



Triton Dynamic Batching Tuning: SLO and Rollback Runbook

Plan and build private AI compute with GPU Smith. We connect workload requirements with GPU hardware, networking, deployment and operating decisions so your infrastructure fits the work it must perform.

Published by GPU Smith · 2026-10-16 · triton dynamic batching tuning · triton dynamic batching · triton inference server · perf analyzer · model analyzer · latency slo · p99 latency · inference optimization · configuration rollback

Original article: <https://gpusmith.com/articles/en/triton-dynamic-batching-tuning-rollback-runbook>



In this article

1. Executive Summary
 2. Introduction and Background
 3. Key Changes: Make Batching a Controlled Variable
 4. Traffic and Measurement Design
 5. Implementation Considerations and Process Changes
 6. Bound the Search and Diagnose the Result
 7. Canary Rollout and Configuration Rollback
 8. Data Analysis and Evidence
 9. Implications and Future Directions
 10. Conclusion
-

Executive Summary

Triton dynamic batching tuning is an experiment in meeting a **latency service-level objective (SLO)** at higher useful throughput. Enable batching only for an eligible model, compare it with a verified baseline, and accept a candidate only when the client-observed latency distribution and error rate remain within the service's own limits. NVIDIA documents dynamic batching for stateless models, while Google's SLO guidance recommends measuring both typical experience and the latency tail. (docs.nvidia.com) (sre.google)

The recommended sequence is to inspect the loaded configuration, test batching without intentional delay, sweep **max_queue_delay_microseconds**, and investigate **preferred_batch_size** only when model-specific evidence justifies it. NVIDIA advises against preferred sizes for most models; a documented exception is TensorRT optimization profiles whose performance differs materially by batch size. A batch-formation delay is not an end-to-end latency guarantee: Perf Analyzer's server queue component also includes waiting for an available model instance. Use separate client and server observations to identify where time goes. (docs.nvidia.com) (opentelemetry.io) (sre.google)

Use fixed concurrency to explore capacity and request-rate tests to evaluate an independently arriving workload. A closed workload slows its own arrivals when responses slow; an open workload separates arrivals from completion. Consequently, a throughput gain under concurrency alone cannot establish that production traffic will meet its SLO. Record input shapes, request batch size, arrival distribution, warmup, observation windows, errors, and **p50, p95, and p99 latency**. The experiment matrix below is a recording template, not a published GPU benchmark. (grafana.com) (mlcommons.org)

As of **October 16, 2026**, NVIDIA's live release page identifies **26.09** as available (docs.nvidia.com); the corresponding official SDK distribution includes Performance Analyzer and Model Analyzer. (github.com) A planned **26.08** deployment should therefore be treated as a local upgrade milestone, with its own verification. This report recommends keeping that release change separate from batching changes. Stage the candidate alongside the baseline, judge equivalent traffic cohorts, and restore the complete known-good artifact set if acceptance fails. Kubernetes rollback restores the Pod template only, so model configuration stored elsewhere needs its own restoration mechanism. (kubernetes.io) No universal batch size, delay, throughput gain, or canary percentage is established here.

Introduction and Background

Dynamic batching combines compatible inference requests into work executed together. Triton exposes the scheduler through the model configuration, making batching a model-level serving choice. Eligibility matters: the documented dynamic batcher is intended for stateless models, rather than an interchangeable scheduler for every workload. This report addresses inference platform engineers operating owned GPUs who need to decide whether extra batching improves their particular service. (docs.nvidia.com)

The engineering decision begins with a measurement boundary. A server's execution measurement answers a different question from a user's elapsed request time. **OpenTelemetry** distinguishes remote procedure call (RPC) client and server duration metrics, providing a useful framework for correlating those observations without treating them as identical. The proposed acceptance boundary is the production client's logical operation, with backend timings retained for diagnosis. (opentelemetry.io) (sre.google)

Benchmark conditions also determine what a result means. **MLCommons** defines inference scenarios through both request patterns and the metric being measured. Its distinction supports an important discipline here: a capacity result belongs to the workload that produced it. A configuration that looks attractive under steady load still needs evaluation against the service's input mix and arrival pattern before it becomes a deployment candidate. (mlcommons.org)

From the perspective of an adjacent infrastructure engineering advisor such as **GPU Smith**, the useful deliverable is a reproducible decision record: which workload was tested, which constraint applied, which candidate passed, and how the previous configuration can be restored. The publisher is not a serving-runtime alternative in the comparison.

This runbook presents a tuning procedure, not measured benchmark results. **Warmup**, repeated runs, and an explicit inventory of artifacts therefore form part of the procedure. PyTorch's benchmarking guidance illustrates why: initialization can be lazy, and replicated measurements are needed to characterize noise. (docs.pytorch.org)

Keep **successful logical operations per second** as the local useful-throughput measure. Report transport attempts separately when retries are enabled, since retry behavior can add streams and backoff time. That avoids treating repeated attempts as additional completed user work. (grpc.io) (sre.google)

Key Changes: Make Batching a Controlled Variable

Verify the actual baseline

The baseline must describe the configuration Triton loaded. Merely deleting a handwritten batching block is insufficient: the model-configuration documentation states that backend auto-completion can enable dynamic batching. Inspect the generated configuration through the model configuration endpoint, and retain that response with the experiment record. (docs.nvidia.com)

Hypothetical Example: for a model named `image_encoder`, this command inspects its local endpoint:

```
curl --silent http://localhost:8000/v2/models/image_encoder/config
```

The documented endpoint returns the generated model configuration as JSON. Check its scheduling configuration, legal batch dimension, maximum batch size, input/output specification, and instance groups before labeling a run “no batching.” Preserve a fully specified known-good configuration for the baseline; disabling backend auto-completion is an available server control, but the loaded response remains the practical check. (docs.nvidia.com)

Keep the model artifact, runtime, protocol, request batch size, memory placement, and hardware allocation fixed during the first comparison. For example, **ONNX Runtime** documents that CPU-resident inputs can be copied to the device during execution, whereas I/O Binding can arrange transfers and allocation beforehand. Changing those conditions alongside batching would make attribution difficult. (onnxruntime.ai)

Separate the controls

Treat the experiment as a sequence of decisions:

- **Baseline:** prove that the loaded scheduler matches the intended control.
- **Immediate batching:** enable the batcher without adding intentional formation delay.
- **Delay sweep:** vary only the formation-delay setting within the local queue allocation.
- **Preferred sizes:** test these only after evidence identifies a useful size-specific execution profile.
- **Instance sweep:** vary execution instances after retaining a promising batching candidate.
- **Revalidation:** repeat the same traffic tests after any combined candidate is assembled.

These are proposed experiment stages, not NVIDIA performance thresholds. The vendor's documented default behavior forms available batches without intentional delay; when a configured delay expires, the scheduler sends the batch it has formed. (docs.nvidia.com)

A minimal fragment for the immediate-batching candidate is:

```
dynamic_batching { }
```

Retain the model's existing valid configuration around that fragment. Do not infer a legal maximum batch size from this example, and do not copy a preferred size from a different model.

Interpret delay as one contributor

Queue delay has two meanings that must remain distinct: intentional waiting to improve batch formation, and observed time waiting for service. Perf Analyzer's queue measurement includes waiting for a model instance. Accordingly, the acceptance test must observe total queue behavior rather than assume the configured formation delay bounds every wait. (docs.nvidia.com)

A useful operational check is to keep upstream failures visible too. For example, **Envoy** can reject requests under its overload controls or stop accepting new connections. A latency comparison that retains only completed requests could overlook those outcomes. Record admission, rejection, timeout, and completion counts at the service boundary. (www.envoyproxy.io) (opentelemetry.io)

Triton's model-control mode also determines how a configuration change takes effect. Its documentation distinguishes repository changes ignored in NONE mode from explicit model-control operations, and currently calls for unloading an existing model before loading its update. Document the real control path and its availability implications before relying on an in-place reload. (docs.nvidia.com)

Traffic and Measurement Design

Choose concurrency or request rate deliberately

Concurrency mode attempts to maintain a fixed number of outstanding requests. Use it to map capacity and compare configurations under controlled outstanding work. It answers how the server behaves when clients keep that amount of work in flight; it does not reproduce an independently specified arrival stream. (docs.nvidia.com)

For the SLO test, choose the arrival model that matches the service. Perf Analyzer supports request-rate and custom-interval modes; the latter consumes intervals expressed in microseconds. Its CLI also documents constant and Poisson request distributions. Retain the selected mode and distribution as experiment inputs rather than describing every run simply as “load testing.” (docs.nvidia.com) (github.com)

The distinction is consequential. Grafana's load-model documentation explains that a closed model waits for iteration completion before starting the next iteration, and can therefore lower its arrival rate as the system slows. An open-model test is needed when arrivals should remain independent of response completion. (grafana.com) (mlcommons.org)

Use representative inputs and timing

Production-like data must cover the model's relevant shapes and content classes. Perf Analyzer's default random data is generated once per input and reused, whereas JSON data can be supplied and iterated in round-robin order. Record the dataset identity and shape distribution explicitly. The CLI's `-b` controls the batch within a client request, distinct from the scheduler's combined batch; supply variable input shapes through `--shape` or JSON shape fields. (docs.nvidia.com)

Backend behavior provides further reasons to keep that distribution stable. **TensorFlow** documents shape-dependent function matching and expensive tracing. A different shape mix can therefore change the execution path being measured. Keep shape cohorts visible in the results rather than hiding them inside a blended average. (www.tensorflow.org) (mlcommons.org)

For backend microbenchmarks, GPU timing needs additional care. **PyTorch** documents asynchronous GPU operations and warns that measurements without synchronization are inaccurate. Keep such measurements separate from the ordinary service path: the purpose is to diagnose execution, not to introduce synchronization into every production request. (docs.pytorch.org)

Archive reproducible commands

Hypothetical Example: the following commands contain local experiment settings, not recommended production rates or sample sizes. They assume an existing compatible model named `image_encoder`, a model version named 1, representative `inputs.json`, and a local gRPC endpoint. All CLI options were checked against official Perf Analyzer documentation; no GPU benchmark was executed for this report. The CLI's `--percentile` option influences the latency used in stability measurement. (docs.nvidia.com)

```
perf_analyzer --help

perf_analyzer -m image_encoder -x 1 -i grpc -u localhost:8001 \
  --async -b 1 --input-data inputs.json \
  --concurrency-range 1:16:1 \
  --measurement-mode=count_windows --measurement-request-count=10000 \
  --warmup-request-count=1000 --percentile=99 \
  --verbose-csv -f baseline-concurrency.csv

perf_analyzer -m image_encoder -x 1 -i grpc -u localhost:8001 \
  --async -b 1 --input-data inputs.json \
  --request-rate-range 100:500:100 --request-distribution poisson \
  --measurement-mode=count_windows --measurement-request-count=10000 \
  --warmup-request-count=1000 --percentile=99 \
  --verbose-csv -f candidate-request-rate.csv
```

Here `-x` selects the model version; it is not a client binary version flag. Archive `--help`, the SDK image digest, and the package manifest to identify the actual toolchain. Customize the illustrated rates, counts, and concurrency range before running. (github.com)

Implementation Considerations and Process Changes

Freeze the experiment manifest

The manifest should identify enough state to reproduce the result without reconstructing a running machine. Record these items alongside every trial:

- **Model identity:** repository revision, model version, artifact checksum, backend, precision, and optimization profiles.
- **Runtime identity:** server and SDK image digests, backend packages, host driver, and deployment configuration.
- **Input contract:** dataset checksum, shape cohorts, request batch size, protocol, and preprocessing location.
- **Traffic:** arrival model, intended rate or concurrency, ramp profile, burst trace, and routing policy.
- **Measurement:** warmup rule, window rule, request count, repetitions, percentile method, and histogram boundaries.
- **Resources:** GPU allocation, memory conditions, host CPU allocation, client placement, and network path.

This manifest is the report's proposed audit record. Backend-specific controls belong in it because their effects extend beyond GPU batch execution. ONNX Runtime, for example, documents a thread-pool spinning tradeoff that consumes additional CPU cycles and power while potentially speeding inference. Keep that policy fixed during the batching sweep. (onnxruntime.ai)

Explicitly separate **warmup** from measurement. PyTorch's benchmark utility performs warmups because some components initialize lazily. Replicated measurements then help quantify noise. Adopt a local rule for reaching the intended warm state, retain the rule, and apply it to both baseline and candidate. A warm candidate should not be compared with a cold control. (docs.pytorch.org) (www.tensorflow.org)

Keep deadlines, retries, and errors in the record

Set and record the client deadline according to the service contract. **gRPC** does not set one by default, and its guidance recommends validating realistic values through load testing. A deadline should reflect the complete request path rather than just an execution-time observation. (grpc.io)

Retries require separate accounting. gRPC documents creation of another retry stream after backoff, so a logical operation may differ from a transport attempt. Compare user-visible completion time and successful logical operations, while retaining attempt counts to explain additional offered work. Record the retry policy for both control and candidate. (grpc.io) (opentelemetry.io)

Use bounded error categories so failed operations remain analyzable alongside successful latency observations. OpenTelemetry's RPC conventions recommend low-cardinality error types. The scorecard should distinguish successful completions, deadline expiry, admission rejection, and other failures rather than presenting successful latency alone. (opentelemetry.io) (sre.google)

Observe service and backend boundaries

Maintain separate client and server duration series. OpenTelemetry explicitly splits these measurements for correlation; a gap is a diagnostic signal to investigate, not proof of a particular bottleneck. Correlate request identifiers or traces where available before assigning that difference to network, preprocessing, or queueing. (opentelemetry.io)

Also verify that the load generator reaches the intended steady state. Locust distinguishes ramp-up rate from peak load, so a faster ramp does not by itself establish a larger sustained offered workload. Retain the achieved traffic trace through warmup, steady measurement, and bursts. (docs.locust.io)

Validate the generator as well as the server. Locust's guidance calls out generator CPU, network utilization, and operating-system metrics when increasing request rates. A saturated generator can prevent a test from delivering the traffic it was supposed to send. The run record should distinguish **offered**, **started**, **completed**, and **successful** operations. (docs.locust.io)

Bound the Search and Diagnose the Result

Model Analyzer can reduce manual trial management. Its configuration search supports custom model-configuration sweeps, including dynamic-batcher queue delay, and its configuration interface exposes search bounds and constraints. Use `model-analyzer profile --help` to inspect the command-line surface, then

set explicit search bounds and constraints. Use the tool to enumerate legal candidates inside the service's resource envelope, not to choose the business objective implicitly. (docs.nvidia.com)

A bounded search plan should make the following choices explicit:

- **Batch range:** include only sizes supported by the model and optimization profiles.
- **Delay range:** derive candidate waits from the queue allocation, beginning with no added wait.
- **Instance range:** constrain experiments to available GPU memory and deployment resources.
- **Traffic range:** cover the service's relevant low, normal, peak, and burst conditions.
- **Objective:** maximize useful completed work only after the latency and error constraints pass.
- **Checkpoint identity:** isolate measurements from experiments with different manifests.
- **Readiness:** require the generated configuration to load before measuring it.

Do not equate a syntactically generated configuration with a deployable configuration. Model Analyzer's documentation states that it cannot guarantee every generated configuration will load. Require an actual load and readiness check before counting a candidate as measured, and retain failed-load records rather than assigning them invented performance values. (docs.nvidia.com)

Use a distinct checkpoint directory for each materially different experiment. Model Analyzer loads its latest checkpoint at startup, making reused measurements helpful only when the underlying experiment identity remains valid. Changing the model, input distribution, GPU allocation, or toolchain should trigger a fresh experiment record. (docs.nvidia.com)

For toolchain selection, prefer a deliberately matched release distribution and then perform a local compatibility check. The official Model Analyzer installation instructions describe containers based on corresponding Triton server containers. That provides a reproducible starting point, but no retrieved source established a blanket guarantee for arbitrary cross-version combinations. (github.com)

Treat the recommended order as a way to preserve attribution. After delay tuning, investigate preferred sizes only when the model's execution profiles justify that experiment. Then test instance changes as a separate factor. If a combined candidate passes, retain the intermediate results so the team can explain which change produced the improvement.

For every load point, retain a row for each delay candidate, including failed trials. Filter first on client latency and errors. Among passing candidates, select the smallest delay that supplies the required capacity and reserve. If immediate batching already meets the requirement, require a measured benefit before adding wait. When testing preferred sizes, compare them at the same delay and instance setting, retaining only repeatable gains for the actual input cohorts. GPU utilization alone is not an acceptance condition.

Canary Rollout and Configuration Rollback

Apply a pass/fail scorecard

Table 1 defines the deployment decision. The service owner supplies its limits before testing. Google's event-based SLI formulation expresses good events relative to all eligible operations. (sre.google)

Acceptance dimension	Evidence	Pass condition	Rollback condition
Latency	Client p50, p95, p99 and timely-event fraction	Every declared latency objective passes in relevant cohorts	Any declared hard latency objective fails
Useful throughput	Successful logical completions at matched load	Required improvement or capacity target passes	No required gain, or completion capacity regresses
Errors and deadlines	All eligible operations, including rejected and timed-out requests	Within the predeclared error allowance	Allowance is exceeded
Queue stability	Time series of queue observations and completed work	Backlog clears under the declared traffic pattern	Backlog persists or grows beyond the local limit
Resources	GPU memory, GPU/host activity, network, generator health	Within the declared operating envelope	Resource limit or generator validity check fails
Correctness and recovery	Output checks, loaded config identity, rollback rehearsal	Outputs and complete restoration pass	Output acceptance or restoration fails

Every scorecard condition must pass: throughput does not cancel a latency failure. Evaluate shape, tenant, or request-class cohorts where their contracts differ. Keep the same percentile method for both deployments, and use an aggregatable representation when reporting a fleet-wide distribution. (prometheus.io)

Stage, compare, and restore

A practical rollout procedure is:

- **Snapshot:** preserve the known-good image digest, model artifacts, configuration, routing state, and experiment manifest.
- **Stage:** load the candidate separately and verify configuration identity and representative outputs.
- **Warm:** apply the same declared warmup policy before admitting measurement traffic.
- **Route:** send the predeclared canary share through a controlled traffic policy.
- **Compare:** evaluate matched cohorts against every scorecard condition.
- **Stop:** stop increasing exposure when a hard condition fails.
- **Restore:** return routing to the ready baseline and restore changed artifacts through the deployment's documented mechanism.
- **Confirm:** verify loaded configuration, output checks, traffic recovery, and SLO recovery before closing the change.

Kubernetes documents multiple Deployments as a canary approach. Readiness then determines whether a Pod receives traffic through Services, with readiness probes continuing throughout its lifecycle. Check the candidate and the restored baseline rather than treating process startup as sufficient evidence of serving readiness. (kubernetes.io) (opentelemetry.io)

Where **Envoy Gateway** manages weighted routing, proportions are calculated from each backend weight relative to the total. Multiple backends otherwise receive equal weighting by default. Retain the actual route and observed traffic split; merely adding a second backend should not be assumed to create the intended

canary. (gateway.envoyproxy.io) (docs.locust.io)

For a Kubernetes deployment, an illustrative rollback command is:

```
kubectl rollout undo deployment/triton-candidate --to-revision=2
```

Hypothetical Example: revision 2 represents a verified revision, not a universal recovery target. Kubernetes supports selecting a rollback revision, but only the Pod-template portion is restored. Separately managed model repositories, configuration, and routing must therefore be included in the recovery procedure. (kubernetes.io)

Track proxy and client outcomes during restoration. Envoy's overload action can return **HTTP 503** (www.envoyproxy.io), so a drop in successful throughput must be interpreted alongside rejection counts. gRPC cancellation also leaves the server application responsible for stopping work it started; a client deadline alone should not be treated as proof that all backend activity has ceased. (grpc.io)

Data Analysis and Evidence

Calculate a working queue allocation

Use an explicit planning equation:

Working queue allocation = end-to-end SLO budget – measured non-queue allocation – safety reserve.

The minus signs above express ordinary subtraction, not a percentile identity. The measurement boundary should follow the user-visible operation, consistent with Google's recommendation to move latency measurement closer to the user. The non-queue allocation should be informed by traces and measured path components, with a separately documented reserve. (sre.google)

Hypothetical Example: an internally chosen **50 ms** end-to-end budget, **35 ms** non-queue allocation, and **5 ms** reserve leave **10 ms** of working queue allocation: $50 - 35 - 5 = 10$. These are invented planning inputs, not Triton recommendations or observed results. A positive remainder permits an experiment; a nonpositive remainder calls for investigation before intentional waiting is added.

Do not subtract independently measured percentiles and treat the result as a guaranteed queue percentile. Nor should replica percentiles be averaged: Prometheus explicitly warns that averaging quantiles produces statistically meaningless values. Validate the resulting end-to-end distribution directly, with the chosen estimator and aggregation method recorded. (prometheus.io)

Preserve the experiment matrix

Hypothetical Example: *Table 2* is a reproducible trial template. **B** is the model's verified supported maximum batch setting, **D** is the local working queue allocation expressed in the scheduler's units, and **N** is the baseline instance configuration. Fractional-delay expressions are illustrative search points, not optimal settings. Every row carries the same manifest described above, including model/version, shapes, warmup, measurement window, and repetition identifier.

Trial	Scheduler / maximum	Arrival and load	Added delay	Preferred sizes	Instances	Results to record
Control	Verified default scheduler / B	Fixed concurrency, then representative arrivals	None	None	N	Throughput, errors, p50/p95/p99
Immediate	Dynamic / B	Identical trace or load point	Zero	Unset	N	Same fields and batch counts
Delay low	Dynamic / B	Identical trace or load point	D/4	Unset	N	Same fields and queue behavior
Delay middle	Dynamic / B	Identical trace or load point	D/2	Unset	N	Same fields and queue behavior
Delay upper	Dynamic / B	Identical trace or load point	D	Unset	N	Same fields and queue behavior
Profile candidate	Winning legal setting	Identical trace or load point	Retained candidate	Justified profile sizes	N	Same fields and execution behavior
Instance candidate	Retained legal setting	Identical trace or load point	Retained candidate	Retained decision	Varied separately	Same fields and resource usage

The matrix keeps changed controls visible. Convert units explicitly before producing a configuration, and treat D as an experimental boundary rather than a latency guarantee. Replay the same representative arrivals after each promising search result. MLCommons' documented scenario methodology likewise makes request pattern part of the result's definition. (mlcommons.org)

Separate averages, distributions, and useful work

For aggregate batch formation, compute **average executed batch size = inference-count delta / execution-count delta**, using matching intervals and labels. NVIDIA defines the underlying relationship between inference and execution counts. Handle resets and empty intervals explicitly. This reports an aggregate, not the distribution of batch sizes. (docs.nvidia.com)

For queue diagnosis, `nv_inference_queue_duration_us` records cumulative scheduler waiting time. Collect `nv_inference_request_duration_us`, `nv_inference_compute_infer_duration_us`, and `nv_inference_pending_request_count` alongside `nv_gpu_utilization`, `nv_gpu_memory_used_bytes`, and `nv_gpu_power_usage`. A matched interval calculation can estimate average queue time, but cumulative sums alone cannot reconstruct p99. Use client observations or a suitable histogram for the tail, and keep cache and request-accounting semantics consistent across the comparison. (docs.nvidia.com)

For cross-replica aggregation, Prometheus histograms support aggregation before quantile calculation; precomputed quantiles cannot simply be averaged. Histogram quantiles are estimates, so bucket resolution and sample size belong in the acceptance record, especially when a result lies close to the SLO boundary. (prometheus.io)

Implications and Future Directions

The tuning result should expire when its assumptions change. Refresh the experiment after model or backend changes, a different shape distribution, altered routing, different client placement, or a material resource-allocation change. The point is to preserve evidence for the deployed workload, rather than keep a configuration indefinitely because it once won a benchmark.

Keep a release upgrade on a separate change record. The official **26.09** release describes the corresponding SDK tool distribution. A team's planned **26.08** rollout is therefore a local release choice. Pin the chosen artifacts, verify their behavior, and rerun the batching matrix after that upgrade lands. (github.com)

There is likewise no basis here for turning a preferred batch size or canary fraction into a universal recommendation. Scenario-specific latency constraints are part of MLCommons' documented evaluation approach. The same principle is useful operationally: the service's contract determines acceptable performance, and the experiment establishes whether the configuration meets it. (mlcommons.org)

Three refresh checks deserve particular attention:

- **Distribution changes:** repeat relevant shape cohorts; TensorFlow's shape matching illustrates why inputs belong in the experiment identity. (www.tensorflow.org)
- **Measurement changes:** retain warmup and replicated measurements so changes in noise are visible. (docs.pytorch.org)
- **Generator changes:** recheck CPU and network headroom before comparing a new harness or traffic scale. (docs.locust.io)

For longer-term observability, keep client and server duration boundaries explicit. OpenTelemetry's split RPC metrics provide a useful naming model, while the deployed instrumentation and its semantic-convention version should remain part of the manifest. Preserve enough raw evidence to recompute the acceptance decision after a dashboard or aggregation method changes. (opentelemetry.io) (prometheus.io)

For a custom client harness, use an elapsed-time clock. Python documents that its monotonic clock is unaffected by system clock changes, and that its performance counter includes elapsed sleep time. That is appropriate for observing waits across a request, unlike measuring only the harness's CPU consumption. (docs.python.org)

Do not describe GPU power telemetry as full-system energy. MLCommons' benchmark power boundary measures AC power at the wall. If the team evaluates energy alongside latency and throughput, identify whether the observation covers the GPU, host, or complete system. (mlcommons.org)

Conclusion

Triton dynamic batching should be enabled through an explicit, reversible performance decision. Begin with the configuration that actually loaded, retain a valid no-batching control, and test immediate batching before adding formation delay. The report's experiment matrix then changes delay, preferred sizes, and instances in a sequence that makes attribution possible.

The latency contract must remain attached to the user-visible operation. A good server execution average cannot substitute for an end-to-end distribution, and a closed load test cannot establish behavior under independently arriving traffic. Preserve the offered workload and achieved completions together; the distinction between open and closed models explains why both are necessary. (grafana.com) (mlcommons.org)

Acceptance should require the complete scorecard to pass: latency, useful throughput, errors, queue behavior, resources, output correctness, and recovery. Keep quantile aggregation valid, because averaging replica quantiles does not produce a meaningful fleet percentile. Treat results near the boundary with the precision of the estimator in view. (prometheus.io)

Finally, retain a ready baseline and restore the complete artifact set when a candidate fails. The rollback boundary must be explicit: Kubernetes restores a Deployment's Pod template, while separately managed serving assets require separate recovery. A tuning result is operationally useful when another engineer can reproduce its workload, understand its acceptance decision, and restore its predecessor. (kubernetes.io)

Source links

Links cited in this article, in order of first appearance. Full addresses are provided for readers using extracted text.

1. https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/user_guide/batcher.html
2. <https://sre.google/workbook/implementing-slos>
3. <https://opentelemetry.io/docs/specs/semconv/rpc/rpc-metrics>
4. <https://grafana.com/docs/k6/latest/using-k6/scenarios/concepts/open-vs-closed>
5. <https://mlcommons.org/benchmarks/inference-datacenter>
6. https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/introduction/release_notes.html
7. <https://github.com/triton-inference-server/server/releases/tag/v2.73.0>
8. <https://kubernetes.io/docs/concepts/workloads/controllers/deployment>
9. https://docs.pytorch.org/docs/2.14/benchmark_utils.html
10. <https://grpc.io/docs/guides/retry>
11. https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/user_guide/model_configuration.html
12. <https://onnxruntime.ai/docs/performance/tune-performance/iobinding.html>
13. https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/perf_analyzer/docs/measurements_metrics.html
14. https://www.envoyproxy.io/docs/envoy/latest/configuration/operations/overload_manager/overload_manager
15. https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/user_guide/model_management.html
16. https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/perf_analyzer/docs/inference_load_modes.html
17. https://github.com/triton-inference-server/perf_analyzer/blob/main/docs/cli.md
18. https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/perf_analyzer/docs/input_data.html
19. <https://www.tensorflow.org/guide/function>
20. <https://docs.pytorch.org/docs/main/notes/cuda.html>
21. https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/perf_analyzer/docs/cli.html
22. <https://onnxruntime.ai/docs/performance/tune-performance/threading.html>

23. <https://grpc.io/docs/guides/deadlines>
24. <https://docs.locust.io/en/stable/increasing-request-rate.html>
25. https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/model_analyzer/docs/config_search.html
26. https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/model_analyzer/docs/checkpoints.html
27. https://github.com/triton-inference-server/model_analyzer/blob/main/docs/install.md
28. <https://prometheus.io/docs/practices/histograms>
29. <https://gateway.envoyproxy.io/latest/tasks/traffic/http-traffic-splitting>
30. https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/user_guide/metrics.html
31. <https://mlcommons.org/2026/03/mlperf-inference-gpt-oss>
32. <https://docs.python.org/3/library/time.html>

THE PUBLISHER

About GPU Smith

GPU Smith provides independent engineering and research for private AI infrastructure. We help technical buyers and operators reason about GPU workload sizing, hardware selection, procurement, deployment and inference operations, from the compute system to the networking, power and cooling requirements around it.

Start with the workload

Useful infrastructure decisions begin with the models, data, throughput, latency and operating constraints a team actually has. GPU Smith helps connect those requirements with system design and deployment choices. The goal is a privately controlled AI environment whose capacity and operational demands are understood before a purchasing decision.

Hardware and supplier research

Our public reference library covers GPUs, complete systems and networking components. The server-vendor directory and manufacturing research help teams investigate suppliers, compare options and follow sources behind technical claims. We distinguish manufacturer specifications from measured benchmarks and advertised capabilities from tested configurations. Reference pages are research resources, not a live inventory listing or a binding equipment quote.

Deployment and operations

GPU Smith's engineering scope includes infrastructure integration, networking, power, cooling and the practical operation of inference workloads. Our published articles explain the tradeoffs behind deployment, procurement and ongoing operations so teams can ask better questions and document decisions.

Work with GPU Smith

Explore the [hardware library](#), [GPU references](#), [networking references](#) and [vendor research](#). [Contact GPU Smith](#) with your workloads and deployment constraints to discuss a project and confirm current scope, pricing and availability.

Inclusion of a manufacturer or product in our research does not imply a partnership, certification or endorsement.

Website: <https://gpusmith.com>

This article is educational. Verify consequential medical, legal, financial, regulatory and technical decisions with appropriate professionals and the cited primary sources. Company services are described separately from the article's research findings.