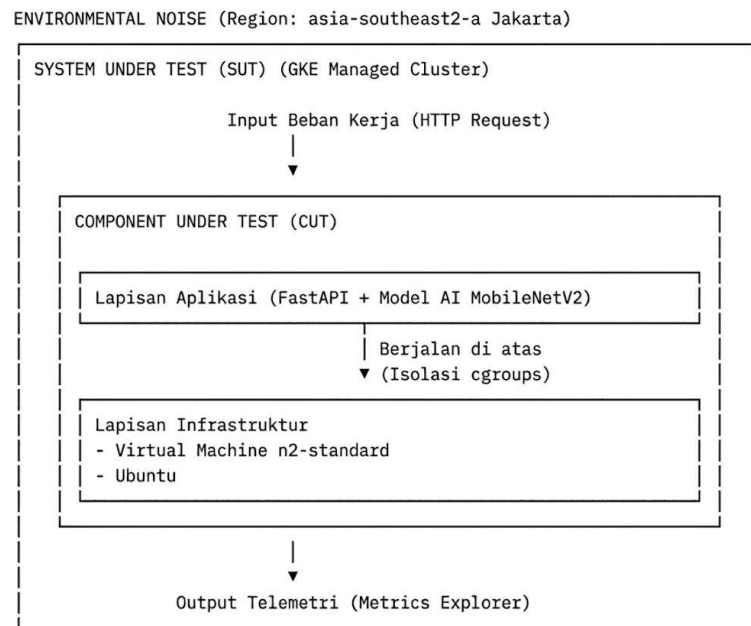
The present study therefore examines a controlled, configuration-specific comparison of static and autoscaling GKE deployments. It combines application-level measurements with resource telemetry to identify performance differences and plausible operational explanations. Its contribution is an empirical baseline for the tested CPU-based deployment, rather than a new autoscaling algorithm or a general capacity limit for Kubernetes. The research questions are how cluster configuration, user load, and their interaction are associated with throughput and latency, and what telemetry accompanies high request-failure rates. A 3×3 factorial benchmark compares three configurations at three concurrency levels. The analysis distinguishes observed outcomes from proposed mechanisms and limits its conclusions to the tested workload and configuration.

METHOD

This study employs an empirical benchmarking method using a 3×3 full factorial design. The scope of the System Under Test (SUT) is confined to a managed GKE cluster within the GCP data center in the `asia-southeast2-a` (Jakarta) region to eliminate environmental noise from the public network. The Component Under Test (CUT) consists of an application layer (FastAPI + MobileNetV2) packaged within a Pod container and running on an Ubuntu-based e2-standard-2 virtual machine (2 vCPUs, 8 GB RAM).

Experimental Design and System Boundary

An empirical benchmark with a 3×3 full-factorial design was conducted in the GCP asia-southeast2-a zone in Jakarta. The system under test was a managed GKE cluster. The component under test comprised a FastAPI inference service and a MobileNetV2 model packaged in a Kubernetes pod on Ubuntu-based e2-standard-2 worker nodes, each with 2 virtual CPUs and 8 GB of RAM. Locating the load generator and serving environment within the selected cloud region reduced external network variability but did not eliminate platform or control-plane noise. MobileNetV2 was selected for its lightweight architecture based on inverted residuals and linear bottlenecks [3], picture 1 shows the system under test.



Picture 1. Under Test System

The experiment included two factors: cluster configuration and user load. Throughout this manuscript, user load denotes the number of concurrent simulated users, not a financial expense. The configuration factor comprised Scenario A (static two-node cluster), Scenario B (static four-node cluster), and Scenario C (autoscaling between two and four nodes). The user-load factor comprised 10, 50, and 100 concurrent users.

Each of the nine treatments comprised five 15-minute Locust runs, giving 45 run-level experimental units. Pod resource limits were fixed at 1000m CPU and 2Gi memory. The recorded client-side metrics were throughput in requests per second (RPS), average latency in milliseconds, and error rate in percent. Infrastructure measurements comprised CPU utilization, CPU throttling, and pod lifecycle events from Advisor and GCP Monitoring. The available summary does not specify whether reported RPS includes failed requests; it is therefore described as reported throughput rather than successful-inference throughput. The statistical procedure used run-level throughput and log-transformed latency as dependent variables in a two-way analysis of variance (ANOVA), with cluster configuration, user load, and

their interaction as effects. Normality of model residuals was assessed using the Shapiro–Wilk statistic in equation (1):

$$W = \frac{(\sum a_i e_{(i)})^2}{\sum (e_{ij} - \bar{e})^2} \quad (1)$$

Homogeneity of variance was assessed using Levene’s statistic in equation (2):

$$F = \frac{MS_{between(deviasi)}}{MS_{withi (deviasi)}} \quad (2)$$

The source analysis described both assumption tests only as $p > 0.05$; their numerical statistics and exact p-values were not retained in the available report. This statement is insufficient to independently verify the assumptions. The ANOVA decomposition is shown in equation (3), where the terms represent total, configuration, user-load, interaction, and residual sums of squares:

$$SS_T = SS_A + SS_B + SS_{AB} + SS_E \quad (3)$$

RESULT AND DISCUSSION

Client-Side Performance and Cluster Infrastructure

The empirical test results from 45 experimental runs are summarized in Tables 1 and 2, presenting a comparison of application macro-metrics and cluster resource utilization.

Table 1. Summary of Client-Side Performance Metrics (Locust)

Scenario	User Load (users)	Throughput (RPS)	Avg Latency (ms)	Error Rate (%)
A (Static 2-Node)	10	6.60	1277.18	26.27
	50	10.13	4845.87	60.33
	100	14.40	7121.81	70.67
B (Static 4-Node)	10	20.75	183.54	0.00
	50	16.09	2785.73	23.44
	100	19.62	4636.13	50.53
C (Autoscaling 2-4 Node)	10	13.68	431.71	0.08
	50	11.07	4034.13	46.80
	100	19.82	4663.88	69.79

Table 2. Summary of Cluster Resource Utilization & Kernel Forensics

Scenario	User Load (users)	CPU Utilization (%)	CPU Throttling (%)	Pod Status / Observation
A (Static 2-Node)	10	24.83	35.06	Normal (2 Pods)
	50	20.09	24.80	Normal (2 Pods)
	100	20.67	25.43	Resource Starvation
B (Static 4-Node)	10	31.18	52.98	Normal (4 Pods)
	50	23.01	24.01	Normal (4 Pods)

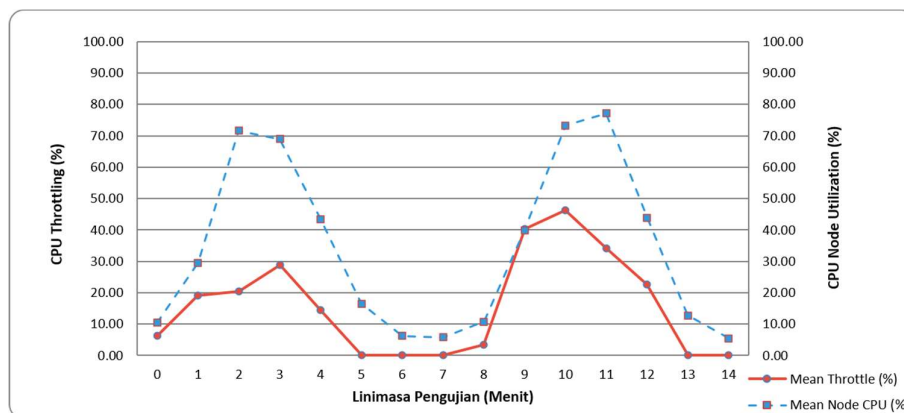
Scenario	User Load (users)	CPU Utilization (%)	CPU Throttling (%)	Pod Status / Observation
C (Auto scaling 2-4 Node)	100	18.89	22.35	Ingress Congestion
	10	72.41	61.87	Normal (2 Pods)
	50	40.51	23.64	Pod Thrashing
	100	37.27	22.74	OOMKilled

In Scenario A, reported throughput increased from 6.60 to 10.13 and 14.40 RPS as user load increased from 10 to 50 and 100 users. This increase was accompanied by average latency rising from 1,277.18 to 7,121.81 ms and error rate increasing from 26.27% to 70.67%. Thus, the deployment did not provide reliable service even at the lowest tested load, and the 100-user condition showed severe service degradation. The high error rate may reflect the combined effects of the 1000m CPU limit, 2Gi memory limit, request timeouts, application-worker settings, and queueing behavior. Table 2 reports 20.67% CPU utilization and 25.43% throttling at 100 users; these aggregate figures do not establish complete node-level CPU saturation. The evidence identifies a limitation of the tested configuration, not a fundamental inability of all two-node architectures to serve concurrent inference requests.

Scenario B recorded error rates of 0.00%, 23.44%, and 50.53% at 10, 50, and 100 users, respectively. At 100 users, it achieved 19.62 RPS with 4,636.13 ms average latency, improving on Scenario A while still failing approximately half of requests. The descriptive throughput gain was $[(19.62 - 14.40) / 14.40] \times 100 = 36.25\%$. Doubling the node count therefore produced a less-than-proportional increase in reported throughput at this load. This comparison does not measure cost efficiency or establish the cause of the scaling shortfall. The ingress-congestion label in Table 2 suggests a possible bottleneck, but attributing the difference to ingress routing or cross-node communication requires controlled ablation and network measurements.

Kernel-Level Forensics and Pod Dynamics: Scenario C

At 100 users, Scenario C achieved 19.82 RPS, the highest reported throughput among the three configurations at that load, with 4,663.88 ms latency and a 69.79% error rate. This was not the maximum throughput across all treatments: Scenario B recorded 20.75 RPS at 10 users. Within Scenario C, throughput fell from 13.68 RPS at 10 users to 11.07 RPS at 50 users, while the error rate rose from 0.08% to 46.80%. This non-monotonic response coincided with the pod-thrashing observation in Table 2. Picture 2 reproduces the infrastructure evidence included in the source manuscript.



Picture 2. Kernel and Container Monitoring Evidence

The reported HPA target was 70% average Utilization. Pod churn near a scaling threshold and delays between replica demand and node availability are plausible explanations for transient instability. However, the available summaries do not include synchronized HPA decisions, node-provisioning events, and request outcomes sufficient to establish this causal sequence. HPA CPU utilization is assessed relative to resource requests [4], so the target cannot be compared directly with a differently aggregated CPU-utilization percentage in Table 2. The resource-request settings and stabilization parameters are needed for a reproducible interpretation.

At 100 users, Scenario C was associated with an OOMKilled event and reported CPU throttling of 22.74% in Table 2. CPU throttling and memory-limit enforcement are distinct mechanisms [5]; the observed termination does not by itself demonstrate that CPU throttling caused memory accumulation. The tabulated error rate of 69.79% is used consistently here. Episode-level figures previously quoted for throttling, recovery time, telemetry gaps, and error rate cannot be reconciled with the treatment summaries without timestamped logs and defined aggregation windows. Accordingly, these figures are not treated as additional validated treatment estimates.

Inferential Analysis and Reporting Completeness

Table 3 retains the F-statistics and p-values from the source analysis, using decimal points and English labels. At $\alpha = 0.05$, all listed effects were reported as significant. The throughput interaction had $p = 0.0168$, so the results do not support the statement that all effects had $p < 0.001$. Because an interaction was reported, main effects should not be interpreted as uniform across user-load levels. The numerical results remain conditional on verification of the original run-level data and model diagnostics.

Table 3. Reported Two-Way ANOVA Results

Outcome	Source	df	SS	MS	F-statistic	p-value
Throughput	Configuration	2	NR	NR	22.941	< 0.001
Throughput	User load	2	NR	NR	10.741	< 0.001
Throughput	Interaction	4	NR	NR	3.479	0.0168

Outcome	Source	df	SS	MS	F-statistic	p-value
Throughput	Residual	36	NR	NR	—	—
Throughput	Total	44	NR	—	—	—
Log-latency	Configuration	2	NR	NR	43.894	< 0.001
Log-latency	User load	2	NR	NR	345.801	< 0.001
Log-latency	Interaction	4	NR	NR	12.867	< 0.001
Log-latency	Residual	36	NR	NR	—	—
Log-latency	Total	44	NR	—	—	—

Note: SS = sum of squares; MS = mean square; NR = not reported in the available analysis. Residual df = $3 \times 3 \times (5 - 1) = 36$ and total df = $45 - 1 = 44$ are implied by the stated complete design, assuming all runs were analyzed. F and p are retained from the source report. SS and MS cannot be uniquely recovered from F-statistics and degrees of freedom alone.

Table 4. Availability of Assumption-Test Results

Outcome	Diagnostic	Statistic	Exact p-value
Throughput	Shapiro-Wilk	NR	NR
Throughput	Levene	NR	NR
Log-latency	Shapiro-Wilk	NR	NR
Log-latency	Levene	NR	NR

The source report states only $p > 0.05$ for the prerequisite tests, without outcome-specific numerical results. NR does not indicate a passed test. Exact statistics, p-values, the Levene variant, and the log transformation must be recovered from the original analysis. The coefficient-of-variation stability criterion was also not numerically reported.

Practical Implications and Limitations

For comparable deployments, the findings motivate controlled tests of CPU requests and limits, memory headroom, application-worker concurrency, and request timeouts. Setting requests below limits can permit use above the requested allocation when capacity is available, but it does not remove the CPU limit [5]. HPA stabilization and conservative scale-down settings should be evaluated to reduce unnecessary replica changes [4], [6]. Request rate, queue depth, or latency may also be evaluated as complementary scaling signals. These are configuration hypotheses for subsequent tests, not interventions validated by the present experiment.

The experiment covered one GCP zone, one worker type, one model, one API implementation, and three user-load levels. Image size, batch size, ingress configuration, resource requests and limits, timeout values, autoscaler timing, and network placement were not independently varied. Cloud background load and control-plane timing could not be eliminated. The absence of run-level measurements, dispersion estimates, exact diagnostic outputs, and complete ANOVA components limits independent statistical verification. The workload definition, run order, warm-up policy, software versions, and latency transformation also require fuller reporting for replication. Cost and energy were not measured. Explanations involving ingress congestion, queue growth, or

autoscaling delays are observational and require targeted experiments before causal attribution.

CONCLUSION

This benchmark compared static two-node, static four-node, and two-to-four-node autoscaling configurations for MobileNetV2 inference on GKE. At 100 concurrent users, four static nodes increased reported throughput from 14.40 to 19.62 RPS, a descriptive gain of 36.25%, and reduced average latency from 7,121.81 to 4,636.13 ms; nevertheless, the error rate remained 50.53%. Autoscaling achieved 19.82 RPS at the same load but retained a 69.79% error rate. These outcomes indicate that adding nodes or enabling the tested CPU-driven autoscaling configuration did not ensure reliable service. They do not establish a fundamental limit of the architectures. The reported ANOVA effects were significant at $\alpha = 0.05$, subject to verification of missing diagnostic and model outputs. Future work should recover the complete run-level analysis, distinguish successful throughput from total request rate, and isolate the effects of resource limits, timeouts, ingress behavior, and autoscaler timing before extending the findings to other deployments.

BIBLIOGRAPHY

- [1] A. R. Atmadja, A. Rahmawati, C. N. Alam, P. Dauni, and Y. Saputra, "Sentiment Analysis on Tourism Place using Naive Bayes," *Proceeding of 2023 17th International Conference on Telecommunication Systems, Services, and Applications, TSSA 2023*, pp. 1–6, 2023, doi: 10.1109/TSSA59948.2023.10366891.
- [2] Y. Saputra, A. N. Herlita, and E. S. Nurpajriah, "Strategic Information System Planning at UMKM in Padalarang Village using Ward and Peppard Methods," *CoreID Journal*, vol. 1, no. 3, pp. 122–129, 2024, doi: 10.60005/coreid.v1i3.15.
- [3] H. Ahn *et al.*, "Performance Characterization of Using Quantization for DNN Inference on Edge Devices," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium Workshops*, IEEE, 2023. [Online]. Available: <https://arxiv.org/abs/2303.05016>
- [4] Y. Li *et al.*, "EfficientFormer: Vision Transformers at MobileNet Speed," in *Advances in Neural Information Processing Systems*, 2022, pp. 12934–12949. [Online]. Available: <https://arxiv.org/abs/2206.01191>
- [5] Y. Xu *et al.*, "CloudEval-YAML: A Practical Benchmark for Cloud Configuration Generation," in *Proceedings of Machine Learning and Systems*, 2024. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2024/file/554e056fe2b6d9fd27ffcd3367ae1267-Paper-Conference.pdf
- [6] V. J. Reddi and others, "MLPerf Mobile Inference Benchmark," in *Proceedings of Machine Learning and Systems*, 2022, pp. 352–369. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2022/file/a2b2702ea7e682c5ea2c20e8f71efb0c-Paper.pdf

-
- [7] Q. Zhang and others, "A Comprehensive Benchmark of Deep Learning Libraries on Mobile Devices," in *Proceedings of the ACM Web Conference 2022*, Association for Computing Machinery, 2022, pp. 3298–3307. doi: 10.1145/3485447.3512148.
 - [8] Q. Weng and others, "MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters," in *Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, USENIX Association, 2022, pp. 945–960. [Online]. Available: <https://www.usenix.org/conference/nsdi22/presentation/weng>
 - [9] S. Choi and others, "Serving Heterogeneous Machine Learning Models on Multi-GPU Servers with Spatio-Temporal Sharing," in *Proceedings of the 2022 USENIX Annual Technical Conference (USENIX ATC 22)*, USENIX Association, 2022, pp. 199–216. [Online]. Available: <https://www.usenix.org/conference/atc22/presentation/choi-seungbeom>
 - [10] M. Aach, A. Al-Saadi, and H. Fröning, "Large Scale Performance Analysis of Distributed Deep Learning Frameworks for Convolutional Neural Networks," *J. Big Data*, vol. 10, 2023, doi: 10.1186/s40537-023-00765-w.
 - [11] P. Rajendran, A. K. Sharma, and S. S. Chouhan, "Comparison of Cloud-Computing Providers for Deployment of Machine-Learning Applications," *Applied Sciences*, vol. 13, no. 23, p. 12577, 2023, doi: 10.3390/app132312577.
 - [12] G. Turin, M. Di Nitto, and D. A. Tamburri, "Predicting Resource Consumption of Kubernetes Container Systems Using Resource Models," *Journal of Systems and Software*, vol. 202, p. 111750, 2023, doi: 10.1016/j.jss.2023.111750.
 - [13] B. Wu, Z. Zhang, Z. Bai, X. Liu, and X. Jin, "Transparent GPU Sharing in Container Clouds for Deep Learning Workloads," in *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, USENIX Association, 2023, pp. 69–85. [Online]. Available: <https://www.usenix.org/conference/nsdi23/presentation/wu>
 - [14] J. Gu, Y. Zhu, P. Wang, M. Chadha, and M. Gerndt, "Fast-GShare: Enabling Efficient Spatio-Temporal GPU Sharing in Serverless Computing for Deep Learning Inference," in *Proceedings of the 52nd International Conference on Parallel Processing*, Association for Computing Machinery, 2023, pp. 635–644.
 - [15] H. Qiu and others, "AWARE: Automate Workload Autoscaling with Reinforcement Learning in Production Cloud Systems," in *Proceedings of the 2023 USENIX Annual Technical Conference (USENIX ATC 23)*, USENIX Association, 2023, pp. 387–402.
 - [16] D. R. Augustyn, M. W. Wójcik, and M. W. Szpyrka, "Tuning a Kubernetes Horizontal Pod Autoscaler for Meeting Performance and Load Demands," *Applied Sciences*, vol. 14, no. 2, p. 646, 2024, doi: 10.3390/app14020646.
 - [17] V. Dakić, M. Radovanović, M. Kovačević, and D. urić, "Methodology for Automating and Orchestrating Network Performance Evaluation in Kubernetes," *Computers*, vol. 13, no. 11, p. 283, 2024, doi: 10.3390/computers13110283.
-

- [18] H. Qiu and others, "Power-Aware Deep Learning Model Serving with μ -Serve," in *Proceedings of the 2024 USENIX Annual Technical Conference (USENIX ATC 24)*, USENIX Association, 2024, pp. 75–93. [Online]. Available: <https://www.usenix.org/conference/atc24/presentation/qiu>
- [19] Google Cloud, "Best Practices for Autoscaling Machine-Learning Inference Workloads on GKE," 2026. [Online]. Available: <https://docs.cloud.google.com/kubernetes-engine/docs/best-practices/machine-learning/inference/autoscaling-tpu>