KUBECON + CLOUDNATIVECON NORTH AMERICA 2026

Seeing Through the Stack

End-to-End Observability for llm-d with OpenTelemetry

Metrics · Traces · Logs, correlated across the llm-d Router, KV-cache indexer, P/D sidecar and vLLM



Guangya Liu
SIG-Observability Lead, llm-d (CNCF Sandbox)
Executive Director, JPMC

Nicole Xin
Gateway API Inference Extension Maintainer
llm-d Contributor
Senior Software Engineer, Google

A FAMILIAR STORY

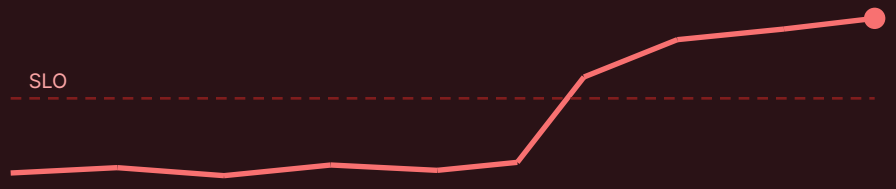
P99 TTFT tripled. Every dashboard is green.

ALERT FIRING

P99 TTFT (Time to First Token)

3x

~0.2 s → ~0.6 s



- GPU utilization — **normal**
- Pods Ready — **2/2**
- Error rate — **flat**
- vLLM queue — **normal**

Gateway, Router (EPP), KV-cache indexer or vLLM: who is to blame?

Gateway · EPP · KV-cache indexer · 2 vLLM replicas on 1 GPU | 44 chat sessions, ~3.8k-token prompts each

WHERE WE ARE GOING

Agenda: from blind spots to one-click answers

01

Background

Architecture walk

Three questions

02

Signals in llm-d

OTel + Collector

How llm-d uses it

Semconv

Metrics

03

End-to-end tracing

The span tree

A healthy trace

Context propagation

04

Correlation

The triangle

Exemplars

Logs with trace_id

Demo

05

What's next

Gaps by component

How to help

Running example: the P99 TTFT incident from slide 2, solved in the part 04 demo

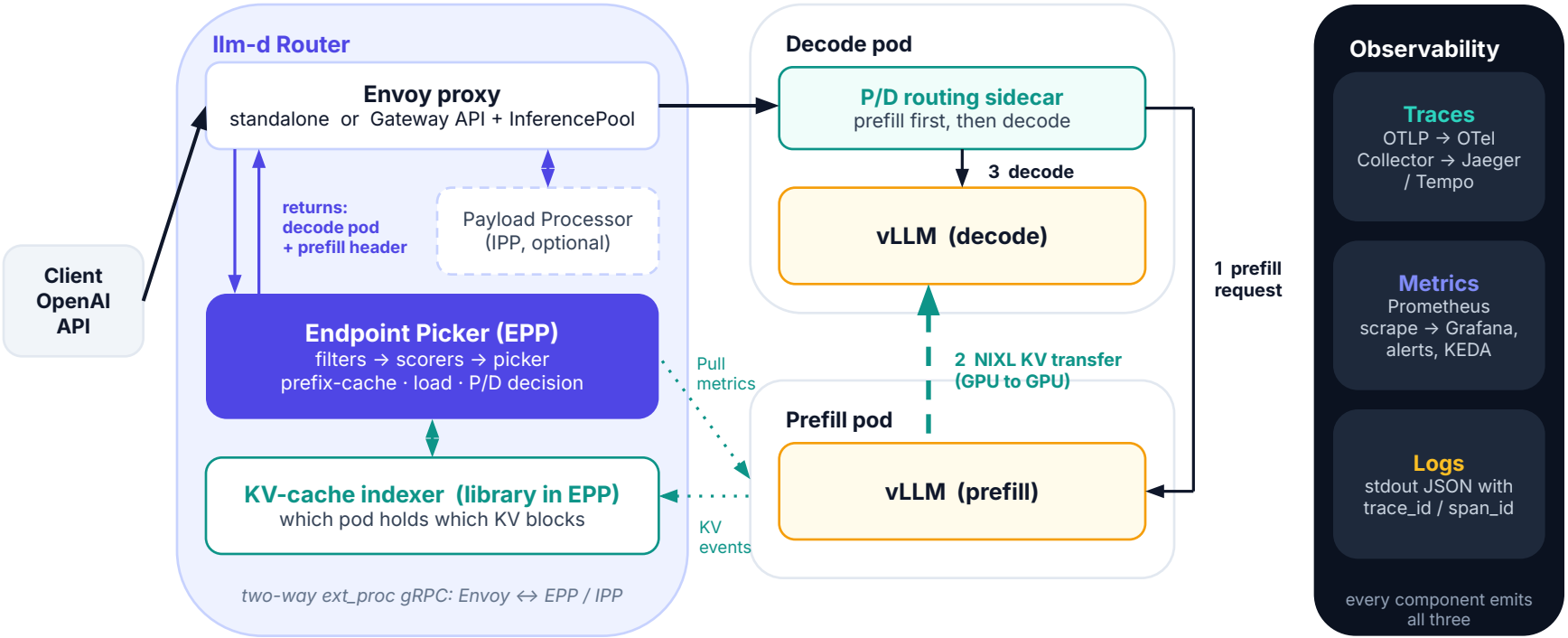
01

Background

The llm-d architecture and three questions behind every slow request

THE SYSTEM WE NEED TO OBSERVE

llm-d architecture: one request, many components



Solid arrows: request path Dashed / dotted: KV data path (NIXL, KV events) Based on llm-d v1.0

BACKGROUND · THREE QUESTIONS

Three questions behind every slow request

1 Token-level latency

- TTFT + ITL / TPOT, not one duration
- Long answers inflate total time
- Queue vs prefill vs decode time

Which phase got slow?

2 Cache locality

- Per-request prefix-cache hit / miss
- Which pod holds which KV blocks
- Hit from GPU, CPU RAM or disk?

Was the cache warm where we sent it?

3 Routing decisions

- Why did EPP pick pod X?
- Filters, scorer weights and scores
- P/D: disaggregate or not, and why

Was the decision right?

Why these questions matter: the win erodes silently

pod restarts wipe cache · dropped KV events · wrong scorer weights · prompt not cached yet

No errors. Just a lower hit rate and a higher TTFT, exactly the incident on slide 2.

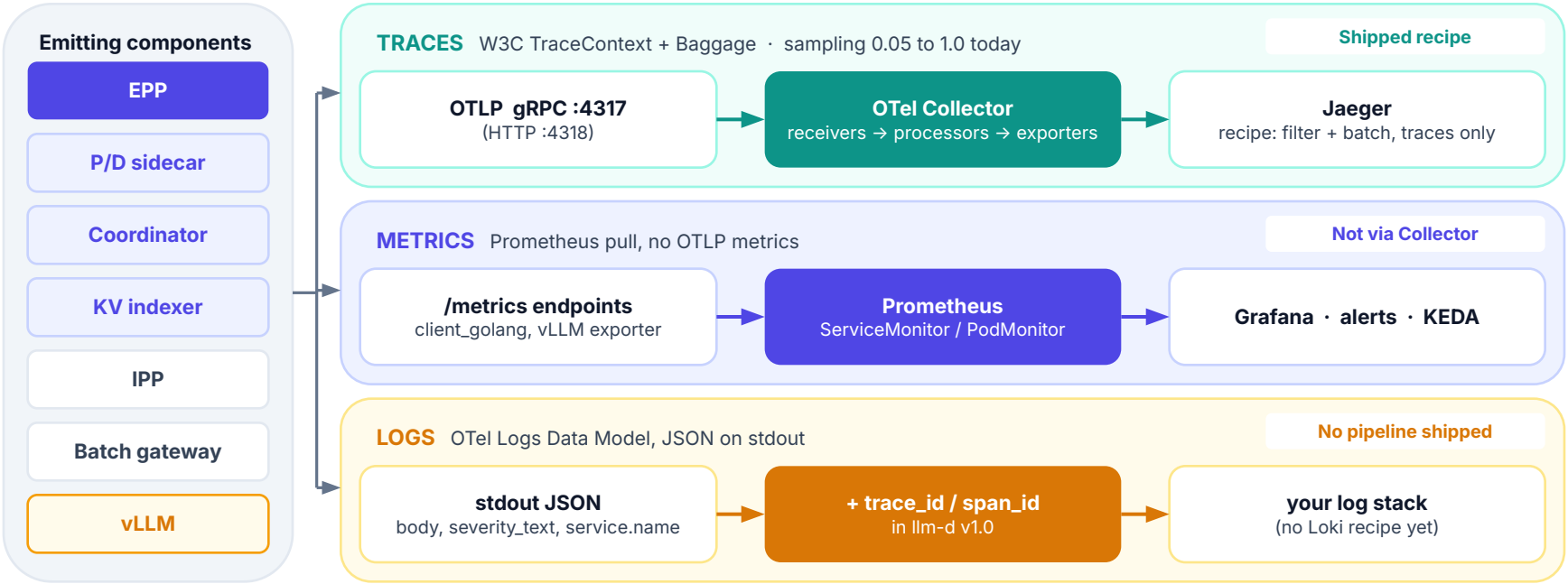
02

Signals in llm-d

Traces, metrics, logs, with OpenTelemetry at the core: OTLP traces, semantic conventions, OTel log format

SIGNALS IN LLM-D

How llm-d uses OpenTelemetry today



OTel-native for traces, semantic conventions and log format; **Prometheus** for metrics; **the Collector is the convergence point.**

SIGNALS IN LLM-D

Semantic conventions: spans that speak llm-d

Namespace	Example attributes	What it answers
`gen_ai.*` (EPP)	`gen_ai.request.model, gen_ai.request.id`	Which model, which request
`gen_ai.*` (vLLM)	`gen_ai.latency.time_to_first_token` `gen_ai.latency.time_in_queue, gen_ai.usage.*`	Engine time breakdown; tokens
`llm_d.epp.scorer.*`	`{name, weight, score.max}`	Why this pod won
`llm_d.epp.picker.*`	`top_endpoints, top_scores`	Who the the top ranked candidate pods
`llm_d.epp.pd.*`	`disaggregation_used, reason`	Split prefill / decode? Why?
`llm_d.kv_cache.*`	`prefix_match.longest_chain`	How much prefix was cached
`llm_d.pd_proxy.*`	`true_ttft_ms, prefill_duration_ms`	Time until decode starts (P/D)

Yellow rows: the attributes that solve the demo (slides 13, 19 and 20)

SIGNALS IN LLM-D

Metrics: golden signals per component

Component	Golden signals (Prometheus metric names)	Answers

EPP	`llm_d_epp_request_duration_seconds`
	`llm_d_epp_request_error_total`
	`llm_d_epp_ready_endpoints`
		Is routing slow, failing, or starved?

KV indexer	`llm_d_epp_kv_cache_index_*` (lookup hits, latency) `llm_d_epp_kv_cache_events_*` (pool queue depth)	Is the prefix index useful and fresh?
P/D sidecar	`llm_d_disagg_sidecar_prefill_duration_seconds` `llm_d_disagg_sidecar_decode_duration_seconds`	Which leg is slow?

vLLM	`vllm:time_to_first_token_seconds`
	`vllm:inter_token_latency_seconds`
	`vllm:num_requests_waiting`, `vllm:kv_cache_usage_perc`
	`vllm:prefix_cache_hits` / `vllm:prefix_cache_queries`
		User latency, queueing, KV pressure, prefix hit rate

NIXL KV transfer	`vllm:nixl_xfer_time_seconds`
	`vllm:nixl_bytes_transferred`
	`vllm:nixl_num_failed_transfers`
		Is P/D KV transfer slow or failing? (fleet-level)

Tiered offload	`vllm:kv_offload_tiering_*` (per-tier bytes, time)	Are offload tiers paying off?

Lesson: metric names are an API. Renaming one breaks every dashboard, alert and KEDA trigger that uses it.

Ships as recipes

Shipped

11 Grafana dashboards (llm-d-*)

```
router · tpu-overview
vllm-overview · sglang-overview
pd-coordinator-metrics
performance-kv-cache
failure-saturation-dashboard
diagnostic-drilldown-dashboard
batch-gateway-apiserver
batch-gateway-processor
batch-gateway-gc
```

7 PrometheusRule alerts (EPP)

```
EPPHighErrorRatio
EPPCriticalErrorRatio
EPPNoReadyEndpoints
EPPMetricsAbsent
EPPDataLayerPollErrors
EPPDataLayerExtractErrors
EPPExtProcStreamErrors
```

Install scripts

```
install-prometheus-grafana.sh
load-llm-d-dashboards.sh
```

03

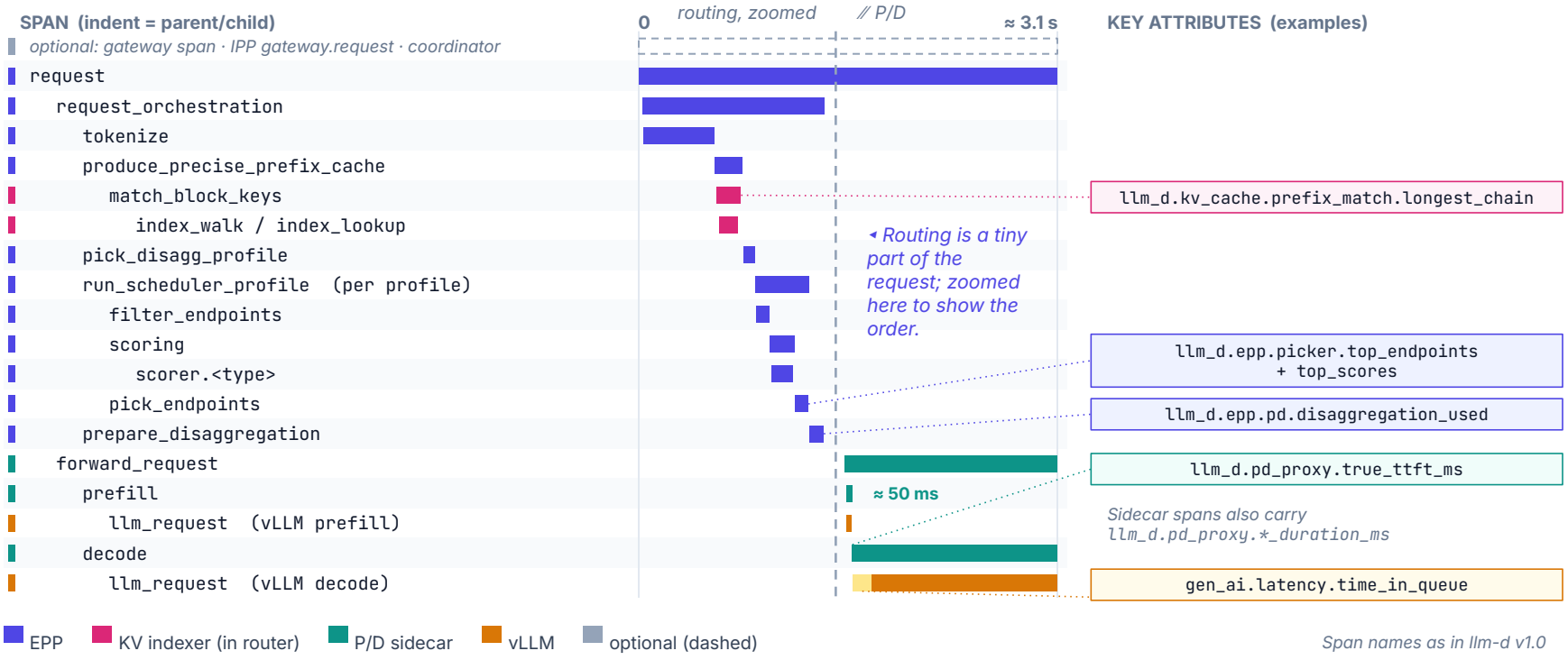
End-to-end tracing

One trace across Router, KV indexer, P/D sidecar and two vLLM engines

END-TO-END TRACING

Anatomy of a P/D request trace

Simplified example, not a real capture: how to read a P/D trace. Bar lengths are illustrative; the demo cluster has no P/D.



https://github.com/gyliu513/langX101/tree/main/llm-d/llm-d-spark-full-demo/kubecon-obs-demo

What a healthy request looks like

Data: demo trace ee729b07 · llm-d v1.0 · EPP → vLLM (2 replicas) · healthy config, prefix weight 3 · viewed in Jaeger

ONE REQUEST, ONE TRACE — `span count · vLLM gen_ai.usage.*`

17 spans

One chat completion from the demo: 3,839 prompt tokens, 24 completion tokens, 2 services (EPP and vLLM). Routing and the engine sit in one trace.

ROUTING COST — `request_orchestration / request`

14.4 ms of 665 ms

Entire EPP orchestration. **Tokenizing 3,839 tokens is 13.9 ms of it.** Prefix match, filter, score and pick add about 0.4 ms.

CACHE-AWARE PICK — `match_block_keys · pick_endpoints`

		cached pod		other pod
prefix cache	×3	1	→ 3	0 → 0
cache space	×2	0.965	→ 1.93	0.932 → 1.86
queue empty	×2	1	→ 2	1 → 2
total		**6.93**		**3.86**

Cache space + queue empty = how busy each pod is (both nearly idle). Score (0 to 1) × weight, summed: the 3 prefix points decide.

ENGINE TIME — `vLLM llm_request`

52 ms prefill, TTFT 66 ms

`time_in_model_prefill` is 52 ms for 3,839 tokens, because this pod already holds the prefix in its KV cache. `time_in_queue` is 0.02 ms.

This is the baseline: the cached pod wins, prefill 52 ms. On slides 18 and 19, the same kind of request takes 569 ms.

Overhead test: turning on llm-d observability had no measurable impact on llm-d performance.
Blog PR: https://github.com/llm-d/llm-d.github.io/pull/543

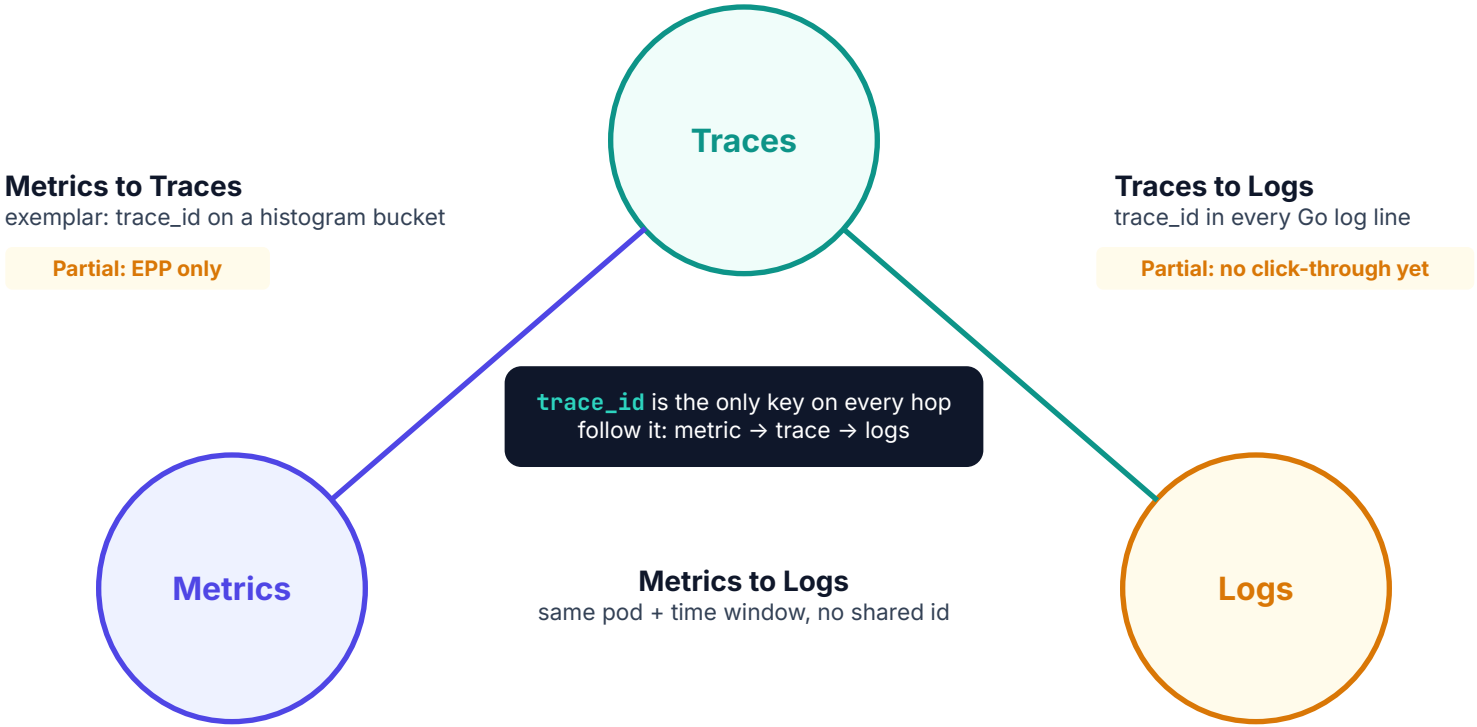
04

Correlation: metrics, traces, logs

Exemplars, trace-aware logs and the request id

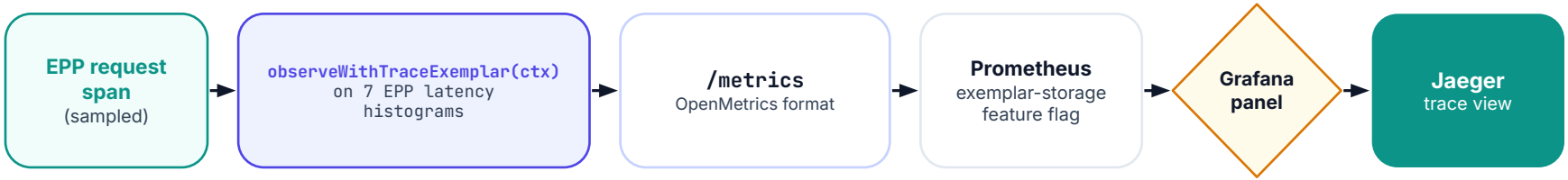
CORRELATION

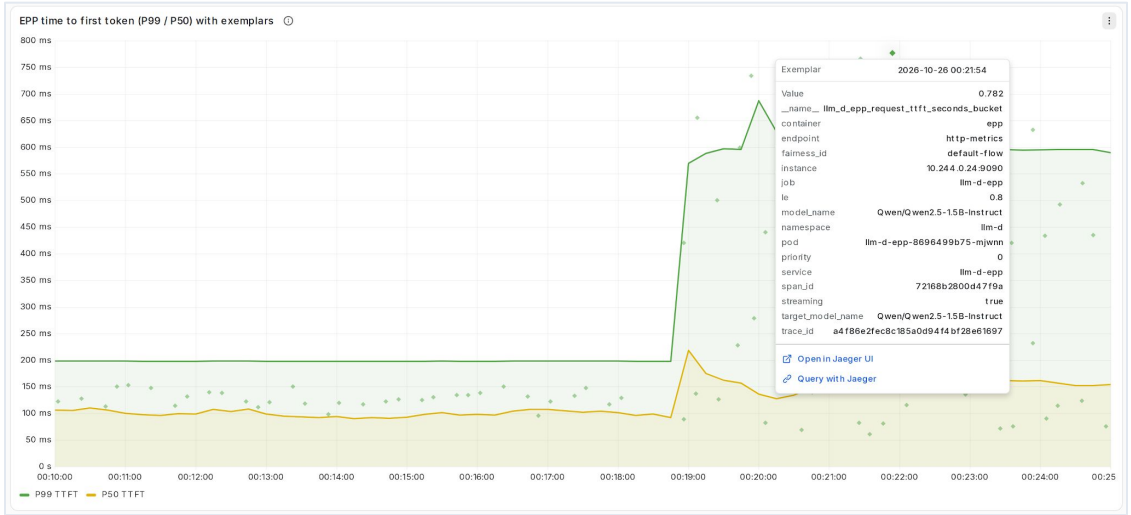
The correlation triangle



CORRELATION · METRICS TO TRACES

Metrics to traces: exemplars





Own Prometheus / Grafana?

- Prometheus: enable exemplar-storage
- Grafana datasource: exemplarTraceIdDestinations → Jaeger URL
- llm-d's install-prometheus-grafana.sh does both

CORRELATION · LOGS TO TRACES

Logs and traces

Works today

trace_id / span_id on EPP request-path log lines
OTEL Logs Data Model

```
illustrative (fake ids)
{"timestamp": "2026-11-10T15:04:05.123Z",
 "severity_text": "INFO", "severity_number": 9,
 "body": "request scheduled",
 "service.name": "llm-d-epp",
 "trace_id": "4bf92f3577b34da6a3ce929d0e0e4736",
 "span_id": "00f067aa0ba902b7"}
```

Tip: the message is in body, not msg

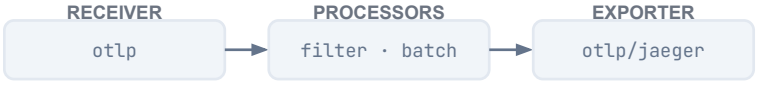
What you can't do yet

- Click from a trace to its logs: no log pipeline ships, so you grep pod logs by hand
- Find vLLM logs from a trace: vLLM logs no trace_id
- Find P/D transfer logs from a trace: sidecar connector logs carry no trace_id

How to close the gap (not shipped)

OTel Collectors: one binary, two roles

gateway (Deployment) · traces pipeline, shipped today



+ agent (DaemonSet) · logs pipeline, add this · straight to Loki



filelog operators parse JSON: container, json_parser, trace_parser
root only, no caps, read-only /var/log/pods
Loki 3.x native /otlp · OpenShift: ClusterLogForwarder

Grafana



Logs skip the gateway: agent → Loki
traces path unchanged

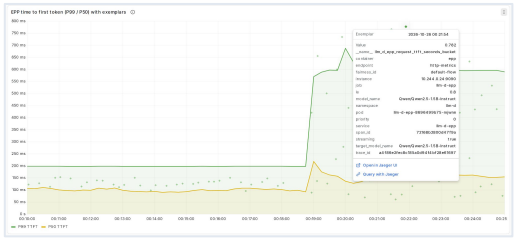
DEMO

Demo: from alert to trace

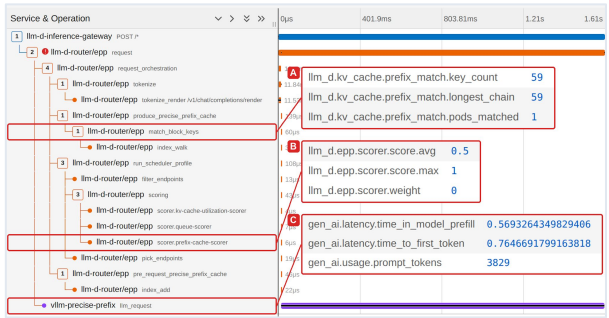
1 Alert fires (Prometheus) P99 TTFT > 250 ms for 2 m



2 Grafana P99 panel same as slide 17



3 Jaeger trace



A longest_chain 59, pods_matched 1: the whole prefix is cached, on one pod
B weight 0: the cached pod loses its 3 points, the other pod wins 3.93 vs 3.86
C vLLM prefill 569 ms for 3,829 tokens

		cached pod	other pod (picked)
prefix cache	×0	1 → 0	0 → 0
cache space	×2	0.931 → 1.86	0.966 → 1.93
queue empty	×2	1 → 2	1 → 2
total		**3.86**	**3.93**

Setup: llm-d v1.0, 2 vLLM replicas on one GB10. Exemplars ship in llm-d v1.0; the TTFT panel and vLLM tracing are added by hand.

DEMO

Demo: from trace to root cause

4 Logs by trace_id

EPP logs for one trace_id

```
$ kubectl -n llm-d logs deploy/llm-d-epp \
  | jq -c 'select(.trace_id ==
      "a4f86e2fec8c185a0d94f4bf28e61697")
      | {body, trace_id, svc: ."service.name"}'
{"body":"EPP received request",
"trace_id":"a4f86e2fec8c185a0d94f4bf28e61697",
"svc":"llm-d-router/epp"}
{"body":"EPP sent request body response(s) to proxy",
"trace_id":"a4f86e2fec8c185a0d94f4bf28e61697",
"svc":"llm-d-router/epp"}
{"body":"EPP sent response body back to proxy",
"trace_id":"a4f86e2fec8c185a0d94f4bf28e61697",
"svc":"llm-d-router/epp"}
```

No errors for this request, so it is a config problem, not a crash.
EPP lines only: vLLM and P/D sidecar logs carry no trace_id yet

5 Root cause

Root cause: the EPP config change

```
$ helm history llm-d -n llm-d | tail -2 | cut -f1-3
3               Mon Oct 26 00:06:22 2026        superseded
4               Mon Oct 26 00:18:07 2026        deployed

$ diff -u helm-values/epp-config-{before,after}.values.yaml
@@ -74,7 +74,7 @@
               - pluginRef: queue-scorer
                 weight: 2.0
               - pluginRef: prefix-cache-scorer
-                weight: 3.0
+                weight: 0.0

$ ./scripts/compare-traces.py traces/before-ee729b07.json \
      traces/after-a4f86e2f.json
                                  prefix w=3.0    prefix w=0.0
trace_id                             ee729b07        a4f86e2f
longest_chain                              59              59
pods_matched                                1               1
prefix scorer weight                        3               0
prefix scorer score.max                     1               1
picker top_scores               [6.93, 3.86]    [3.93, 3.86]
vLLM prefill                           52 ms          569 ms
vLLM TTFT                              66 ms          765 ms
```

The weight is EPP config, so check its history: helm history
rev 4 at 00:18:07, P99 up at 00:18:45: weight 3 → 0
prefill 52 → 569 ms, and setting it back to 3 fixes it
= the slide 2 TTFT spike, explained

Cold cache is a symptom. Other causes to rule out: the cache pod restarted, competing traffic evicted the KV cache, KV events were lost (stale index).

05

What's next

What's hard and how to help

WHAT'S NEXT

What's hard, and how to help

Today: traces in 7 of 9 components · logs → trace in 6 · metric → trace in EPP and coordinator

Component	Metrics	Traces	Log → trace	Dashboards / alerts	Metric → trace
llm-d Router / EPP	Yes	Yes	Yes	Yes + 7 alerts	7 hist. + 1 counter
KV-cache indexer	Yes	Yes	Yes	Partial	Gap
P/D sidecar	Yes	Yes	Yes	Gap	Gap
Coordinator (exp.)	Yes	Yes	Yes	4 alerts, no dashboard	1 histogram
vLLM	Yes	Yes (config)	Gap	Yes	Gap
KV offload (vLLM)	Yes	Gap	Gap	PromQL only	Gap
Payload Processor	Partial	Yes	Yes	Gap	Gap
Batch gateway	Yes	Yes	Yes	Yes + alerts	Gap
Autoscaling (KEDA)	Yes	Gap	Gap	Partial	Gap

Yes | Partial | Gap

Checked against llm-d main, Nov 2026. Not all of it is in a release yet.

Thank you

Join SIG-Observability

WEB

`llm-d.ai`

SLACK

#sig-observability (llm-d Slack)

CODE

`github.com/llm-d/llm-d`

START HERE

label: good first issue



Scan for SIG-Observability