PERFORMANCE BENCHMARK

The Cost of Governance

What it costs to run virtual-key auth, quotas, verified zero-drop audit, and inline content scanning on an LLM gateway's hot path — measured across seven gateways on one shared upstream, plus a durability endurance test. Among gateways that durably audit under load, Nexus leads; this report quantifies the cost.

Version	Date	Method	Gateways
v1.0	July 2026	2 rounds + 550 tiebreak · 6 tiers · shared mock upstream	Nexus · agentgateway · Bifrost · Kong · TensorZero · Portkey · LiteLLM

Contents

1 Executive summary

This report answers one question: what does it cost to run a gateway that *governs* traffic — virtual-key auth, quotas, verified zero-drop audit, and inline content scanning on the hot path — rather than one that simply forwards bytes?

Most gateway benchmarks pit one bare reverse proxy against another, which is not the decision operators face. We measured the real tradeoff on identical hardware, against one shared upstream, with the full rig published (github.com/AlphaBitCore/llm-gateway-benchmark) so anyone can reproduce or contest it. The one gateway faster than Nexus (agentgateway, ~2×) ran as a bare proxy: its request-log store was disabled for the throughput comparison because enabling it exhausts memory at benchmark-scale request rates. In the separate 5,000 req/s endurance test its logging retained nearly all records but was not zero-loss (§5). Nexus is not the fastest bare proxy; it is the only gateway in this test to combine high governed throughput with zero-record-loss audit durability through a database outage.

0
AUDIT RECORDS NEXUS LOST — 30-MIN SOAK THROUGH A DATABASE OUTAGE

83%
RECORDS BIFROST'S LOGGING LOST UNDER THE SAME TEST (99% ON STREAMING)

39,754
SUSTAINED REQ/S, 550-TOKEN CHAT, FULL AUDIT ON (MEDIAN OF FIVE ROUNDS)

30–51%
MEASURED COST OF INLINE CONTENT SCANNING

What we found

- **The only gateway faster than Nexus ran as a bare proxy.** agentgateway led raw throughput (~2× Nexus) — with its request-log store disabled, because enabling it exhausts memory at benchmark-scale rates (§5, Appendix A). Among gateways that durably audit under load, Nexus leads: **39,754 req/s** at the chat tier, 72% ahead of Bifrost.[1]
- **Under a 30-minute soak and a mid-run database outage, Nexus lost zero records.** In an endurance test at 5,000 req/s with the backend PostgreSQL cut for 30 s, Nexus persisted all 9,000,000 audit records; agentgateway's logging, enabled at this lower rate, was near-lossless but not zero-loss (71 records lost); **Bifrost lost 83%** (99% on streaming); and LiteLLM exhausted memory at 1,000 req/s (§5).
- **Inline content scanning has an honest, measurable cost.** Turning on the five content hooks (PII + policy, on request and response) reduced throughput by roughly **30–51%** on non-streaming workloads — a real tradeoff you opt into per route, quantified here rather than hidden.
- **Full-body audit capture is nearly free.** Storing complete request and response bodies (the heaviest durable-audit mode) stayed within run-to-run variance of metadata-only, because the capture happens off the request path.
- **Throughput isn't the whole story — behavior under load separates the field.** As concurrency rose, Nexus and Kong kept end-to-end latency near one second, while LiteLLM's streaming latency climbed past **19 seconds** (throughput flat at ~90 req/s) and TensorZero's 12.5k tier collapsed reproducibly (one round to 0 req/s, p99 tail to ~91 seconds). A peak-only number hides this (§4.6).

1. The 550 non-streaming tier showed ~13–15% round-to-round variance at its saturated stage — real saturation noise, not a fault — so it was re-run for three additional fresh rounds on the same ladder (ok 100% in every round). The published value is the median of the five-round series: scanning-off 35,259 / 36,363 / **39,754** / 41,050 / 41,328; scanning-on median 22,819. Every raw round is published in the repo — nothing is cherry-picked. Other cells converged within ~6% and are the best of two rounds.

2 What we tested

Seven LLM gateways, each isolated on its own dedicated machine, all forwarding to one shared OpenAI-compatible mock upstream. Isolating each gateway removes cross-contention; the shared upstream makes results directly comparable.

> **What this benchmark exercises of Nexus.** These results measure Nexus's **AI Gateway** data plane — the network-proxy path. That is one of three interception layers Nexus offers (an in-process SDK, this network proxy, and an endpoint agent), all sharing a single policy and compliance engine. The benchmark characterizes the proxy hot path, not the full product; the other planes are out of scope here. Nexus is included as one of the seven gateways (shown in two scanning configurations), but it is not a member of the LLM-proxy category so much as one deployment mode of a broader governance platform.

Feature parity — what each gateway actually does per request

Every gateway forwards to the same mock; what differs is the real work each performs on the hot path. This is the enabled-feature matrix for this rig, not the products' full capability lists.

Gateway	Audit / logging	Content scanning	Rate limit / quota	Hot-path datastore
Nexus (Go)	**Verified zero-drop full-body audit, async & off-path (NATS→hub→PG + disk spill) — dropped=0 in all fully reconciled runs**	**5 hooks (~423 rules), swept off/on; req+resp; redacts**	**Quota + cred-stats (async write-behind)**	**PG · Redis · NATS**
Kong (Lua)	File-log (best-effort/buffered — not zero-drop)	None (enterprise-only in free build)	Rate-limiting (counts; set so high it never throttles)	PG
Bifrost (Go)	Logging on (PostgreSQL)	None	None	PG
TensorZero (Rust)	Observability on (full inference rows → ClickHouse)	None	None	ClickHouse
Portkey (Node)	None (no DB, cache off)	None	None	None
LiteLLM (Python)	Message logging off (pure proxy)	None	None	None
agentgateway (Rust)	Request-log store **disabled** in throughput runs (bare proxy — enabling it exhausts memory at benchmark-scale rates); enabled in the §5 endurance test	None	None	None (throughput runs)

Table 1. Enabled features per gateway. Nexus is the only gateway with a free content-scanning axis, so it is shown in two configurations (scanning off / on).

> Use Nexus's **scanning-off** row for cross-gateway throughput comparison. It is still not a bare proxy — it runs routing, virtual-key auth, quota, and async zero-drop audit. The scanning-on row is the added cost of inline inspection.

> **Durable audit under load, as used in this report.** Successful requests remain fully accounted for in durable storage or a recoverable durable buffer while the gateway sustains the stated workload, including the database-outage test defined in §5. Four levels appear in this report:

Category	What it means
Bare forwarding	No audit log in the measured configuration
Best-effort logging	Records may be dropped, delayed, or lost under pressure
Durable audit	Records remain recoverable under stated load and failure conditions
Zero-loss durable audit	No successful-request audit record is lost in the stated test

Audit categories used throughout this report.

On agentgateway. Now in the matrix (Linux Foundation project, Rust, v1.3.1, standalone-binary `llm` mode). In the throughput matrix it ran as a **bare proxy**: its request-log store was disabled because enabling it exhausts memory at benchmark-scale request rates (unbounded writer queue, no backpressure or spill; reported upstream — see Appendix A). Its matrix throughput is therefore a pure-forwarding ceiling with nothing persisted — read it alongside LiteLLM/Portkey (bare), not the audit gateways. At the lower 5,000 req/s endurance rate its log store does operate; its durability there is measured directly in §5.

Workload tiers

Tier	Input tokens	Output tokens	What it stresses
Routing-core	128	128	Pure gateway overhead (NVIDIA / vLLM shape)
Realistic chat **(headline)**	550	150	The de-facto “average chat” (LLMPerf / Anyscale)
Long-context / RAG	~12,500	150	Large-body handling & bandwidth

Table 2. Three prompt sizes, each run non-streaming (throughput) and streaming. Sizes follow industry-standard benchmark shapes.

3 Methodology & disclosure

Disclosure. This benchmark rig, the shared mock upstream, the load generator, and this report are built and operated by the Nexus team at AlphaBitCore (ABC), and **Nexus is one of the gateways under test**. To make that auditable rather than a matter of trust, the entire rig is published at **github.com/AlphaBitCore/llm-gateway-benchmark** — infrastructure templates, the automation that installs and tunes *every* gateway, the load profiles, and the raw per-stage results for both rounds. Anyone can reproduce or contest these numbers end to end. Nexus was benchmarked on the same rig and instance type as the other gateways.

Standing offer. We will re-run any vendor's suggested configuration of their own gateway on this published rig and publish the result unedited — reproducibility works in both directions. This applies in particular to any cell in this report a vendor believes a configuration change would improve (e.g. Kong's upstream connection pool, Appendix A).

The shared-upstream design

Every gateway forwards to one shared mock that returns a deterministic OpenAI-format echo, instantly. Before any measurement, each gateway must prove it actually reaches that mock — guarding against a misconfigured gateway short-circuiting or calling a real provider. Because the upstream is identical and instant for everyone, gateway differences show up cleanly in sustained throughput at matched concurrency.

How to read the numbers

- **Throughput is primary.** Peak sustained requests/sec at matched load is the headline metric and the most robust across rounds.
- **Streaming latency is a relay artifact, not user latency.** The mock emits all tokens back-to-back with no inter-token delay, so streaming time-to-first-token reflects only how fast a gateway relays the first chunk — not what a user would feel behind a real model. Against a slow model these gaps compress sharply. The streaming *ranking* (relay efficiency) is meaningful; the absolute millisecond values are not user-facing latency.[2]
- **Only one row is a true bare proxy.** agentgateway's throughput rows ran with its request-log store disabled; every other gateway does its own real work (Nexus: auth + quota + async audit; Bifrost: PG logging; TensorZero: ClickHouse observability; Kong: file-log + rate-limiting; LiteLLM/Portkey: closest to bare). Cross-gateway numbers reflect these architectural choices, not an identical workload.
- **"vs. LiteLLM"** multiples use the slowest gateway as the 1.0× baseline, so they flatter the leaders; absolute RPS is always shown alongside.

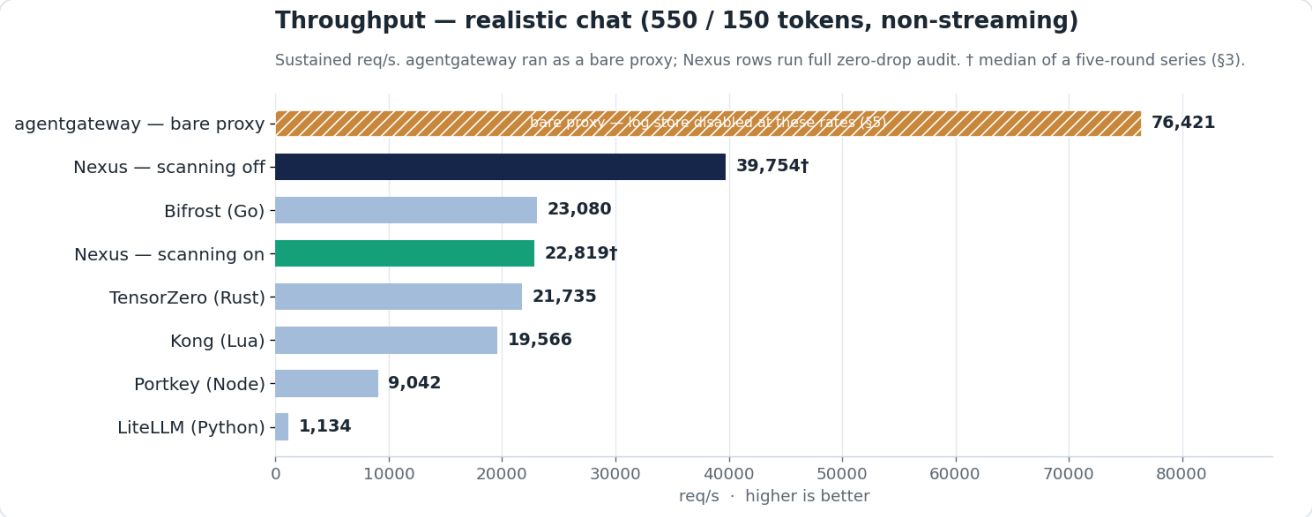
Fairness, by construction

Identical instance type, OS, kernel and socket tuning, and storage throughput on every box; host-native installs (no container overhead); the same mock and the same load ramps for all gateways at each tier. Each gateway was run two rounds; tiers with >10% disagreement or <99.5% ok were re-run. One tier needed further handling: the Nexus 550-token non-streaming result stayed ~13–15% variable at the top-concurrency stage — genuine saturation noise — so it was re-run for three additional fresh rounds and is reported as the **median of the five-round series**, with every raw round published (§4.1). Known asymmetries are disclosed, not hidden — see Appendix A.[3]

2. For the same reason, inter-token latency is ~0 for every gateway and streaming RPS tracks non-streaming RPS. We report streaming results for relay efficiency, clearly separated from non-streaming throughput.

4 Results

4.1 Headline — realistic-chat throughput (550 / 150, non-streaming)



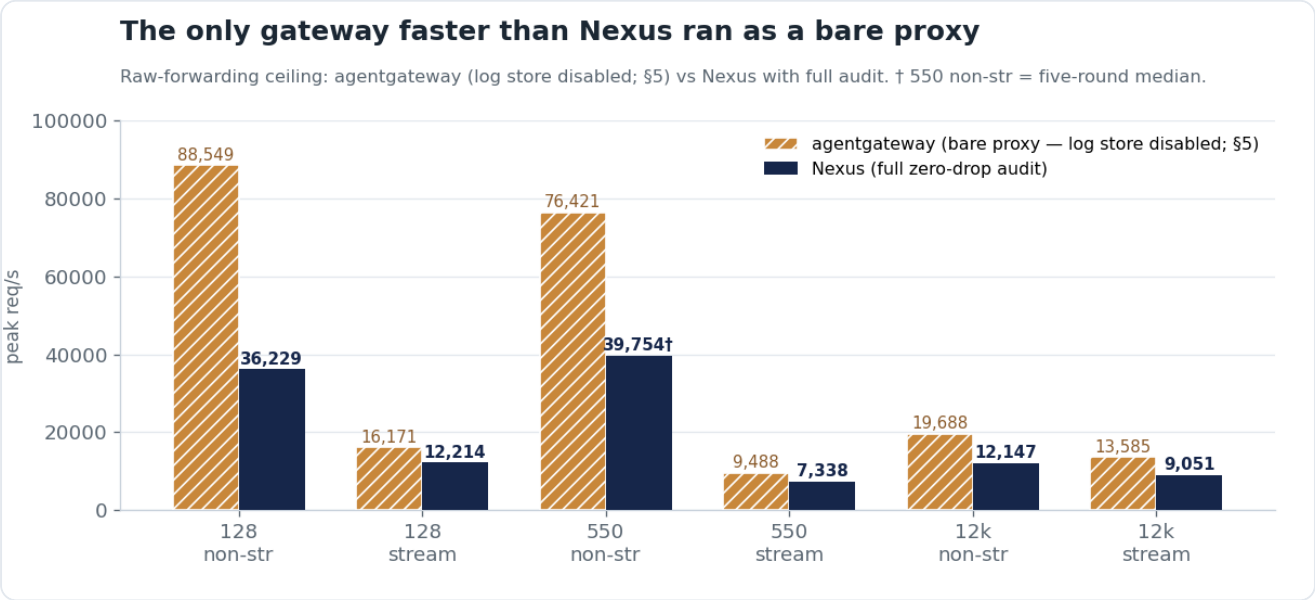
Gateway	Reported sustained RPS	vs. LiteLLM	ok %	p99 latency (ms)
agentgateway ‡ (bare proxy)	76,421	67.4×	100.0	42
Nexus — scanning off	**39,754**†	**35.1×**	**100.0**	**130**
Bifrost (Go)	23,080	20.4×	100.0	167
Nexus — scanning on	22,819†	20.1×	100.0	233
TensorZero (Rust)	21,735	19.2×	100.0	127
Kong (Lua)	19,566	17.3×	100.0	168
Portkey (Node)	9,042	8.0×	100.0	262
LiteLLM (Python)	1,134	1.0× (base)	100.0	2,044

Table 3. Sustained throughput at the 550/150 chat tier. Nexus rows run full zero-drop audit; the scanning-off row is the primary cross-gateway comparison. † Nexus 550 values are the median of a five-round tiebreak series (§3); every raw round is published. Other cells are the best of two converged rounds. ‡ agentgateway ran as a bare proxy (request-log store disabled; enabling it exhausts memory at these rates) — a pure-forwarding ceiling, not comparable to the audit gateways (§5).

One gateway is faster: **agentgateway**, at 76,421 req/s — but it ran as a bare proxy with its request-log store disabled, because enabling it exhausts memory at these request rates (§5). Among gateways that durably audit under load, Nexus leads: at 39,754 req/s (median of a five-round series) with full zero-drop audit it is 72% ahead of Bifrost, the nearest such competitor. With inline content scanning on (22,819 req/s), Nexus lands just below Bifrost's scanning-free 23,080 — the measured price of inspecting every request and response inline, on top of durable audit.

3. Examples: TensorZero runs its ClickHouse observability sink (no other gateway runs an equivalent), which collapses its large-body tier; Kong's content scanner is enterprise-only and was not tested; Kong's upstream connection pool ran at its stock default. All are noted in Appendix A.

4.2 Throughput across all tiers



Gateway	128 / 128	550 / 150	~12.5k
agentgateway ‡	88,549	76,421	19,688
Nexus — scanning off	**36,229**	**39,754**†	**12,147**
Bifrost	28,514	23,080	9,099
Nexus — scanning on	25,481	22,819†	5,909
TensorZero	24,361	21,735	**465***
Kong	21,059	19,566	10,273
Portkey	9,585	9,042	4,782
LiteLLM	1,129	1,134	1,096

Table 4. Sustained RPS by gateway × tier, non-streaming. Nexus rows shaded. † median of a five-round tiebreak series (§3). ‡ agentgateway = bare proxy in these runs (log store disabled; §5). Other cells are the best of two converged rounds.

* TensorZero's 12.5k non-streaming tier **collapsed reproducibly** across re-runs — throughput fell toward zero (one round measured 0 req/s) with a p99 tail reaching ~91 seconds — as its ClickHouse observability sink became the bottleneck ingesting ~50 KB inference rows. No other gateway ran an equivalent sink. We publish the collapsed figure as-is rather than discard the tier.

agentgateway tops every tier on raw forwarding, with its log store disabled (§5). Among the gateways that durably audit, Nexus leads — widest at the routing and chat tiers and narrowing at 12.5k, where larger bodies cost more to audit and scan (the by-design cost of durable audit and inline compliance at scale, shown explicitly so readers can judge it for their own workload).

4.3 Streaming (relay efficiency)

> Read this section as relay efficiency, not user latency: the mock does not pace tokens, so absolute streaming numbers are an artifact of an instant upstream (§3). The ranking still shows how efficiently each gateway forwards a stream. One important distinction, developed in §4.6: for gateways that keep up, the low absolute latency here

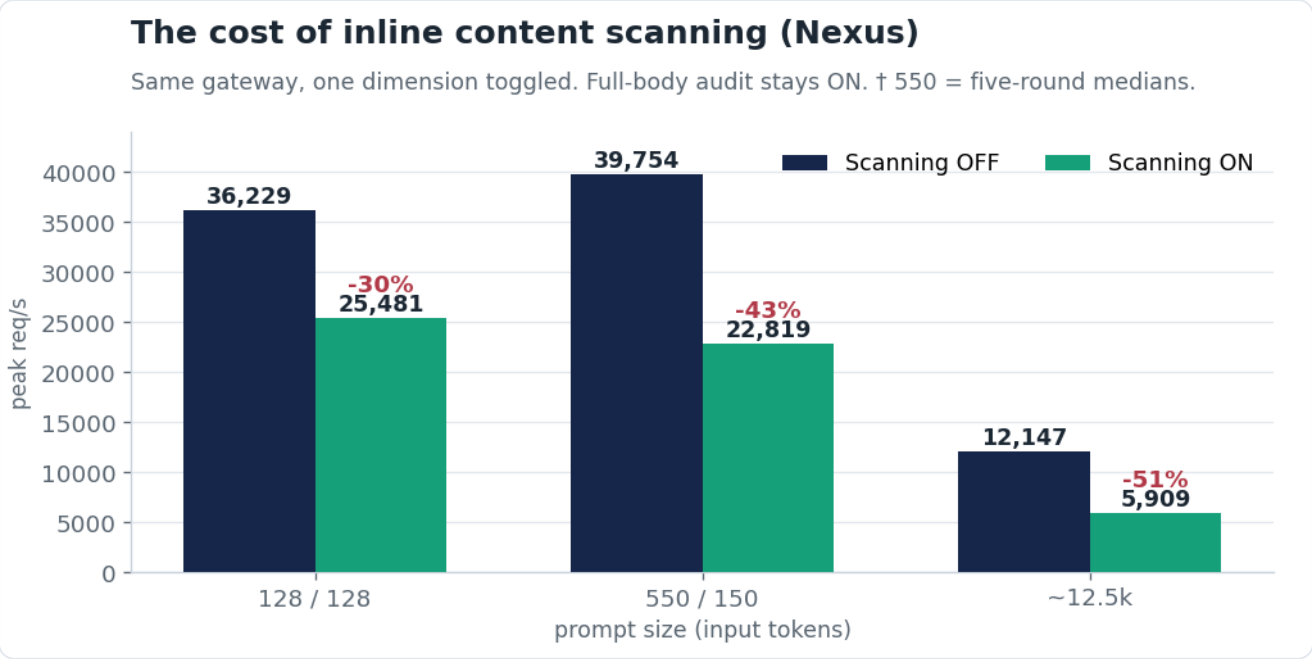
is that relay artifact; for gateways that saturate (LiteLLM, Bifrost), the latency growth under load is real queueing in the gateway, not an artifact.

Gateway	128	550	~12.5k
agentgateway ‡	16,171	9,488	13,585
Nexus — scanning off	**12,214**	**7,338**	**9,051**
Nexus — scanning on	7,547	3,909	4,665
Kong	9,804	6,401	7,056
TensorZero	6,192	2,402	1,798
Portkey	4,318	3,154	3,084
Bifrost	3,760	1,318	1,449
LiteLLM	178	101	203

Table 5. Peak streaming RPS by gateway × tier (best observed across two runs). ‡ agentgateway = bare proxy in these runs (log store disabled; streaming became unstable above 4,000 req/s — §5).

Nexus (scanning off) has the lowest SSE relay overhead among the audit gateways — an efficient chunk-by-chunk relay with immediate flushing in this test setup. Scanning on erases that edge on streaming, since the stream is held back and re-normalized at checkpoints; that cost is heaviest here and is the price of inspecting streamed content.

4.4 The cost of content scanning



This is the tradeoff the report exists to quantify. Enabling the five content hooks compiles ~423 rules into Vectorscan automata and runs a full-body scan of every request and response. The cost is real and grows with body size:

Tier	Scanning off	Scanning on	Change
128 / 128	36,229	25,481	**−30%**
550 / 150	39,754†	22,819†	**−43%**
~12.5k	12,147	5,909	**−51%**

Table 6. Nexus throughput, scanning off vs on, non-streaming. † 550 values are the medians of a five-round tiebreak series (§3).

> Two honest qualifiers. First, body capture is the cheap part of governance: storing full request and response bodies stayed within run-to-run variance, because it happens off the request path — the cost above is scanning, not audit. Second, the benchmark payload is benign, so scans abstain before any match-triggered redaction work; content that actually matched rules would impose additional cost.

4.5 Audit completeness (durability)

Throughput only counts alongside what a gateway actually persisted. On *every fully reconciled* Nexus run — both rounds, all tiers, scanning off and on, save the one round noted below — the rig reconciled every audit event: **dropped = 0, poisoned = 0**, with 100% of enqueued events landing in PostgreSQL, the event bus, or the on-disk spill buffer.

> **What “zero-drop” (“lossless”) means here.** Every created audit event was accounted for after each run — either durably persisted to PostgreSQL, retained on the event bus for downstream delivery, or durably captured in the on-disk spill buffer — and no created event was dropped or poisoned. This establishes pipeline-losslessness (nothing is shed under load); it does not, on its own, prove every event reached the final system of record before the run ended.

Run	Events enqueued	Dropped	Spilled to disk	Result
550 chat, scanning off	**6,971,885**	**0**	**3,126,897**	**Zero-drop**
550 chat, scanning on	4,751,752	0	95,323	**Zero-drop**
128 routing, scanning off	6,661,699	0	1,484,188	**Zero-drop**
~12.5k long-context, scanning off	1,418,816	0	1,059,480	**Zero-drop**

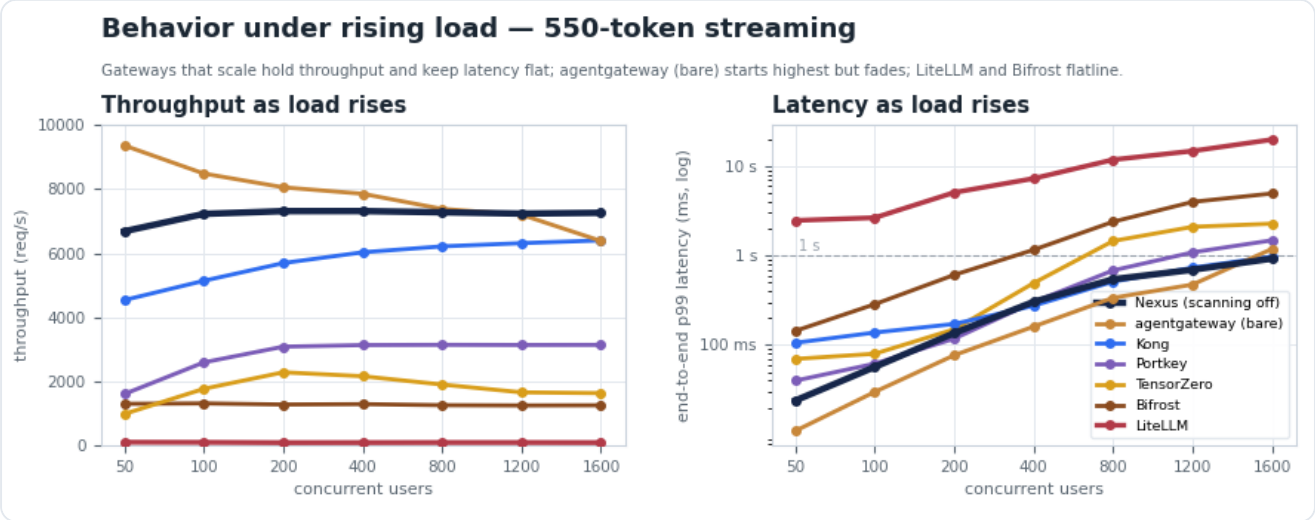
Table 7. Representative audit reconciliation from one fully reconciled round per cell (non-streaming); the 550 throughput headline uses the five-round median (§3), and these rows are not from the median rounds. Spill > 0 means the in-memory queue overflowed under load and events were written to disk — not dropped.

The point: Nexus hit the throughput numbers in this report *while losing zero audit records*. Under overflow it spills to disk by design (a zero-loss spill-block path) rather than dropping data. By contrast, the highest-throughput, less-governed proxies persisted little or nothing in these runs — their throughput figures describe less per-request work.

Two transient anomalies from this run, disclosed in full: in one round of the 12.5k *streaming* tier with full-body audit, Nexus briefly back-pressured to ~0 req/s for ~110 s at high concurrency (requests queued, not dropped; ok stayed ~99.8%) — it did not reproduce, and the paired round was clean. **Disposition:** root cause identified as NATS JetStream flush back-pressure at peak audit volume; a fix is planned. No effect on the reported throughput or durability figures. In one other round the gateway’s metrics counters were not scrapable at flush time, so a numeric dropped=0 could not be computed for that single round, though the audit data still landed (PostgreSQL + on-disk spill) and nothing errored at the client; every other round reconciled to dropped=0. **Disposition:** the metrics scrape raced the flush window; an instrumentation fix is planned so the count is always computable. Both items are tracked internally.

4.6 Behavior under load

Peak throughput is a single point. What an operator actually lives with is how a gateway behaves as concurrency climbs: does it keep serving and stay responsive, or does it stop scaling and let latency run away? The figure below tracks both at the 550-token streaming tier, as load rises from 50 to 1,600 concurrent users.



Three distinct behaviors emerge. Nexus and Kong (and, more modestly, Portkey) hold up — throughput stays high and end-to-end p99 latency stays near or below one second. agentgateway, as a bare proxy, starts highest but **fades** as concurrency rises (9,362 → 6,382 req/s, −32%), dropping below Nexus at the top of the ramp. LiteLLM and Bifrost **saturate early**: throughput flatlines while latency climbs into seconds.

Gateway	RPS @ 50	RPS @ 1,600	Scales with load?	p99 @ 1,600
agentgateway ‡	9,362	6,382	Starts highest, fades (−32%)	1.2 s
Nexus (scanning off)	**6,684**	**7,261**	**Holds high (knee 7,338)**	**0.9 s**
Kong	4,536	6,400	Yes	1.0 s
Portkey	1,604	3,139	Yes (modest)	1.5 s
TensorZero	982	1,633	Peaks then fades	2.3 s
Bifrost	1,300	1,253	**No (flat)**	**4.9 s**
LiteLLM	101	89	**No (flat)**	**19.7 s**

Table 8. 550-token streaming as concurrency rises from 50 to 1,600 users (round v1). p99 is end-to-end request latency at the top of the ramp. ‡ agentgateway = bare proxy in these runs (log store disabled; §5).

> **Why “100% ok” can be misleading.** Every gateway shows 100% success at these stages, but the load generator is closed-loop with *no request timeout* — a response that takes 16 seconds still counts as a success. Under any realistic client timeout or SLA, LiteLLM’s ~20 second streaming responses (and, to a lesser degree, Bifrost’s ~5 second tails) would surface as failures, not successes.

Per-gateway notes

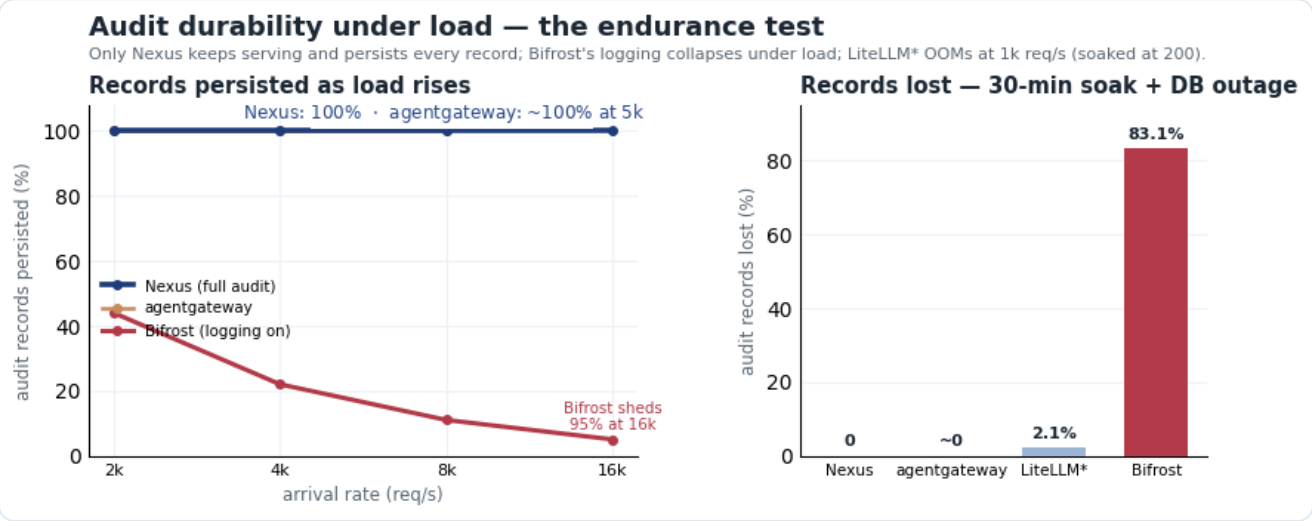
- **agentgateway** (bare proxy in these runs) starts with the highest streaming throughput but declines steadily as concurrency rises — 9,362 req/s at 50 users to 6,382 req/s at 1,600 (−32%), finishing below Nexus — with p99 around 1.2 seconds at the top of the ramp. Its 100%-ok serving here is with nothing persisted; its behavior with logging enabled is measured in §5.
- **LiteLLM** does not scale on streaming: throughput holds at ~90–100 req/s across every concurrency level while end-to-end latency climbs past **19 seconds**. This is real saturation queueing in the gateway, not an artifact of the mock (which responds instantly). Non-streaming LiteLLM is comparatively well-behaved (~1,100 req/s, ~2 s p99).
- **TensorZero** collapses at the largest payload: its ~12.5k non-streaming tier failed **reproducibly across re-runs** — throughput toward zero (one round 0 req/s) with a p99 tail to **~91 seconds** — as its ClickHouse observability sink chokes on ~50 KB rows. (At 128 and 550 tokens it is fast and stable; the failure is body-size specific.)
- **Bifrost** is competitive on non-streaming but weak on streaming: throughput stays flat around 1,270 req/s while p99 latency rises to ~5 seconds at high concurrency.
- **Kong** and **Portkey** scale cleanly and stay responsive; Kong is the most robust of the comparison gateways.
- **Nexus** sustains the highest streaming throughput of any audit gateway — and the highest of any gateway, including the bare proxy, at the top of the ramp — while holding p99 near one second, even with full zero-drop audit running.

> **A note on our own numbers, for symmetry.** Throughout this report, non-streaming figures are taken at each gateway's *saturated* (highest-concurrency) stage — the conservative number. That means Nexus's headline non-streaming results *understate* its peak: at low concurrency Nexus reaches higher still (e.g. ~45,000 req/s at 50 users on the 128-token tier, versus the ~36,000 we report at saturation). Streaming throughput is reported at the stable knee, since saturation RPS is noisy run-to-run; this section shows the full curve behind that single number.

5 Audit durability under load

The point of durable audit is that the record survives the moment you most need it: peak load and an infrastructure hiccup. A separate endurance test measured exactly that — and it is where the cost-of-governance thesis becomes concrete.

Two phases, four gateways in five configurations (Nexus in its scanning-off and scanning-on configurations, Bifrost, LiteLLM, agentgateway), each on its own box with a dedicated PostgreSQL. **Phase A** ramps the arrival rate to find where each gateway stops serving and how much of its audit it still persists. **Phase B** holds 5,000 req/s for 30 minutes and cuts the backend database for 30 s at minute 15. The headline metric is **records lost = completed − persisted** — requests the client was told succeeded but that never reached durable storage.[4]



Phase B — 30-minute soak at 5,000 req/s, database cut mid-run

Gateway	Soak rate (req/s)	Served 30 min	Expected	Persisted	Lost	Lost %
Nexus (scan-off & scan-on)	**5,000**	**yes**	**9,000,000**	**9,000,000**	**0**	**0.0000%**
agentgateway (logging on)	5,000	yes	9,000,000	8,999,929	71	0.0008%
Bifrost (logging on)	5,000	yes	9,000,000	1,517,085	**7,482,915**	**83.14%**
LiteLLM — reduced-rate soak	**200**	yes	360,000	352,463	**7,537**	**2.09%**

Table 9. Records lost in a 30-minute endurance soak with the backend PostgreSQL stopped 30 s at minute 15 (non-streaming). Worst of n≥2 rounds. LiteLLM is separated below the rule: it could not sustain the 5,000 req/s soak (OOM at 1,000; Table 10) and was soaked at 200 req/s — 25× lighter load, so its loss rate is **not comparable** to the rows above it.

4. The test ranks nothing and is asymmetric by construction: these gateways do different per-request work, so raw throughput is not the axis — survival of the audit record is. A bare proxy that persists nothing is legitimately cheaper; whether that is acceptable depends on whether you need the record. Full Phase A/B ladders and raw per-step output are in the published repo (github.com/AlphaBitCore/llm-gateway-benchmark).

On **streaming** the gap widens: Bifrost lost **99.5%** of records (with a p99 of ~900 seconds), while Nexus again lost zero. Nexus recovers the database outage by spilling to its on-disk buffer and replaying — the 30-second cut costs latency, not records.

Phase A — where each gateway stops serving, and what it still persists

Gateway	Serving ceiling (non-str)	Serving ceiling (stream)	Audit persisted under load
Nexus (scan-off / scan-on)	**16,000 req/s**	**8,000 req/s**	**100% at every rate**
agentgateway (logging on)	8,000 req/s	4,000 req/s	100% where it serves; store exhausts memory at matrix-benchmark rates
Bifrost	16,000 req/s	**never served normally**	**44% → 5% as load rises (sheds 95% at 16k)**
LiteLLM	**500 req/s (OOM at 1k)**	**never served normally**	100% only within its narrow envelope

Table 10. Highest arrival rate each gateway served normally (both rounds), and the share of audit records it durably persisted under load.

> **What this shows.** Raw throughput and durable audit are different questions. agentgateway is fastest as a bare proxy, and with logging enabled at 5,000 req/s it was near-lossless (71 records) but not zero-loss — and its log store does not operate at matrix-benchmark rates at all. Bifrost stays up but its best-effort logging sheds most records under load; LiteLLM falls over early. Only Nexus keeps serving *and* keeps every record, including through the database outage — because its audit path is bounded, off the request path, and spills to disk instead of dropping or ballooning memory.

6 Key takeaways

1. **The thesis: governance is the differentiator, not raw speed.** The only gateway faster than Nexus (agentgateway) was measured as a bare proxy; its request-log store does not operate at benchmark-scale rates, and at 5,000 req/s it was near-lossless but not zero-loss (§5). Among gateways that can durably audit under load, Nexus leads on throughput — 39,754 req/s (five-round median), 72% ahead of Bifrost — while quantifying what each governance layer costs.
2. **Durability is verified — and it is where the field separates.** In a 30-minute soak through a database outage, Nexus lost **0** of 9,000,000 records; agentgateway's logging was near-lossless at that rate (71 lost); Bifrost's "logging on" lost **83%** (99% on streaming); and LiteLLM exhausted memory at 1,000 req/s (§5).
3. **Nexus degrades gracefully; some competitors don't.** As load rose, Nexus and Kong held end-to-end latency near one second, while LiteLLM's streaming latency climbed past 19 seconds with no throughput gain and TensorZero's 12.5k tier collapsed reproducibly (a full round to 0 req/s). Under common production timeouts and SLAs, those conditions would often surface as failures rather than merely slower responses.
4. **Content scanning is a real, bounded tradeoff.** Inline PII/policy scanning costs roughly 30–51% of non-streaming throughput (more on streaming) — quantified here so it can be enabled per route with a known cost. Body capture, separately, is nearly free.
5. **Everything here is reproducible and disclosed.** The rig is published in full, Nexus's role as author is stated plainly, and known asymmetries are documented (Appendix A).

> **Scope.** This is a fair head-to-head *between these gateways* on an instant shared mock — which rewards relay/IO efficiency and suppresses real-model latency and cross-provider translation. It is not a substitute for testing against your own provider and traffic, and the figures are not directly comparable to vendors' own single-box published numbers. Two rounds per cell (five on the tiebroken 550 tier) are indicative, not statistically powered.

Appendix A Per-gateway configuration

Full disclosure so results are auditable. OS-level tuning (sysctl, socket and file-descriptor limits) and storage throughput were identical on every box.

Gateway	Configuration as run	Notable asymmetry (disclosed)
Nexus	**Routing + virtual-key auth + quota; verified zero-drop full-body audit (async, off request path); content hooks swept off/on. Hot-path tuning targets its own audit/quota subsystems.**	**Authored the rig; benchmarked in a separate session on the same instance type. Audit is async/off-path by design, so it costs little hot-path latency.**
Bifrost	Official prebuilt gateway (`npx @maximhq/bifrost` , transport v1.5.16). Client pool 5,000, drop_excess_requests off, retries 0, request timeout 300 s; PostgreSQL config + logs stores (idle 10 / open 50); response cache and vector store disabled; NOFILE 1,048,576. Streaming uses the transport binary's default SSE flush (no override).	Logging sits nearer the request path than Nexus's async audit. The full streaming-relevant config is shown so the flat-streaming result (§4.6) cannot be attributed to a tuning omission. Maxim (Bifrost's maintainer) was notified of the §5 record-loss findings before publication, with the standing re-run offer (§3). Maxim responded that a log-retention configuration exists; we requested it — noting we found no documented settings for logging queue depth, batch size, flush interval, backpressure, or delivery guarantees — and it had not been provided as of publication. The offer remains open.
Kong	ai-proxy + file-log + rate-limiting (request counting; limit set so high it never throttles).	File-log is best-effort, not zero-drop. Content scanner is enterprise-only (not tested). Upstream connection pool left at stock default — the value Kong ships and its install docs leave in place, i.e. what a typical operator runs; the standing offer (§3) covers a Kong-suggested pool setting.
TensorZero	Proxy with observability on (full inference rows → ClickHouse), async batched writes (500 ms / 1000 rows).	Only gateway with a ClickHouse sink; its 12.5k non-streaming tier collapses reproducibly as a result (see §4.2).
Portkey	PM2 cluster, one worker per vCPU; no DB, cache off (largely stateless OSS path).	Persists nothing on the hot path.
LiteLLM	FastAPI + Uvicorn, workers = vCPUs; message logging off (pure proxy).	Python/GIL bounds per-box throughput (~1.5k RPS); scales horizontally. Used as the 1.0× baseline.
agentgateway	Rust v1.3.1, single process, standalone `llm` mode; request-log store **disabled** (source: `log_store.rs` at the v1.3.1 release tag — unbounded async writer, batch 64, no backpressure / spill / cap).	Throughput rows ran as a bare proxy (log store disabled). With logging on, the store's unbounded writer queue exhausts memory (OOM) at matrix-benchmark request rates; at 5,000 req/s (§5) it operated and was near-lossless (71 of 9,000,000 lost) but not zero-loss through the database outage. Read matrix throughput as a pure-forwarding ceiling. Reported upstream before publication: agentgateway#2463. Maintainers responded within a day with a mitigation (PR #2484: batch 64→16,384, Postgres COPY; ~50k QPS sustained writes per its author) that raises the drain rate but, per the maintainer, retains the unbounded enqueue path. The version benchmarked here (v1.3.1) predates that change.

Table A1. Configuration and known asymmetries per gateway.

On Nexus's tuning. The performance settings applied to Nexus target subsystems the other gateways largely do not have — its durable-audit pipeline, quota accounting, and credential-stats tracking. Tuning machinery that only Nexus runs is not an edge taken over competitors; it makes Nexus's own governance features perform. General-purpose settings (e.g. OS socket tuning, storage throughput) were identical across all boxes; where a competitor ran at a library or stock default, it is noted above.

Appendix B Environment & reproducibility

The benchmark is fully infrastructure-as-code: cloud templates provision the machines, automation installs and tunes every box host-native, and scripts orchestrate each gated load-test cycle. A run reproduces by re-running, not by hand-editing a box.

Parameter	Value
Compute (per gateway)	AWS c6i.4xlarge (16 vCPU), Amazon Linux 2023
Topology	One box per gateway + one shared mock box + one load-generator box; same subnet / AZ, us-east-1
Storage	Identical gp3 root volume, reconciled to 1000 MiB/s on every box
Upstream	Mock provider, OpenAI-format echo, instant response, no inter-token delay
Load	Project Go load generator, closed-loop (fixed concurrency per stage), 15 s warmup excluded
Rounds	Two per cell (v1 / v2), re-run when rounds disagreed >10% or ok < 99.5%; n=2, indicative not statistically powered. The one tier that stayed noisy (Nexus 550 non-streaming) was settled with a three-round tiebreaker and is reported as the five-round median.
Peak metric	Non-streaming = last (saturated) stage; streaming = max-throughput (knee) stage
Nexus audit	Loss-mode = spill-block (zero-loss); dropped=0 verified for all fully reconciled runs

Table B1. Test environment.

REPRODUCE

Provision the infrastructure, install/tune every box, then run the campaign — each cycle runs as a gated pipeline (`clean → setup → restart → health → cooldown → run → verify-audit → report`). The full rig, load profiles, and raw per-stage results for both rounds are published at **github.com/AlphaBitCore/llm-gateway-benchmark**.

ABOUT ALPHABITCORE

AlphaBitCore (ABC) builds Nexus, an AI governance platform that applies a single policy and compliance engine across three interception layers — an in-process SDK, the network proxy benchmarked in this report, and an endpoint agent. AlphaBitCore built and operates the open benchmark rig used here and publishes it in full, including raw per-stage results, at github.com/AlphaBitCore/llm-gateway-benchmark.

© 2026 AlphaBitCore, Inc. All rights reserved. Nexus is a product of AlphaBitCore. Methodology, disclosures, and configuration notes above are part of the published report. Competitor mechanisms are inferred from public docs/code; Nexus mechanisms were checked against source. Vendors' own published benchmarks use different setups and are not directly comparable to this rig.