# Contract Drift in a Multi-Protocol LLM Gateway: Black-Box Measurement of PLocalSwitch v0.2.27

Daozhendong Yang
School of Intelligent Science and Engineering,
Yunnan Technology and Business University, Kunming, China
hi@yxpil.com

## Abstract

Multi-protocol LLM gateways promise to be many things at once: an OpenAI-compatible front door, a translator to vendor-native APIs, a failover layer, and a billing system. Each promise is a contract, and each contract is testable. This paper measures one such gateway, PLocalSwitch v0.2.27 at commit `b9ce157` (8,642 Rust lines across 89 files), against the contracts it advertises, using a black-box harness that runs the shipped desktop binary against 27 local mock upstreams and exports the gateway's own SQLite ledger. The data plane is genuinely good: all 20/20 non-streaming combinations return byte-identical text. The control and envelope planes are not. Only 5/20 streaming responses use the envelope the client's protocol expects — Anthropic and Ollama clients receive OpenAI `data:` chunks, and a Gemini `streamGenerateContent` call receives a single non-streaming JSON body — and the `model` field is empty in 16/20 non-streaming responses. Tool calls vanish in both directions on the Gemini ingress. Configured retry policy is not consulted: a `false` switch still retries, and a sub-attempt budget of 6 becomes 24 upstream POSTs. A mid-stream premature close is reported as HTTP 200 with a synthetic `[DONE]` and zero billed tokens. Per-alias cache opt-out is ignored, cache purge is a no-op, and the cache statistics endpoint reports 0 entries while the cache is serving hits. The management plane answers 0 where the same process wrote 91 traces. We classify these into four kinds of drift — envelope, vocabulary, policy and telemetry — and report the negative results, including a headless build configuration that does not compile (12 errors), in the hope that gateway authors find the test method worth copying and the findings worth fixing.

**Artifact.** The gateway is public at `https://github.com/yxpil/PLocalSwitch` (yxpil, 2026) at commit `b9ce157`. The harness (`scripts/gateway_lab.py`, `scripts/measure.py`, `scripts/audit_repo.py`), the raw artifact `plocal_measure.json` under `scripts/results/` (generated 2026-10-21), the sanitized lab configuration, the compiled `numbers.tex` and every table and figure in this paper are in the paper directory. No number here was typed by hand; Section 3 gives the reproduction commands.

## 1 Introduction

A local LLM gateway has an unusually wide surface of promises. It must answer the OpenAI `/v1/chat/completions` contract (OpenAI, 2024a) for clients that know nothing about the upstream; it must speak Anthropic Messages (Anthropic, 2024), Gemini `generateContent` (Google, 2024), Ollama native (Ollama, 2024) and the OpenAI Responses API (OpenAI, 2024b) to providers; it must fail over when one provider errors; it must cache, route, and bill; and it must expose enough of itself that an operator can tell what happened. PLocalSwitch makes all of these claims in its own start-up banner, which prints *"100% OpenAI v1 compatible, 13 upstream adapters"*, *"dual-ledger (client / upstream) + tiktoken audit"* and *"protocol sniffer + retry + SSE normalize"* before it binds a socket.

That banner is a specification, and it is checkable. This paper checks it. The question is not "is this gateway fast?" but "where does the implemented behaviour diverge from the advertised contract, and what does that divergence cost an operator?" The answer, briefly: the translation core works and the surrounding contracts do not. All 20 non-streaming inbound/upstream combinations return the exact text the upstream produced, so the five native adapters (Table 2) are real software, not a facade. But the response *envelope*, the *configuration vocabulary*, the *policy switches* and the *telemetry endpoints* each drift from what the same build advertises, and each drift is reproducible in under a second on a laptop.

**Contributions.**

**C1.** A black-box measurement method for LLM gateways: run the shipped binary against wire-accurate mock providers, count POSTs at the socket, and cross-check every management answer against the gateway's own exported database (§3).

**C2.** A quantified conformance matrix over 4 inbound protocols × 5 upstream protocols: 20/20 non-streaming text fidelity, 5/20 streaming envelope fidelity, and 16/20 responses with an empty `model` field (§4).

**C3.** Evidence that four distinct kinds of contract drift — envelope, vocabulary, policy, telemetry — coexist in one codebase with one root cause: the contract is written in

prose and banners rather than in tests (§5).

**C4.** A set of negative results worth more than the successes: a retry policy that is not read, a sub-attempt budget that is floored at 24 when configured 6, a truncated stream billed as success, a cache purge that purges nothing, a per-alias cache opt-out that is ignored, a protocol-hint vocabulary mismatch that silently changes the wire protocol, and a headless build feature that does not compile (§4.4–§4.9).

**Non-goals.** We do not evaluate translation quality against real provider traffic: every upstream here is a mock, chosen precisely so that the wire is fully observable and the experiment repeats. We do not measure throughput limits, TLS termination, multi-instance behaviour, or prompt-injection resistance. We do not claim the gateway is unusual in kind — LiteLLM (BerriAI, 2024) and One-API (songquanpeng, 2024) ship the same feature list at larger scale — only that its own advertised contracts are measurable and measurable failures are present.

# 2 System Under Study

PLocalSwitch is a Rust (tokio-rs, 2024b) gateway embedded in a Tauri (Tauri Programme, 2024) desktop application: an axum (tokio-rs, 2024a) server, a reqwest (McArthur, 2024) client pool, a SQLite (SQLite Development Team, 2024) trace store, and a mini-moka (moka-rs, 2024) cache, driven by a YAML configuration that declares client keys, model aliases, node groups and policies. The crate is 8,642 lines of Rust plus 5,074 lines of TypeScript front end, in 241 files, with 14 inline `#[test]` functions and 27 HTTP routes.

## 2.1 One pipeline, four ingress vocabularies

Figure 1 shows the request path, which is shared by every inbound protocol. `execute_chat_pipeline` initialises a trace, checks per-key RPM/TPM, probes the cache, resolves the client's `model` string to a candidate list, and hands the request to the flex adapter, which owns the retry chain and the protocol sniffer. Inbound normalisation (`inbound_sniffer.rs`) maps four client vocabularies — OpenAI, Anthropic, Gemini, Ollama — onto an internal OpenAI-shaped IR, and outbound adapters map the IR onto provider wire formats.

## 2.2 Adapters are not equal

Table 2 decomposes the 14 protocol kinds. Only 5 adapter modules implement real translation (`anthropic_adapter.rs` is ~332 lines, `responses_adapter.rs` ~280); 8 forward an OpenAI-shaped body with the path substituted, which means a provider that does not in fact accept OpenAI JSON behind an OpenAI-shaped URL is mis-served rather than translated; and 1 (`bedrock_adapter.rs`, 23 lines) returns an error from both entry points, so the Bedrock Converse contract (Amazon Web Services, 2024) is unimplemented rather than deferred. The banner's "13 upstream adapters" and the 14 adapter modules differ by one, which is the kind of arithmetic a contract test would have caught.

## 2.3 What the code says about itself

The static audit (Table 1) finds 22 explicit `TODO`/`stub`/`placeholder` markers and 9 handlers whose bodies return a constant regardless of state — 8 of them in the management route module and the remaining 1 the embeddings handler, e.g.

```
pub async fn cache_purge_handler(...) -> Result<Response,
    ...> {
  Ok(Json(serde_json::json!({"purged": true})).
    into_response())
}
```

and 27 public functions that no call site references, including `detect_path` — the inbound path-detection helper that would have made the catch-all route correct (§4.5). These markers are not defects by themselves; they are a map of where the contract has not been implemented, which is why the dynamic results below are predictable from the source.

# 3 Measurement Method

## 3.1 Harness

The harness is `scripts/measure.py` with wire helpers in `scripts/gateway_lab.py`. It (i) starts 27 threaded HTTP mock servers on loopback, one per behaviour — five protocol-faithful providers, plus fixtures that answer 401, 400, 500, generic 404, "unsupported protocol" 404, and a chunked stream that ends after one delta without `[DONE]`; (ii) generates a lab configuration from the shipped `gateway.yaml` template with `http.listen` on a free port, an isolated SQLite file, in-memory cache, and one alias per scenario; (iii) launches the *shipped release binary* (`plocalswitch.exe`, 11,872,256 bytes, SHA-256 recorded in the artifact) as a subprocess; (iv) issues requests with `urllib` so no client library can hide a wire detail; (v) reads the gateway's SQLite file after the run; and (vi) writes one JSON artifact. Mocks count POSTs only, because the gateway probes `/v1/models` with GET at start-up and those probes would otherwise pollute every attempt count. Responses carry `Connection: close`; without it reqwest reuses sockets that the mocks have already closed and the failure scenarios report transport noise instead of policy.

Table 3 records the environment. The binary reaches `/v1/models` in 0.510 s, idles at 36.43 MB RSS, and — because it is a desktop application — creates a window handle (`31` models are published from the alias table and the empty upstream model lists).

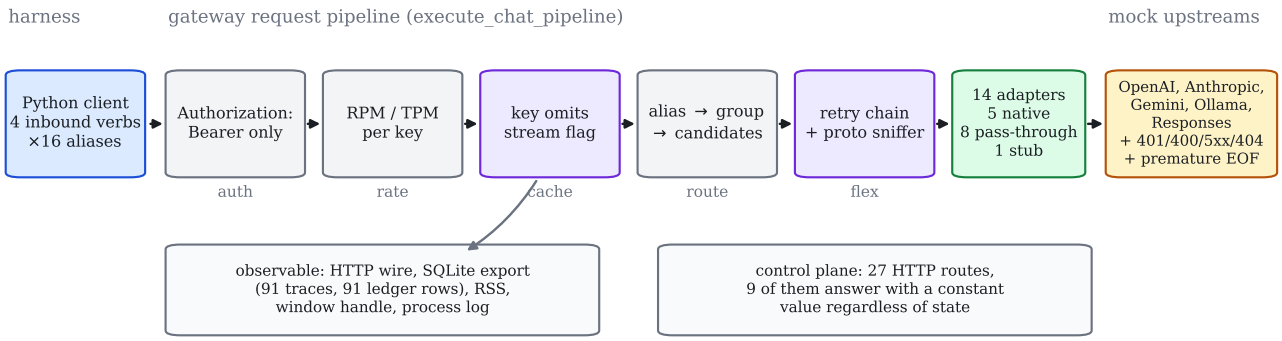



Figure 1: The measurement setup and the gateway's own request pipeline. The client issues the same logical request through four inbound vocabularies and five upstream protocols; every upstream is a mock that records what it received, so "did the provider see the tools?" is answered from the socket rather than from the logs.

Table 1: Static audit of the gateway crate by module. Stub markers are `TODO`, `stub`, `placeholder` or `not implemented` occurrences inside the module.

| Module | Files | LOC | Stub markers |
|---|---|---|---|
| gateway_api | 9 | 1,836 | 5 |
| backend_adapters | 15 | 1,444 | 4 |
| flex_adapter | 8 | 628 | 5 |
| router | 4 | 540 | 0 |
| observability | 6 | 446 | 1 |
| services | 5 | 543 | 0 |
| cache_pool | 5 | 293 | 2 |
| billing | 7 | 223 | 2 |
| node_quality | 5 | 131 | 3 |
| safety_runtime | 7 | 144 | 0 |
| optional_components | 5 | 15 | 2 |
| commands | 6 | 673 | 0 |
| (crate root) | 7 | 1,726 | 0 |
| **Total** | 89 | 8,642 | 24 |

### 3.2 Reproduction

```
cd papers/plocal-switch/scripts
python audit_repo.py            # -> data/audit.json
cargo build --release           # in the gateway repo
python measure.py               # -> plocal_measure.json
python make_tables.py           # -> numbers.tex + tables
python make_figures.py          # -> figures/*.pdf
```

`measure.py` refuses to start if any port is busy or another gateway instance is running, and deletes the SQLite file after exporting it; the committed artifact contains no API keys, and `scripts/data/lab_config.json` is the sanitized configuration with the upstream keys replaced by a lab constant.

### 3.3 Threats to validity

Three limits bound every number below. *First*, the upstreams are mocks written from the vendor documentation (OpenAI, 2024a,b; Anthropic, 2024; Google, 2024; Ollama, 2024), so a real provider that deviates from its own docs deviates from our fixture too; our claims are therefore about the gateway's handling of *documented* shapes, not about providers. *Second*, the gateway runs as a desktop binary with a GUI shell; we could not exercise the headless feature because it does not compile (§4.9), so all measurements include the Tauri process, its window, and its default feature set. *Third*, one host (Windows 11 10.0.26300 x64), one run per scenario, 25 samples for the latency study: we report medians and ranges, not confidence intervals, and we re-ran the whole harness after every harness bug rather than sampling.

## 4 Results

### 4.1 The data plane works; the model field does not

Figure 2 summarises the three grids. Text survives: in all 20 non-streaming combinations the client receives exactly the string the mock produced, and in all 20 streaming combinations the concatenated deltas reconstruct the same string. This is not trivial for five wire formats, and it is the result the design deserves.

Table 2: The 14 `ProtocolKind` variants and their adapters. Only five classes perform translation.

| Adapter class | Files | LOC range | Behaviour |
|---|---|---|---|
| native | 5 | 79–332 | translates request, response and SSE into the internal IR |
| passthrough | 8 | 22–26 | forwards an OpenAI-shaped body with the path substituted |
| stub | 1 | 23 | returns an error on both request and response paths |

Table 3: The measured artifact. The gateway runs as the shipped desktop binary; every upstream is a local mock so that the wire is fully observable.

| Item | Value |
|---|---|
| Repository commit | `b9ce157` (worktree clean: yes) |
| Gateway version | v0.2.27 (banner in `lib.rs`) |
| Release binary | 11,872,256 bytes |
| Binary SHA-256 (16) | `945c6f7e58276bc2` |
| Config SHA-256 (16) | `f9e4b55c123a6ccf` |
| Rust sources | 89 files, 8,642 lines |
| TypeScript sources | 49 files, 5,074 lines |
| Markdown docs | 15 files, 1,334 lines |
| Inline `#[test]` functions | 14 |
| HTTP routes | 27 |
| Host | Windows 11 10.0.26300 x64, Python 3.13.14 |
| Mock upstream nodes | 27 on ports 18,301–18,327 |
| Gateway listen | 127.0.0.1:18,187 |
| Spawn-to-ready | 0.510 s |
| Idle RSS | 36.43 MB |
| GUI window handle | 1,837,332 (present) |
| `/v1/models` entries | 31 |
| Traces / ledger rows written | 91 / 91 |

Table 4: Non-streaming text fidelity over 4 inbound protocols and 5 upstream protocols. A check mark means the returned text equals the upstream text; ∅ marks rows whose client-visible `model` field is empty or absent.

| Inbound | OpenAI | Anthropic | Gemini | Ollama | Responses |
|---|---|---|---|---|---|
| OpenAI | ✓ | ✓ | ✓ (∅) | ✓ | ✓ |
| Anthropic | ✓ (∅) | ✓ (∅) | ✓ (∅) | ✓ (∅) | ✓ (∅) |
| Gemini | ✓ (∅) | ✓ (∅) | ✓ (∅) | ✓ (∅) | ✓ (∅) |
| Ollama | ✓ (∅) | ✓ (∅) | ✓ (∅) | ✓ (∅) | ✓ (∅) |

The envelope is a different story. Table 4 marks with ∅ the 16 non-streaming responses whose `model` field is an empty string or absent. Only the 4 OpenAI-inbound rows carry a model name, and one of those (Gemini upstream) does not, because the Gemini denormaliser passes the model through a path that is never populated. A client that keys its own usage reporting, cache, or retry policy on `response.model` — which the OpenAI contract permits and most SDK examples do — silently groups 16/20 of these calls under an empty string.

## 4.2 Streaming ingress is normalised one way only

Table 5 is the clearest single result in this paper. The gateway translates upstream streams *into* OpenAI chunks, and then writes those chunks to the socket with server-sent-events framing (WHATWG, 2024) — `data: {...}` and a final `data: [DONE]` — regardless of how the client asked. For an OpenAI client that is correct. For an Anthropic client, which expects `message_start`, `content_block_delta` and `message_stop` events, it is a stream no Anthropic SDK parses. For an Ollama client, which expects newline-delimited JSON objects with `done: true`, it is likewise wrong. For Gemini, the mismatch is worse than a wrong envelope: `streamGenerateContent` is routed to a handler that ignores the streaming flag and returns a single `application/json` body, so a client that opened the call in streaming mode receives one non-streaming answer with `Content-Type: application/json`.

The source explains the pattern in one line: the streaming branch of the shared pipeline serialises `SseChunk` values and never calls `denormalize_response`, the function that exists precisely to render an answer in the client's protocol and that the non-streaming branch does call. The 15 mismatching cells are one missing call site, not five missing features.

## 4.3 Tool calls: the Gemini ingress loses them twice

Table 6 walks a tool round trip in both directions. The request side fails once: a Gemini client that sends `tools` with `functionDeclarations` reaches the mock with *no* `tools` key at all (1 of 8 cases), because `normalize_gemini` hard-codes `tools: None, tool_choice: None`. The response side fails twice more: a Gemini ingress receives text where a `functionCall` part should be, because `denorm_gemini`

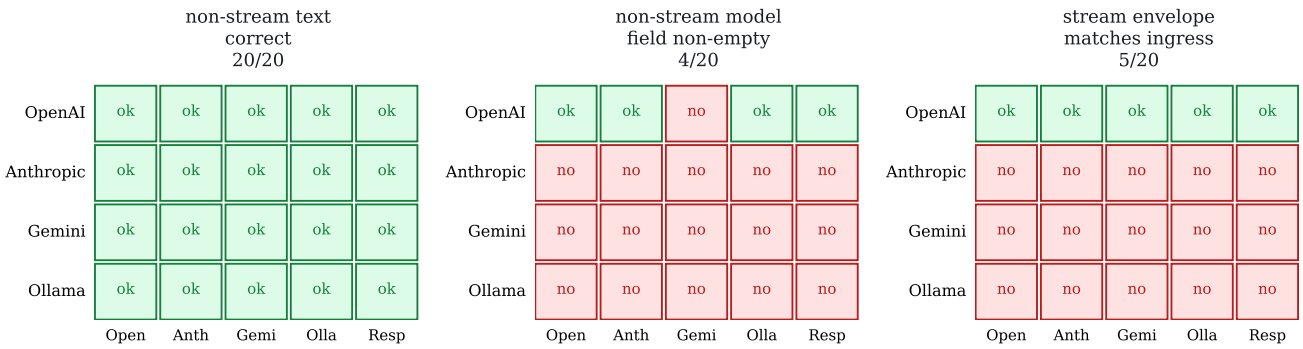



Figure 2: Conformance grids over 4 inbound protocols (rows) and 5 upstream protocols (columns). Text fidelity is perfect; the `model` field survives only on the OpenAI ingress; the streaming envelope matches the ingress protocol only on the OpenAI ingress.

Table 5: Streaming ingress envelope. Expected is the format the inbound protocol's own client library parses; observed is what the gateway emitted, which does not depend on the upstream protocol.

| Inbound | Expected envelope | Observed envelope | Matching | Text recovered |
|---|---|---|---|---|
| OpenAI | `openai_sse` | `openai_sse` | 5/5 | 5/5 |
| Anthropic | `anthropic_sse` | `openai_sse` | 0/5 | 5/5 |
| Gemini | `gemini_sse` | `json_non_stream` | 0/5 | 5/5 |
| Ollama | `ollama_ndjson` | `openai_sse` | 0/5 | 5/5 |

emits only `{ "text": ... }` parts, and an Ollama ingress receives a tool call whose `id` is dropped, so the client cannot construct the matching `role: "tool"` follow-up. Across the grid, 5 of 8 cases preserve a usable call `id`.

A silently dropped tool definition is the most expensive class of drift in this paper: the request succeeds, the model answers in prose, and nothing in the response says that the tools the client declared were never sent.

## 4.4 Failover: policy as prose, budget as floor

Table 7 and Figure 3 give four findings.

**(i) The `policy.retry_on.*` switches are not consulted.** With `auth_401_403 = false` and `bad_param_4xx = false`, a 401 and a 400 on the first node both produce one POST to the failing node and one to the healthy node, and the client sees HTTP 200. The retry loop's own comment states the intent ("any error moves to the next candidate"), so the policy struct is documentation. An operator who disables 401 retries because a bad key must not be replayed against other providers — a reasonable billing and security stance — is silently overruled.

**(ii) The sub-attempt budget is a floor, not a ceiling.** The configured `global_max_sub_attempts` is 6; a ten-node group in which every node returns 500 produces 24 upstream POSTs (`clamp(24, 48)` in the retry controller), touching 8 of 10 nodes at three protocols each, in 25.26 ms. The client then receives HTTP 502. The wall-clock cost is small here because the mocks are fast; against real providers each attempt is a round trip and, for some providers, a billable request.

**(iii) Protocol switching keys on error text.** A 404 whose body is `404 page not found` ends the chain and the client receives HTTP 400 — a status that RFC 9110 reserves for request syntax (Fielding et al., 2022), although nothing about the request was wrong. The same 404 with the body `unsupported protocol; use /v1/messages` triggers the alternate protocol, the chain continues to `/v1/messages`, and the client receives HTTP 200: text matching a keyword list (`unsupported`, `not support`, `protocol`, two Chinese phrases) decides whether a different wire protocol is tried. The heuristic is reasonable; the vocabulary is not, because the two upstreams differ only in prose.

**(iv) A truncated stream is billed as success.** When the upstream ends its chunked body (Nottingham and Fielding, 2022) after one delta and no `[DONE]`, the gateway locks the response as successful: the client receives HTTP 200, 1 data chunk, and a synthetic `data: [DONE]`, the second candidate is never tried, and the trace row records status 200 with zero billed tokens. A client cannot distinguish this from a model that answered in one word.

## 4.5 Vocabulary drift: `openai` is not `open_ai`

The last row of Table 7 is the most instructive single measurement in this paper. A node is configured with

Table 6: Tool-calling round trips. *Upstream saw tools* is decided by inspecting the JSON body the mock received; *lost* means the call reached the client with a `null` or empty `id`.

| Inbound | Upstream | Upstream saw tools | Client tool calls | ID preserved |
|---|---|---|---|---|
| OpenAI | OpenAI | yes | 1 | yes |
| Anthropic | OpenAI | yes | 1 | yes |
| Gemini | OpenAI | *no* | 0 | n/a |
| Ollama | OpenAI | yes | 1 | *lost* |
| OpenAI | Anthropic | yes | 1 | yes |
| OpenAI | Gemini | yes | 1 | *lost* |
| OpenAI | Ollama | yes | 1 | yes |
| OpenAI | Responses | yes | 1 | yes |

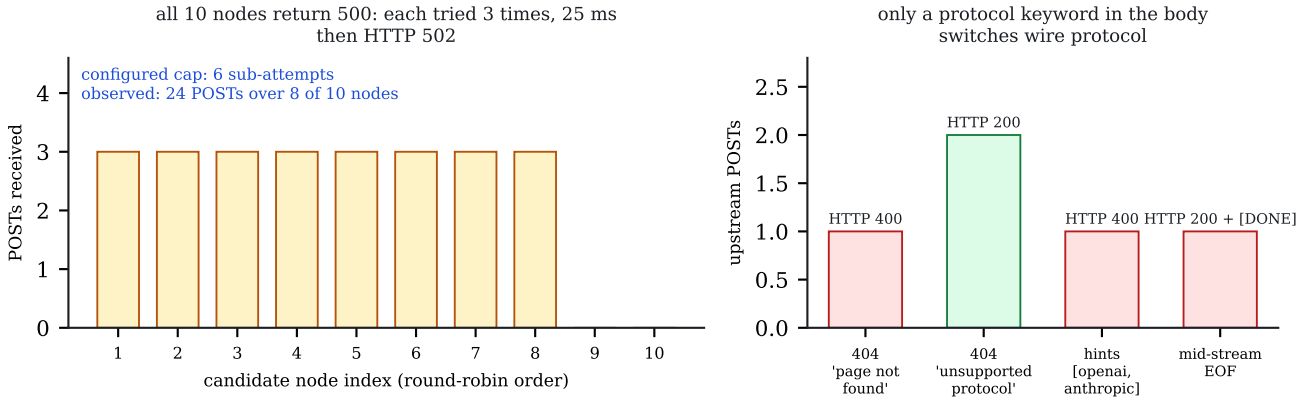



Figure 3: Left: with every node returning 500, each of the first eight candidates is tried three times (one per protocol in its sequence) for 24 upstream POSTs in 25.26 ms, although the configuration caps sub-attempts at 6. Right: protocol switching is triggered by a keyword in the upstream error body, not by the status code; the fourth bar is a silent truncation (HTTP 200 with a synthetic `[DONE]`).

`protocol_hints: [openai, anthropic]` and the upstream genuinely supports OpenAI and answers 404 to `/v1/messages`. The gateway's first request goes to `/v1/messages`, receives 404, does not try the declared protocol, and returns HTTP 400 — a failure caused entirely by the gateway's own protocol choice.

The cause is a naming convention. `ProtocolKind` derives `EnumString` with `serialize_all = "snake_case"`, so the canonical token for the OpenAI kind is `open_ai`; `"openai"` does not parse. The single-hint case accidentally works, because `candidate_sequence` appends the fallback list `[OpenAI, Anthropic, Gemini]` to whatever parsed, and with nothing parsed OpenAI lands first. As soon as a second hint parses, the fallback is appended *behind* it, and the declared first protocol — which never parsed — is skipped. A one-token typo in a config file, undetectable by the parser because unrecognised hints are dropped with `filter_map(|h| h.parse().ok())`, changes which protocol the provider is addressed with. The same vocabulary split appears in the ingress: the gateway accepts `x-api-key` nowhere (Table 8: Anthropic-style key → HTTP 401, Gemini query key → HTTP 401) while its own Anthropic adapter sends `x-api-key` upstream, and the `/manage/audit` endpoint rejects `alarm_only=1`, a spelling its own query deserialiser does not accept.

## 4.6 Cache: opt-out ignored, purge a no-op

Table 9 sends the same request three times through an alias that declares `cache_enable: false`. The upstream is hit once. The second and third requests are served from cache (`is_cached = 1`), so the per-alias opt-out is not consulted — the pipeline reads only the global `cache_pool.enabled`. Between the second and third request the harness calls `/manage/cache/purge`, which answers `{"purged": true}`; the third request is still a cache hit. `/manage/cache/stats` reports 0 entries throughout, including after the hits, so the endpoint cannot be used to size or debug the cache it is describing.

The accounting consequence is subtler. The cold row charges `0.002` CNY for 1,500 tokens at the configured rate. The two cached rows carry the same `billed_usage` but an empty `resolved_model`, so the ledger row's model is empty and the charge resolves to `0.000`. Whether a cache hit should be charged is a policy question the configuration answers per alias (`charge_on_cache_hit`); the implementation answers it by losing the model name, which makes the intended policy unreachable and leaves 3,000 billed tokens attached to rows that cannot be attributed to a model.

Table 7: Failover behaviour against controlled upstream failures. The second column is the status returned to the client; the third counts POSTs that reached the mocks.

| Scenario (configured policy) | Client | Upstream POSTs | Outcome |
|---|---|---|---|
| `retry_on.auth_401_403=false`, 401 on first node | 200 | 1+1 | retried anyway |
| `retry_on.bad_param_4xx=false`, 400 on first node | 200 | 1+1 | retried anyway |
| all 10 nodes return 500 | 502 | 24 (cap configured 6) | 25.26 ms wall |
| stream header 500 on first node | 200 | 1+1 | failover, text intact |
| `retry_on.sse_premature_close=false`, mid-stream EOF | 200 | 1+0 | 1 chunk then `[DONE]` |
| upstream 404 body `404 page not found` | 400 | 1 | no protocol switch |
| upstream 404 body `unsupported protocol; use /v1/messages` | 200 | 2 | switched to `/v1/messages` |
| hints `[openai, anthropic]`, OpenAI-only server | 400 | 1 | first request went to `/v1/messages` |

Table 8: Auxiliary endpoints, authentication vocabulary, routing collision and build configuration probes.

| Probe | Result |
|---|---|
| `/v1/models` | HTTP 200, 31 entries |
| `/v1/embeddings` | HTTP 500 from a constant error handler |
| `/api/generate` (Ollama) | HTTP 200 with an empty `model` field |
| OpenAI body on `/v1/messages` | HTTP 200, answered as an Anthropic `message` with empty `model` |
| OpenAI body on unknown `/v1/auto-probe` | HTTP 200, body sniffer scored Anthropic 80 versus OpenAI 55 |
| Unknown alias `no-such-alias` | HTTP 400 |
| Missing `Authorization` header | HTTP 401 |
| Anthropic `x-api-key` only | HTTP 401 |
| Gemini `?key=` only | HTTP 401 |
| OpenAI `Bearer` header | HTTP 200 |
| Two aliases sharing `real_model` | both resolved to the same group: +2 versus +0 upstream hits |
| Headless build `-no-default-features -features gateway-server` | exit 101, 12 `E0433` errors: `services` is gated behind `desktop-shell` |

## 4.7 Routing: two aliases, one model, one winner

Two aliases with different `group` fields but the same `real_model` (`mock-collide`) both resolve to the first alias's group: the OpenAI node receives 2 extra POSTs and the Anthropic node 0. The router builds a model-to-group catalogue first and consults it before the alias's own group, so an alias is a display name, not a routing constraint. An operator who separates a free tier from a paid tier by alias, with the same underlying model, gets one tier.

## 4.8 Control plane: nine endpoints, empty system

The run writes 91 trace rows, 91 ledger rows, 124,500 billed tokens and 0.162 CNY of charge, and 4 error-log rows. In the same process run, `/billing/summary` reports 0 requests and zero tokens, `/traces` returns an empty list, `/audit` an empty list, `/audit/discrepancy` zero records, `/nodes` a null quality score for 27/27 nodes, and `/metrics` the literal string `# prometheus metrics placeholder`. Figure 4 (left) plots the gap. Table 11 adds the schema-level view: the banner promises a client/upstream ledger pair, and the database contains one ledger table; the tokenizer pool's `count_tokens` returns `None` for every model, and all 91 rows carry the hard-coded `usage_source "C"` (upstream-reported), so the advertised independent token audit does not run. 3 traces have an empty `resolved_model` and 6 have an empty `served_host`, so even the trace table cannot always say which provider answered.

This is the drift with the largest operational cost. A gateway that mis-bills is repairable if the operator can see it; here every visibility endpoint reports a system that handled no traffic, and the only accurate view is a file the operator has to open in SQLite.

## 4.9 Build configuration: headless does not compile

The gateway advertises a `gateway-server` Cargo feature for headless deployment. `cargo check -release -no-default-features -features gateway-server` exits 101 with 12 `E0433` errors: the `services` module, which owns the trace store, is gated behind `desktop-shell`, while the flex adapter and gateway routes call it unconditionally. The log is committed as `scripts/results/headless_check.log`. Every measurement in this paper therefore runs the desktop binary, and the server-only configuration — the one an operator would deploy — is unavailable.

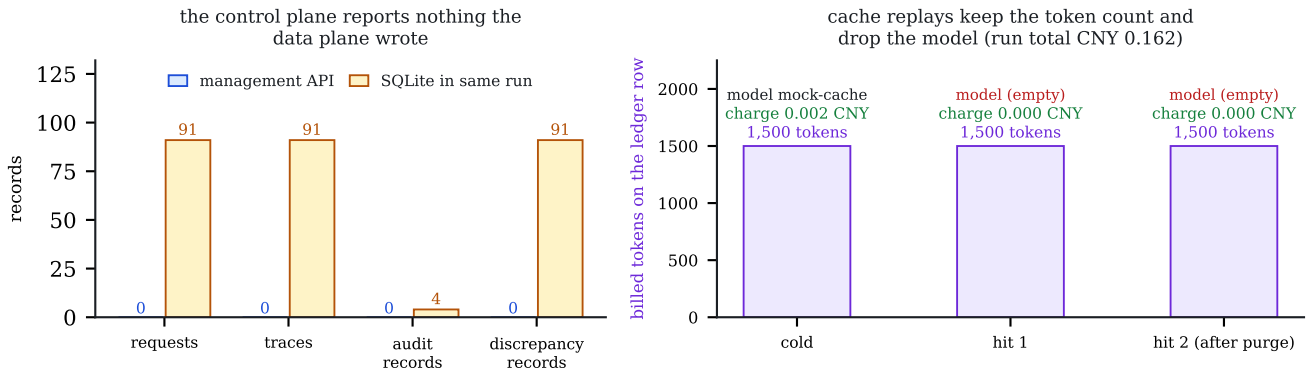



Figure 4: Left: management answers versus the gateway's own SQLite database, read in the same process run. Right: three byte-identical requests; the two cache replays keep the token count, drop the model name, and charge nothing even after the purge endpoint reported success.

Table 9: Cache and billing rows for three byte-identical requests through `lab-cache`, whose alias declares `cache_enable: false`. Upstream hits are cumulative POST counts on the single cache node; `/manage/cache/stats` reported 0 entries and `/manage/cache/purge` returned `{"purged": true}`.

| Request | Upstream hits | `is_cached` | `resolved_model` | `ledger model` | Charge (CNY) |
|---|---|---|---|---|---|
| 1 (cold) | 1 | 0 | `mock-cache` | `mock-cache` | 0.002 |
| 2 (repeat) | 1 | 1 | "empty" | "empty" | 0.000 |
| 3 (after purge) | 1 | 1 | "empty" | "empty" | 0.000 |

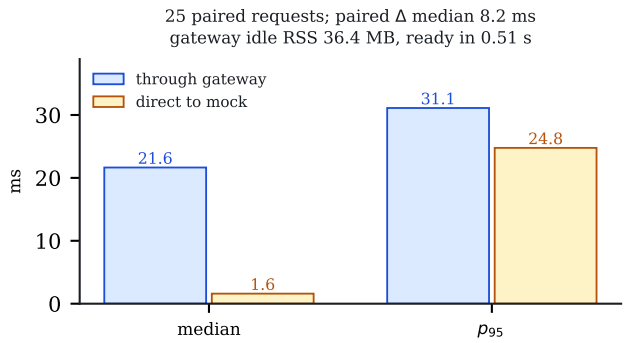



Figure 5: Loopback forwarding cost over 25 paired requests.

### 4.10 Footprint and forwarding cost

Against a mock on the same host, the gateway adds 8.21 ms at the median (21.65 ms through the gateway versus 1.60 ms direct) and 31.11 versus 24.77 ms at $p_{95}$ (Table 12, Figure 5). The absolute numbers are dominated by loopback and Python's client, not by the gateway: the same-process direct path has a 24.77 ms tail, so the tail is mostly client warm-up. The honest reading is that per-request overhead is single-digit milliseconds at the median for a non-streaming request, at the cost of a 36.43 MB resident desktop process that creates a window.

## 5 Four Kinds of Contract Drift

Reading the results back against the source gives four categories, each with a different fix.

**Envelope drift.** The payload is right, the container is wrong: 15 streaming cells answer with the wrong framing (§4.2), 16 responses drop `model` (§4.1), tool `ids` vanish on two paths (§4.3). Envelope drift is invisible to any test that only checks "did I get text?".

**Vocabulary drift.** Two vocabularies exist for the same concept and the implementation reads only one: `openai` versus `open_ai` in configuration (§4.5), `Authorization` versus `x-api-key` and `?key=` at the ingress, `1` versus `true` in a query flag, and `retry_on` keys that exist in the schema but not in the code path. Vocabulary drift is cheap to prevent — a parse that fails loudly instead of dropping unknown hints with `filter_map(|h| h.parse().ok())` would have turned the `openai` typo into a start-up error rather than a protocol change — and expensive to discover at runtime.

**Policy drift.** The configuration declares intent that the code overrides: retry switches ignored, the sub-attempt budget floored at 24 when set to 6, per-alias cache opt-out ignored, `charge_on_cache_hit` unenforceable. Policy drift is the category operators notice first, because it changes who gets billed and which provider receives traffic.

Table 10: Management plane versus the gateway's own SQLite database, read in the same process run. The ground-truth column is the exported database.

| Management endpoint | HTTP | Reported | Ground truth (same run) |
|---|---|---|---|
| `/billing/summary` | 200 | requests `0`, tokens `0/0`, charge `0.000` | 91 traces, 124,500 billed tokens, CNY 0.162 |
| `/traces` | 200 | `[]` | 91 rows in `traces` |
| `/audit` | 200 | `[]` | 4 error-log rows |
| `/audit?alarm_only=1` | 400 | Failed to deserialize query string: provided string was not 'true' or 'false' | – |
| `/audit/discrepancy` | 200 | records `0` | 91 ledger rows |
| `/nodes` | 200 | `quality_score` null for 27/27 nodes | no node-quality table in schema |
| `/metrics` | 200 | `placeholder` string | 91 traces exist |

Table 11: The advertised dual ledger and token audit versus the exported database. The start-up banner promises a client/upstream ledger pair and a tiktoken audit; the schema holds one ledger table and `TokenizerInstance` is a placeholder.

| Ledger / trace observation | Rows |
|---|---|
| Ledger rows written | 91 |
| Rows with `usage_source` = `"C"` | 91 of 91 |
| Rows with empty `model` | 3 |
| Rows with zero charge | 10 |
| Total client charge (CNY) | 0.162 |
| Upstream-cost ledger table present | no; schema holds `client_ledger`, `error_logs`, `traces` |
| Traces with empty `resolved_model` | 3 |
| Traces with empty `served_host` | 6 |
| Billed tokens, all traces | 124,500 |
| Billed tokens on cache hits | 3,000 |

Table 12: Loopback forwarding cost for single non-streaming requests: 25 alternating pairs, same client, same upstream mock.

| Measure | Through gateway | Direct to mock | Paired $\Delta$ |
|---|---|---|---|
| Requests | 25 | 25 | 25 |
| Median (ms) | 21.65 | 1.60 | 8.21 |
| $p_{95}$ (ms) | 31.11 | 24.77 | – |

**Telemetry drift.** The control plane describes a different system than the one running: 0 versus 91, 0 cache entries while serving hits, 9 constant handlers, a placeholder `/metrics`. Telemetry drift is the most corrosive, because it removes the means of detecting the other three.

The four share a root cause. The contracts live in a banner, a README and a YAML schema — all prose — while the only executable specification is 14 unit tests, four of which assert the sniffer's scoring and one of which asserts exactly the behaviour we found surprising: a body with `model`, `max_tokens` and `messages` is classified as Anthropic. That test is correct as written and wrong as a specification: it encodes a scoring rule that misclassifies 16 of our 20 non-streaming shapes when they arrive on a path that has no explicit protocol.

# 6 Related Work

**LLM gateways and routers.** LiteLLM (BerriAI, 2024) and One-API (songquanpeng, 2024) are the widely deployed multi-provider proxies, with larger provider matrices and, in LiteLLM's case, a router with explicit retry and cooldown semantics. RouteLLM (Ong et al., 2024) treats routing as a learned decision between a strong and a weak model; vLLM (Kwon et al., 2023) and its OpenAI-compatible server (vLLM Project, 2024) define the self-hosted end of the same contract. PLocalSwitch is a desktop-scale point in this space: single user, single host, local keys. None of these systems publishes a conformance matrix over inbound protocol $\times$ upstream protocol, which is why we measured one.

**Silent failure.** Gray failure — partial failure that keeps answering requests — is named by Huang et al. (Huang et al., 2017). Our truncated-stream and wrong-envelope results are gray failures in that sense: HTTP 200, plausible body, wrong contract. The empty error surface (`/audit` returning `[]`) removes the operator's only detector.

**Contract testing and error hygiene.** Consumer-driven contract testing (Pact Foundation, 2024) is the established answer to envelope drift: the client's expected shape becomes an executable check, which is exactly what Section §4.2 shows missing. Semantic versioning (Preston-Werner, 2013) prescribes that behaviour changes be declared; a banner claim of "100% compatible" is a version statement with no test behind it. The

gateway's error masking — upstream bodies are truncated and replaced by a generic client message — is deliberate and correct against CWE-209 (MITRE, 2024); our criticism is only that the mask also hides the distinction between "your request was bad" and "the provider rejected my configuration", which is what HTTP 400 on an upstream 404 does.

**Robustness principle.** Postel's law (Braden, 1989) — be strict in what you send — is the design instinct behind the pass-through adapters and the lenient body sniffer. The measurements here are a case study in the other half of the sentence: tolerance without a declared boundary is how a body with `max_tokens` becomes an Anthropic request.

# 7 Limitations

Our fixtures are documentation-faithful, not provider-observed; a provider that returns `404 page not found` for a valid OpenAI path would trigger the same HTTP 400 we measured, but we cannot say how often real providers do. We measured one version at one commit; 22 explicit TODO markers make it likely that some findings are known to the author and in flight — the cache back-end selection carries a `v0.2.26` comment fixing exactly such a mismatch, which is evidence both that drift is fixable and that our snapshot is a moving target. We did not test the Redis cache path, image or audio payloads, concurrency limits, the Tauri front end, or multi-tenant key isolation, and we did not attempt to exploit the management endpoints — the local management token is checked but we did not evaluate its strength. Latency figures come from a Python client on the same host and bound the *difference*, not the gateway's absolute cost. Finally, all findings are behavioural: we report what the binary does and cite the source lines that explain it, without claiming intent.

# 8 Conclusion

PLocalSwitch translates correctly and advertises more than it implements. The fix for the first half of that sentence is already in the code — `denormalize_response` exists and is simply not called on the streaming branch. The fix for the second half is not a code change but a test change: make the banner a test file. A $4 \times 5$ grid of 20 conformance assertions (expected content type, expected `model`, expected tool-call `id`, expected event sequence) plus three policy assertions (a disabled retry stays disabled, a configured budget is a ceiling, a purged cache is empty) would have caught every finding in Sections 4.2–4.8 in the developer's own run, at a cost far below the 91 rows of evidence this harness needed to produce them. Until then, the practical advice for an operator of this build is narrow and concrete: trust the text, verify the envelope, do not rely on the management API for anything, and treat `protocol_hints` as a string that must match `open_ai` exactly.

**Availability.** The harness, the raw JSON artifact, the sanitized lab configuration, the headless-build log, and the scripts that emit every figure and table are versioned with this paper. Re-running `measure.py` against the same commit reproduces the grids in Section 4; re-running `make_tables.py` and `make_figures.py` regenerates Sections 4.1–4.10 verbatim from that artifact, so a future release can be diffed against these numbers by re-running four commands rather than by re-reading prose.

# References


Amazon Web Services. Converse (amazon bedrock runtime API). https://docs.aws.amazon.com/bedrock/latest/APIReference/API_runtime_Converse.html, 2024.

Anthropic. Messages API. https://docs.anthropic.com/en/api/messages, 2024.

BerriAI. LiteLLM: Unified LLM api for 100+ LLMs on Lite and Enterprise. https://github.com/BerriAI/litellm, 2024.

R. Braden. Requirements for Internet hosts – communication layers. Technical report, IETF, Oct. 1989. URL https://www.rfc-editor.org/rfc/rfc1122. Robustness principle, Section 1.2.2.

R. Fielding, M. Nottingham, and J. Reschke. HTTP semantics. Technical report, IETF, June 2022. URL https://www.rfc-editor.org/rfc/rfc9110.

Google. Gemini API: Generate content. https://ai.google.dev/api/generate-content, 2024.

P. Huang, C. Guo, L. Zhou, et al. Gray failure: The Achilles' heel of cloud-scale systems. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems (HotOS '17)*. ACM, 2017. doi: 10.1145/3102980.3103005. URL https://www.usenix.org/conference/hotos17/presentation/huang-peng.

W. Kwon, Z. Li, S. Zhuang, et al. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP '23)*. ACM, 2023. doi: 10.1145/3600006.3613165. URL https://arxiv.org/abs/2309.06180.

S. McArthur. reqwest: Http client for rust. https://github.com/seanmonstar/reqwest, 2024.

MITRE. CWE-209: Generation of error message containing sensitive information. https://cwe.mitre.org/data/definitions/209.html, 2024.

moka-rs. mini-moka: Small concurrent cache for rust. https://github.com/moka-rs/mini-moka, 2024.

M. Nottingham and R. Fielding. HTTP/1.1. Technical report, IETF, June 2022. URL https://www.rfc-editor.org/rfc/rfc9112.

Ollama. Ollama API reference. https://docs.ollama.com/api/introduction, 2024.

I. Ong, A. Almahairi, V. Wu, et al. RouteLLM: Learning to route LLMs with preference data. arXiv:2406.18665, 2024. URL https://arxiv.org/abs/2406.18665.

OpenAI. Chat Completions API reference. https://platform.openai.com/docs/api-reference/chat, 2024a.

OpenAI. Responses API reference. https://platform.openai.com/docs/api-reference/responses, 2024b.

Pact Foundation. Consumer-driven contract testing for microservices. https://docs.pact.io/, 2024.

T. Preston-Werner. Semantic versioning 2.0.0. https://semver.org/, 2013.

songquanpeng. One API: Unified LLM api and Dashboard. https://github.com/songquanpeng/one-api, 2024.

SQLite Development Team. SQLite documentation. https://www.sqlite.org/docs.html, 2024.

Tauri Programme. Tauri 2.x: Build smaller, safer desktop applications. https://tauri.app/, 2024.

tokio-rs. axum: Web framework that rust saw comes true. https://github.com/tokio-rs/axum, 2024a.

tokio-rs. Tokio: An asynchronous Rust runtime for writing network applications. https://github.com/tokio-rs/tokio, 2024b.

vLLM Project. OpenAI compatible server. https://docs.vllm.ai/en/latest/serving/openai_compatible_server.html, 2024.

WHATWG. Server-sent events. WHATWG Living Standard, 2024. URL https://html.spec.whatwg.org/multipage/server-sent-events.html.

yxpil. PLocalSwitch: Local LLM api proxy gateway (version 0.2.27). https://github.com/yxpil/PLocalSwitch, 2026.