DevFest Vienna 2026 · TU Wien · 7 November 2026

Behavioral drift

The silent bugs in LLM workflows

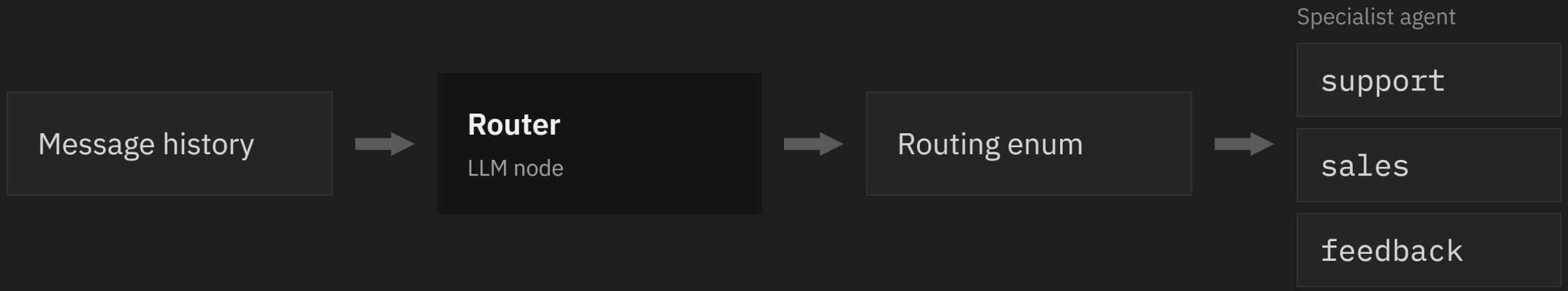
Andrey Minogin

Software Architect

Reliability and Performance of LLM, JVM and Postgres systems

minogin.com

The router



The router prompt

```
Respond only with the exact query type. No explanation. No formatting.
```

One line lost

Expected

`feedback`



Got

`**feedback**`

Lost in a prompt edit that was about something else.

No validation → the chat broke. Not every time — in one branch of the workflow.

Lucky, in hindsight

No validation · what happened

Broke in one branch.

With validation

Same place, better error message.

“Safe” fallback

Keeps working. Every request misrouted.

Unknown value → general support. Only an admin reading the full technical history would see it.

The silent version had already happened here.

Prompt and code disagreed; that fallback hid it. Removed a few days earlier.

Behavioral drift

The implicit contract between LLM and code breaks silently.

1. Looks correct to humans
2. Passes most schema checks
3. No error is thrown

Two sources

Yours

Prompt edits, workflow restructuring, swapping the LLM.

Theirs

The LLM changes under you — even with a pinned id.

arXiv:2606.29719 · Liu Zewen, June 2026

`gpt-4o-2024-08-06 (pinned)`

Same protocol. Effect present in May — zero in 8 of 8 June runs.

Authors: a silent update; one case, not a law — “shelf life measured in weeks”.

Why not validation?

It asks

Is the node output *allowed*?

It misses

- A change inside the JSON
- Routing 40/30/30 → 20/20/60

Still needed everywhere — even a leaf node needs “is this valid HTML”.

Why not unit tests?

Really component tests: machinery real, LLM mocked by a state machine

It asks

Does the workflow behave — routing, parsing, node connections?

It misses

- What the real LLM outputs — they never call it
- So my incident would pass them

Still needed — with a fake that returns the ugly variants: markdown around the enum, an out-of-set value, an empty string. Drift then fails loud and local.

Why not evals?

By evals I mean judge-based quality evaluation: inputs → an evaluator → a score.

The consumer of an internal node is code.

It asks

Is this good *for a human*?

It misses

- **feedback** scores high — and breaks the workflow
- Routing 40/30/30 → 20/20/60: every answer fine, the average doesn't move

Still needed at the leaf, where a human reads the answer.

The missing layer

Where the bug sits: internal edges at runtime — LLM output read by code or another LLM, not by a human. Nothing looks there.



Check	Asks
Validation	Is the output allowed?
Component tests	Does the machinery work?
Evals	Is it good for a human?
?	Did the behavior change — and did you mean it to?

The fix: a behavioral profile

Did the behavior change — and did you mean it to?

It doesn’t forbid change. It shows every difference; you mark the ones you meant.

A diff for behavior

1

Profile

Per node: output form and routing.

2

Diff

Two versions, node by node.

3

Rank

By severity. A human decides what was meant.

No LLM judge. No labelled dataset. No ground truth.

Only a record of what was normal — recorded on every run, diffed automatically.

Before rollout: a CI gate

Prompt change → Fixed scenarios on the real LLM → Diff against the baseline → HIGH blocks until a human confirms

Running it is the only check — behavior isn't derivable from the prompt text.

The scenario set is your definition of normal. Cheap to extend — no labelling.

In production: continuous

The profile

A few attributes on spans you already emit.

The check

A rolling window against the baseline.

Full coverage — and it catches drift with no deploy: the provider’s.

The hard part: the threshold

Needs no threshold

Form changes, vanished branches.

Needs a threshold

Moving shares.

Too tight → nobody reads the alerts.

Too loose → the quiet failure stays quiet.

25%

Mine — so you can disagree

of traffic moved to other targets.

The tool

Small Kotlin library · a checkpoint per node, nextStep where it branches

```
detector
  .checkpoint(
    step = "classify-query",
    input = message,
    output = answer
  )
  .nextStep("feedback")
```

Demo · no API key, no network, scripted LLM

```
git clone https://github.com/minogin/ai-anomaly-detector.git
./gradlew -q demo
```

JitPack

```
implementation("com.github.minogin:ai-anomaly-detector:0.3.1")
```

The profile: what normal looked like

```
PROFILE (1.0)

  classify-query: STRING -> handle-feedback, handle-sales, handle-support
  handle-feedback: STRING -> summarize
  handle-sales: JSON_OBJECT {reply=STRING, discount=INTEGER} -> summarize
  handle-support: STRING -> summarize
  summarize: STRING
```

Per node: the form of its output and where it routes next.

Demo: 1.0 → 1.1, the incident

```
Anomaly Detector: 1.0 -> 1.1
================================================
FINDINGS

[HIGH] Output form changed at 'classify-query'
  Reference: STRING  e.g. support
  Current:   STRING  e.g. support
             MARKDOWN  e.g. **feedback**
[HIGH] Inconsistent output forms at 'classify-query'
  Forms: STRING  e.g. support
         MARKDOWN  e.g. **feedback**
[MID] Routing distribution changed at 'classify-query'
  Reference: handle-feedback 40%, handle-sales 30%, handle-support 30% (10 samples)
  Current:   handle-feedback 20%, handle-sales 20%, handle-support 60% (10 samples)
  Shift:     30% of routings changed target (threshold 25%)
```

One bug, three findings — each one step further downstream.

Demo: 1.0 → 1.2, a change inside the JSON

```
Anomaly Detector: 1.0 -> 1.2
================================================
FINDINGS

[HIGH] JSON schema changed at 'handle-sales'
  Reference: JSON_OBJECT {reply=STRING, discount=INTEGER}
             e.g. {"reply": "Happy to help with pricing.", "discount": 10}
  Current:   JSON_OBJECT {reply=STRING, offer={discount=INTEGER}}
             e.g. {"reply": "Happy to help with pricing.", "offer": {"disco...
  Added:     offer: {discount=INTEGER}
  Removed:   discount: INTEGER
```

JSON schema changed at handle-sales: discount moved into offer.

Demo: 1.0 → 1.3, a branch disappears

```
Anomaly Detector: 1.0 -> 1.3
================================================
FINDINGS

[MID] Transitions changed at 'classify-query'
  Removed:   handle-feedback
  Reference: handle-feedback 40%, handle-sales 30%, handle-support 30% (10 samples)
  Current:   handle-sales 30%, handle-support 70% (10 samples)
[MID] Step missing: 'handle-feedback'
  Reference forms: STRING  e.g. Thank you for the feedback, it has been passed on to the ...
  Reference transitions: summarize
```

The other kind: a branch that stopped being taken at all.

Limits

Limit	What happens
Meaning	Seen only where it turns into control flow. Meaning that ends up in text or data is what evals are for
Traffic	Low-volume node: no distribution, form and schema only
Baseline	Only as good as the last review that blessed it. Record a broken version and it gets protected; update it blindly and slow drift walks away

What to do tomorrow

1. Prompts in git, full commit history — their own release cycle, through the CI gate, no redeploy.
2. Fail loud, not safe.
3. Component tests with a scripted LLM, ugly variants included.
4. **Diff behavior before rollout** — the cheapest thing in this talk. Put it in first.
5. Then diff in production over a rolling window.

Non-deterministic software is still software.

Ask the cheap question first: did it change in a way nobody chose?

Record what normal looked like.

Try it on your workflow — and tell me where it falls short.

The pattern matters more than the tool: build the same in Python if that’s your stack.

`github.com/minogin/ai-anomaly-detector`

Andrey Minogin

Reliability and Performance of LLM, JVM and Postgres systems

---

minogin.com

andrey@minogin.com

github.com/minogin/ai-anomaly-detector

How I work with teams

Two sessions on your system

When something is clearly wrong and nobody can say where to start.

Architecture review · area audit · architecture development

When it needs a deeper look, or something new has to be built.

Training for your team

PostgreSQL under load, Kotlin coroutines, hexagonal architecture, reliable LLM workflows.