

Leveraging SRE and Observability Techniques

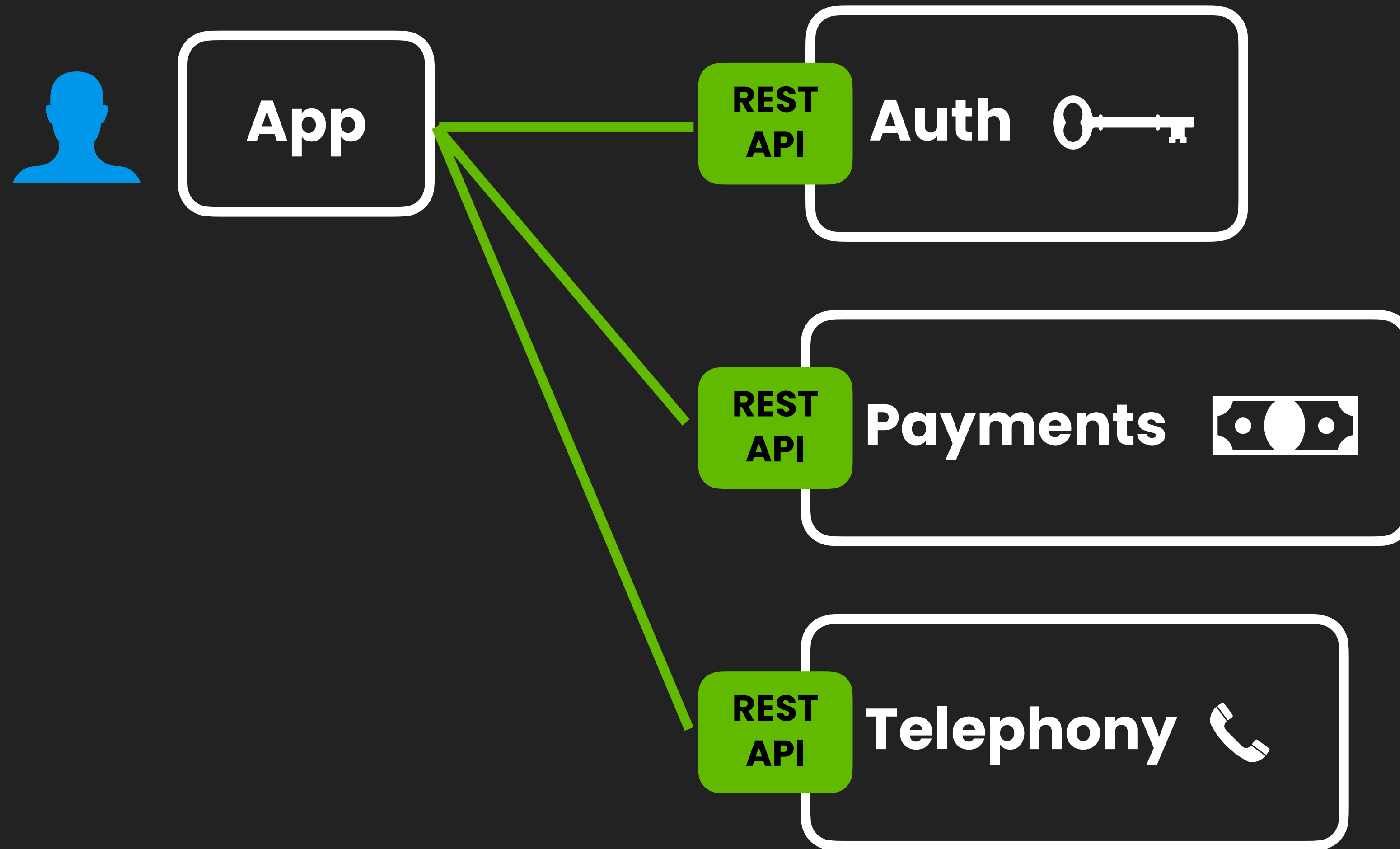
for the Wild World of Building on LLMs

@cyen
@honeycombio





LLMs ≈ like APIs we know and

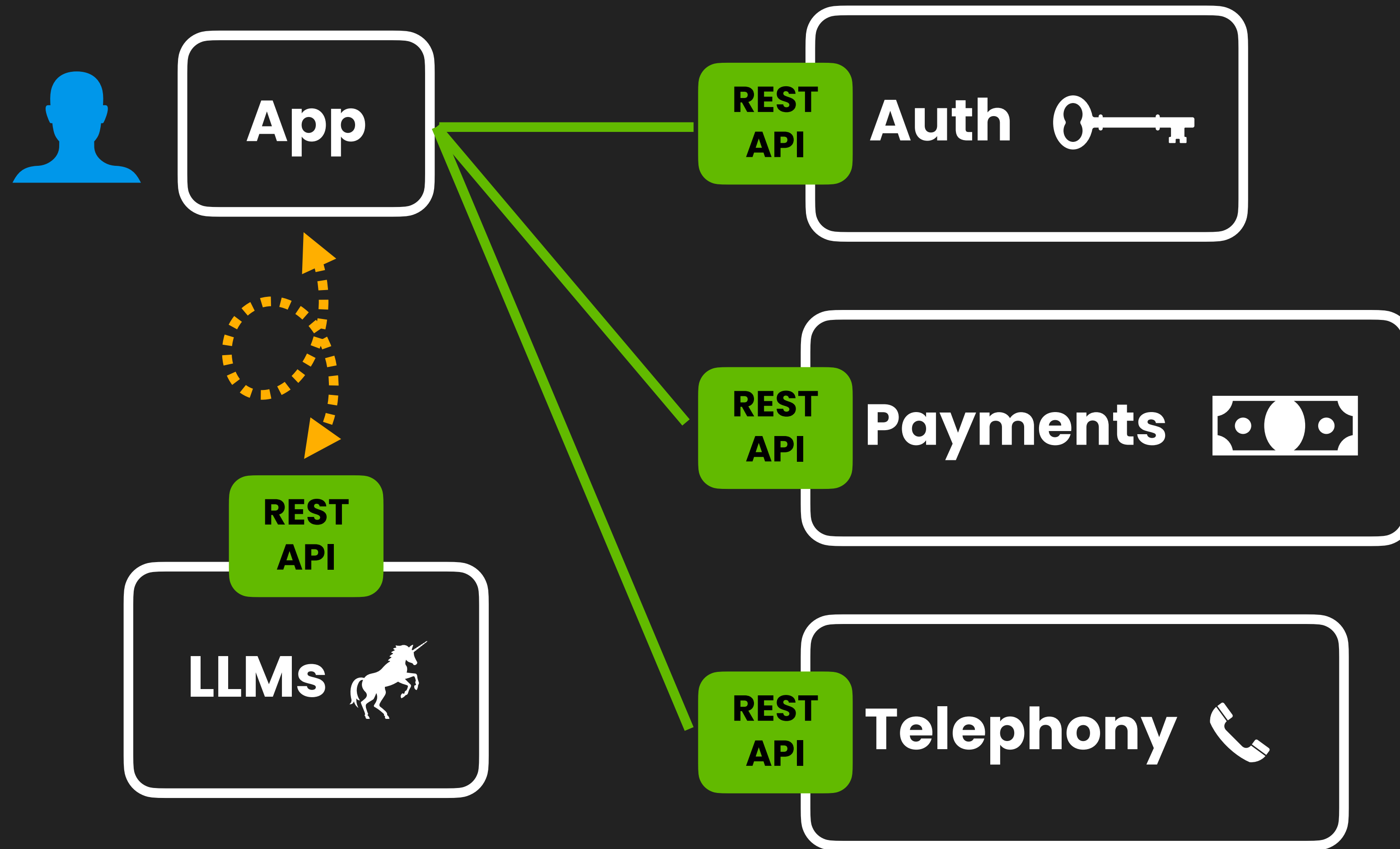


- ▶ **Well-formed inputs** according to a spec
- ▶ **Cleaned-up user** inputs
- ▶ **Well-formed outputs** according to a spec
- ▶ **Standard** protocols (e.g. HTTP, SMTP)

= **testable**
mockable



LLMs != like APIs we know and ❤️



~~predictable~~
∴ ~~testable~~
∴ ~~mockable~~

LLMs != like APIs we know and

Normal APIs

unit tests

can conceivably scope the range of inputs

reproducible
(AKA mockable)

deterministic +
(ideally) idempotent

explainable
(AKA debuggable)

based on spec, can understand how a change in input → change in output

LLMs

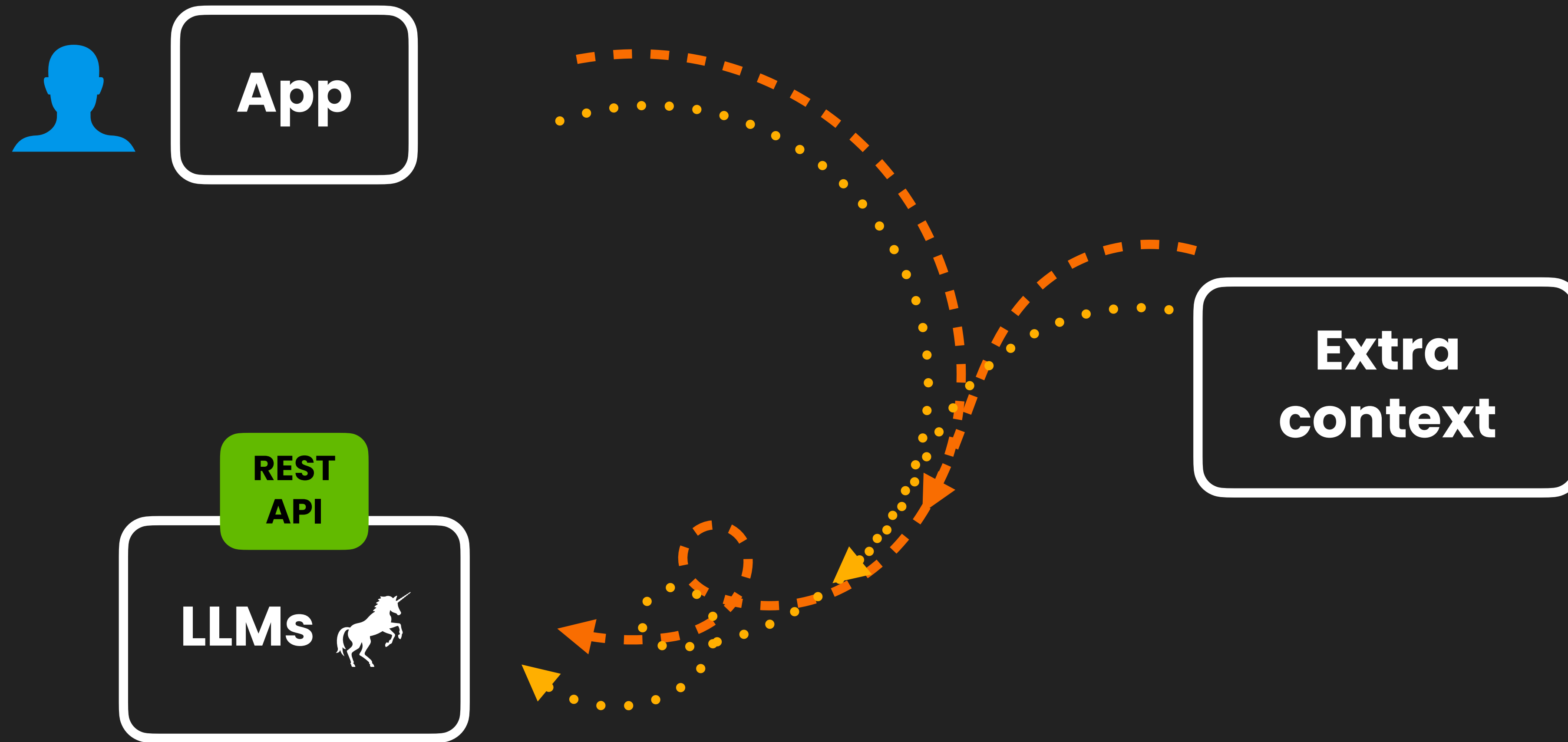
intentionally invites free-form, natural-language input from users

subject to change ("drift" in model behavior) via public API access

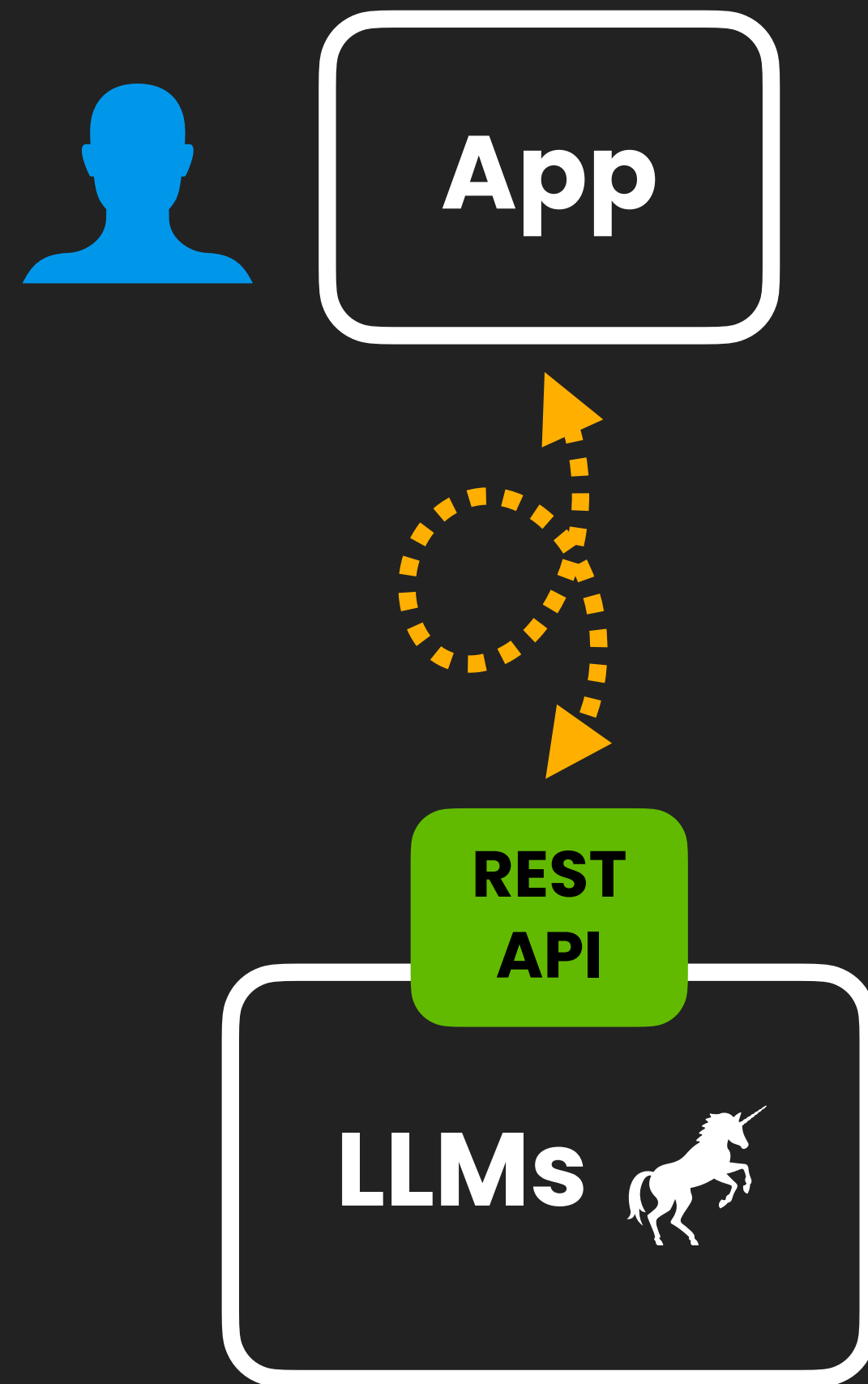
prompting can yield very different responses through small, subtle changes to prompt



LLMs: even more unpredictability



LLMs: how do we define "correct"?



"early access"

staging env


integration tests

unit tests





observability



AKA: an understanding the
behavior of a system based on
knowledge of its external outputs.

observability

expected

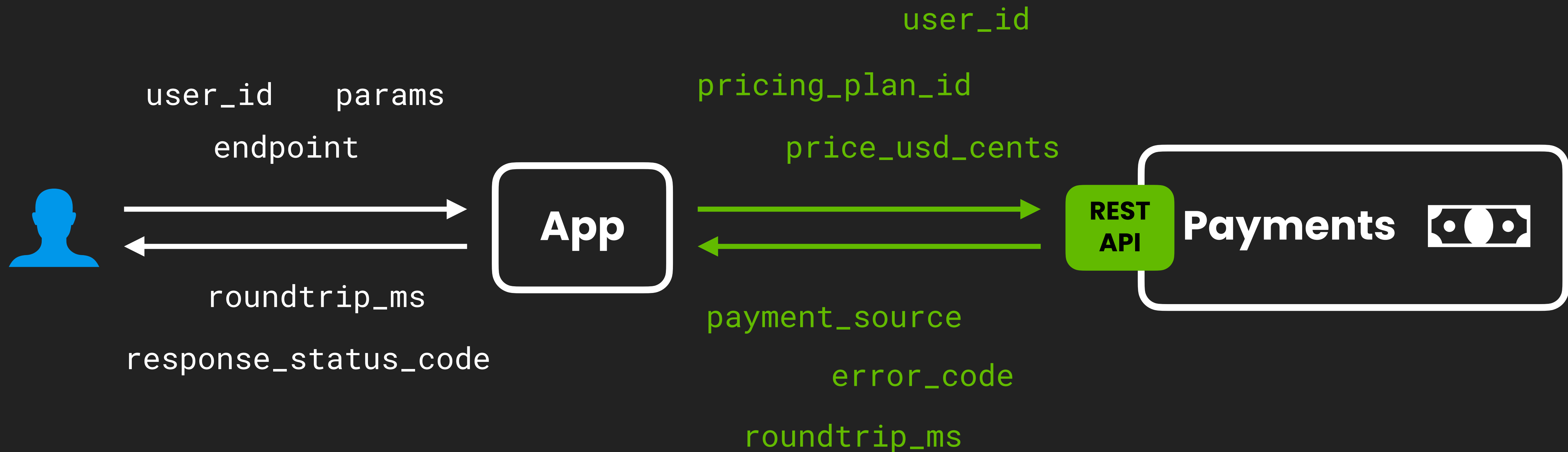


actual

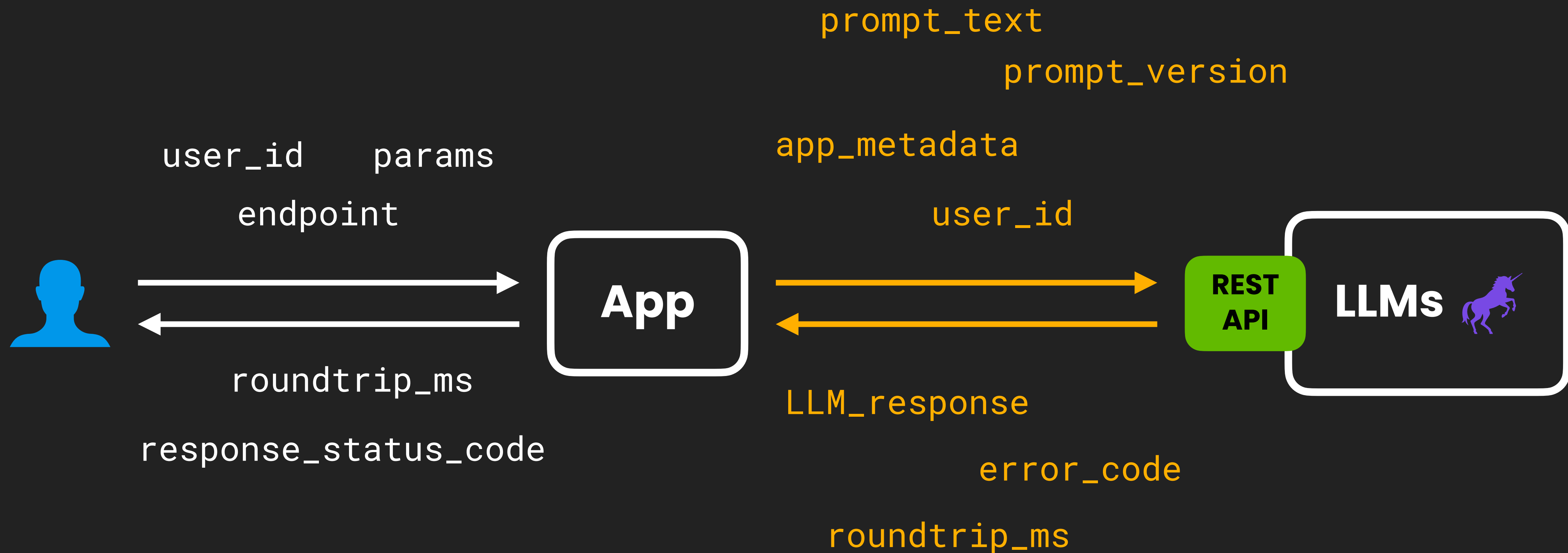
(especially in prod!)

observability

Observability: what's in the box?

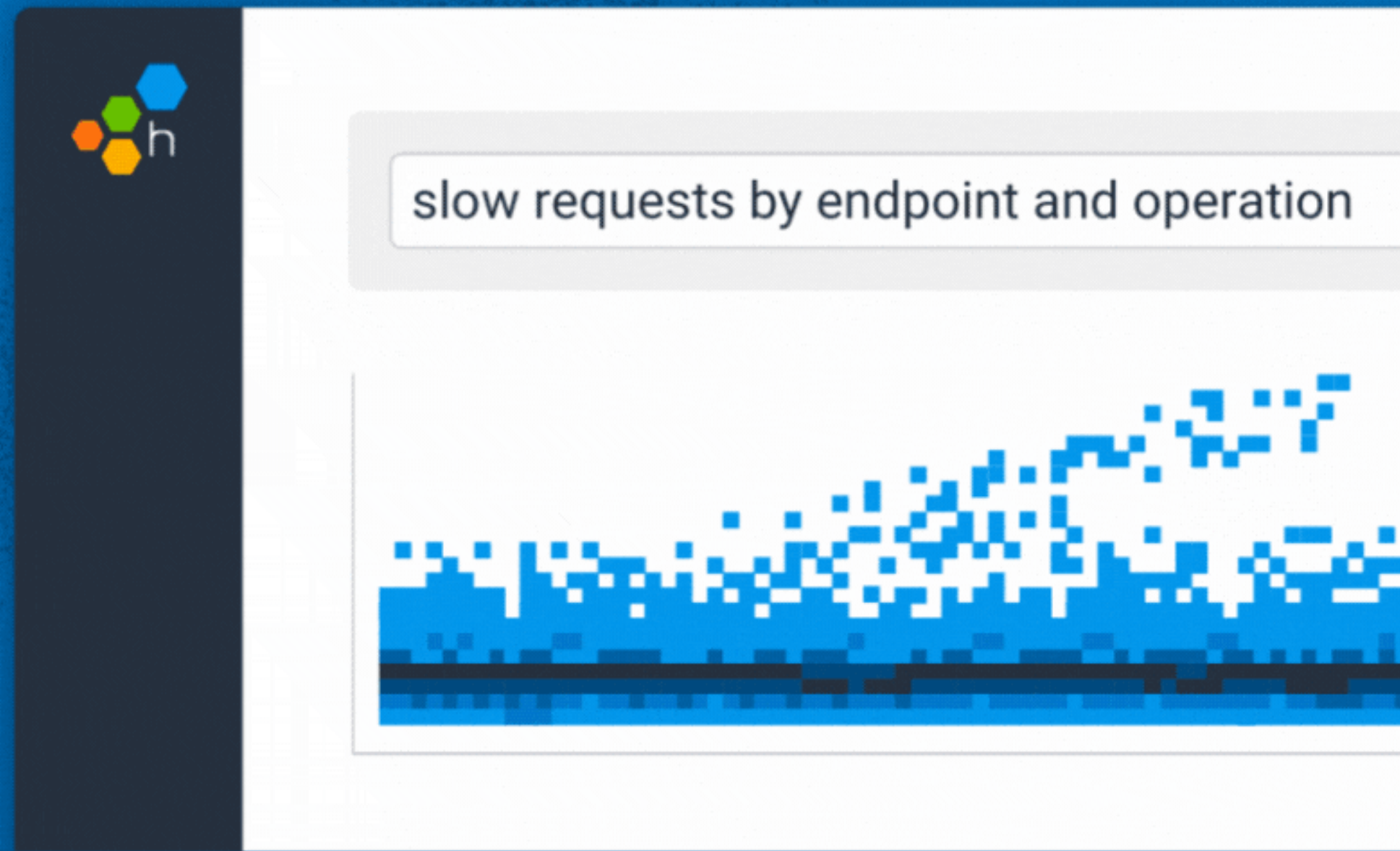


Observability: what's in the box?



Observability: ∞ feedback loops





Why believe me?

Query Assistant: timeline

○ 6 weeks of development

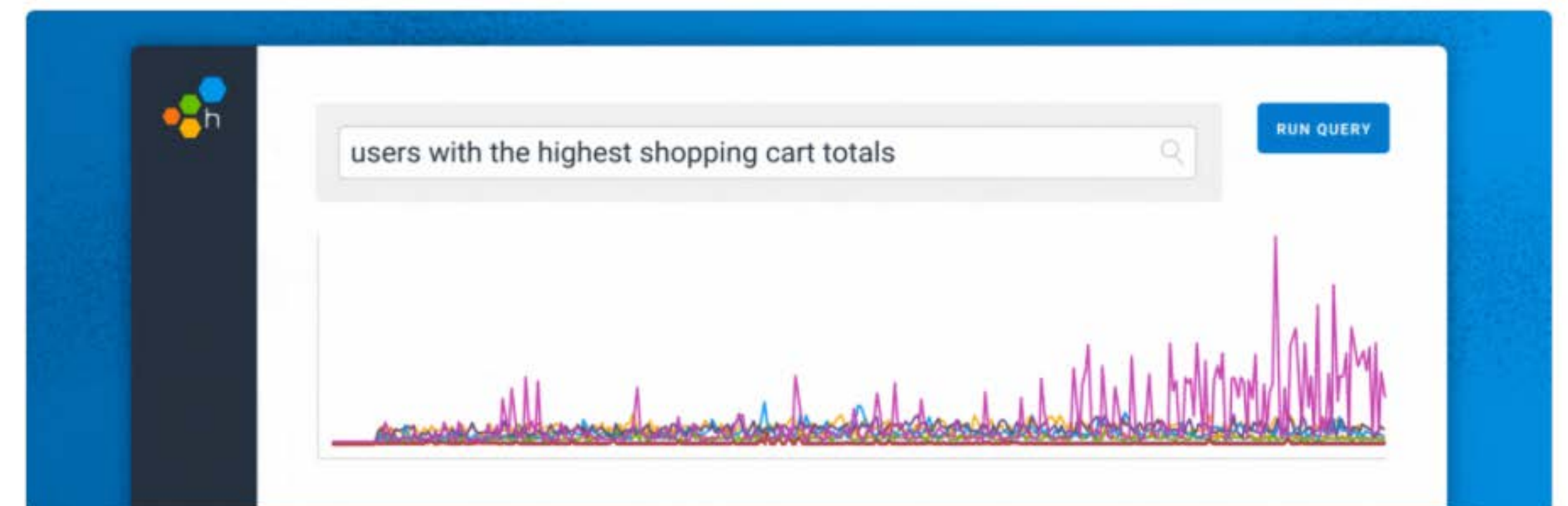
● May 2023

○ 8 weeks of iteration

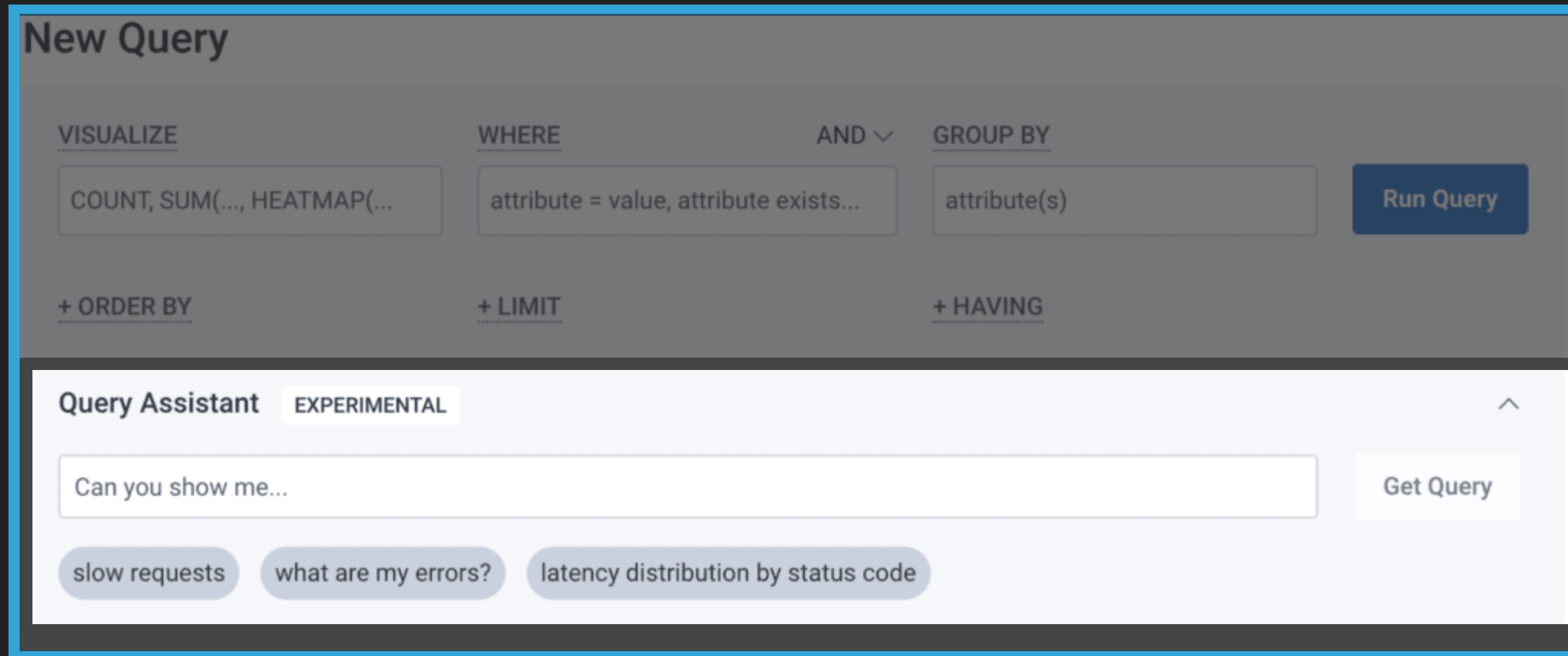
[Product Updates](#)

Observability, Meet Natural Language Querying with Query Assistant

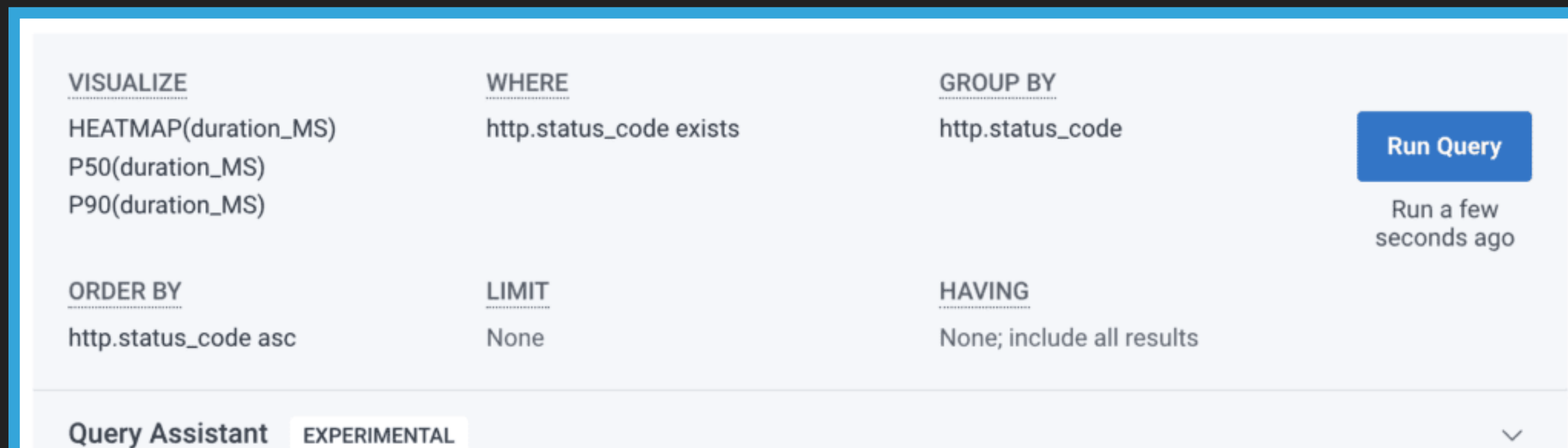
By [Phillip Carter](#) | Published May 3, 2023



Query Assistant: goals



The screenshot shows the 'New Query' interface. At the top, there are sections for 'VISUALIZE', 'WHERE', 'AND', and 'GROUP BY'. Below these are input fields for 'COUNT, SUM(..., HEATMAP(...)', 'attribute = value, attribute exists...', and 'attribute(s)'. A 'Run Query' button is visible. Below the main query builder, there is a 'Query Assistant' section labeled 'EXPERIMENTAL'. It contains a text input field with the placeholder 'Can you show me...' and a 'Get Query' button. Below the input field are three suggested queries: 'slow requests', 'what are my errors?', and 'latency distribution by status code'.



The screenshot shows the generated query interface. It displays the following configuration:

VISUALIZE	WHERE	GROUP BY
HEATMAP(duration_MS) P50(duration_MS) P90(duration_MS)	http.status_code exists	http.status_code

Below the main query builder, there is a 'Query Assistant' section labeled 'EXPERIMENTAL'.

ORDER BY	LIMIT	HAVING
http.status_code asc	None	None; include all results

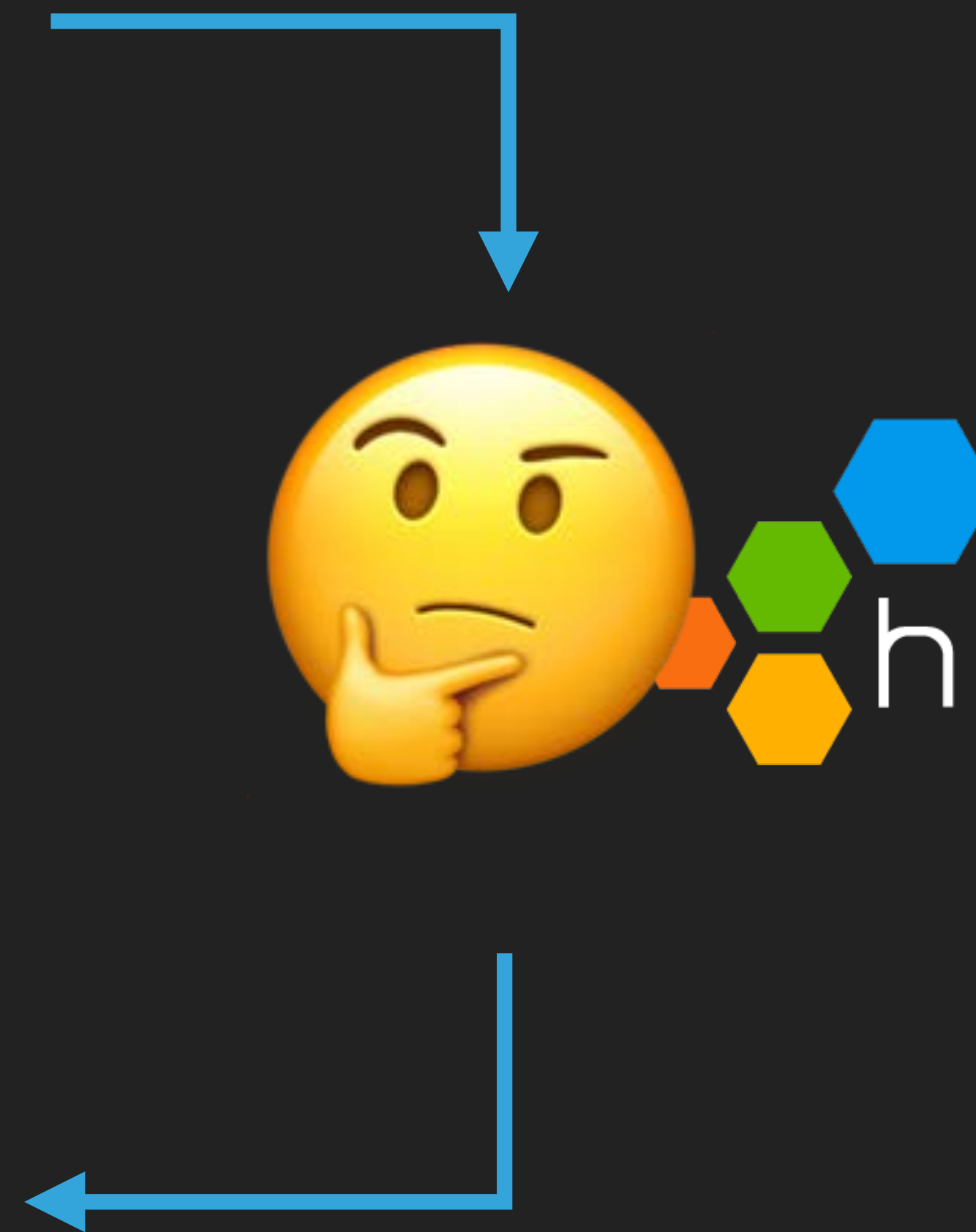
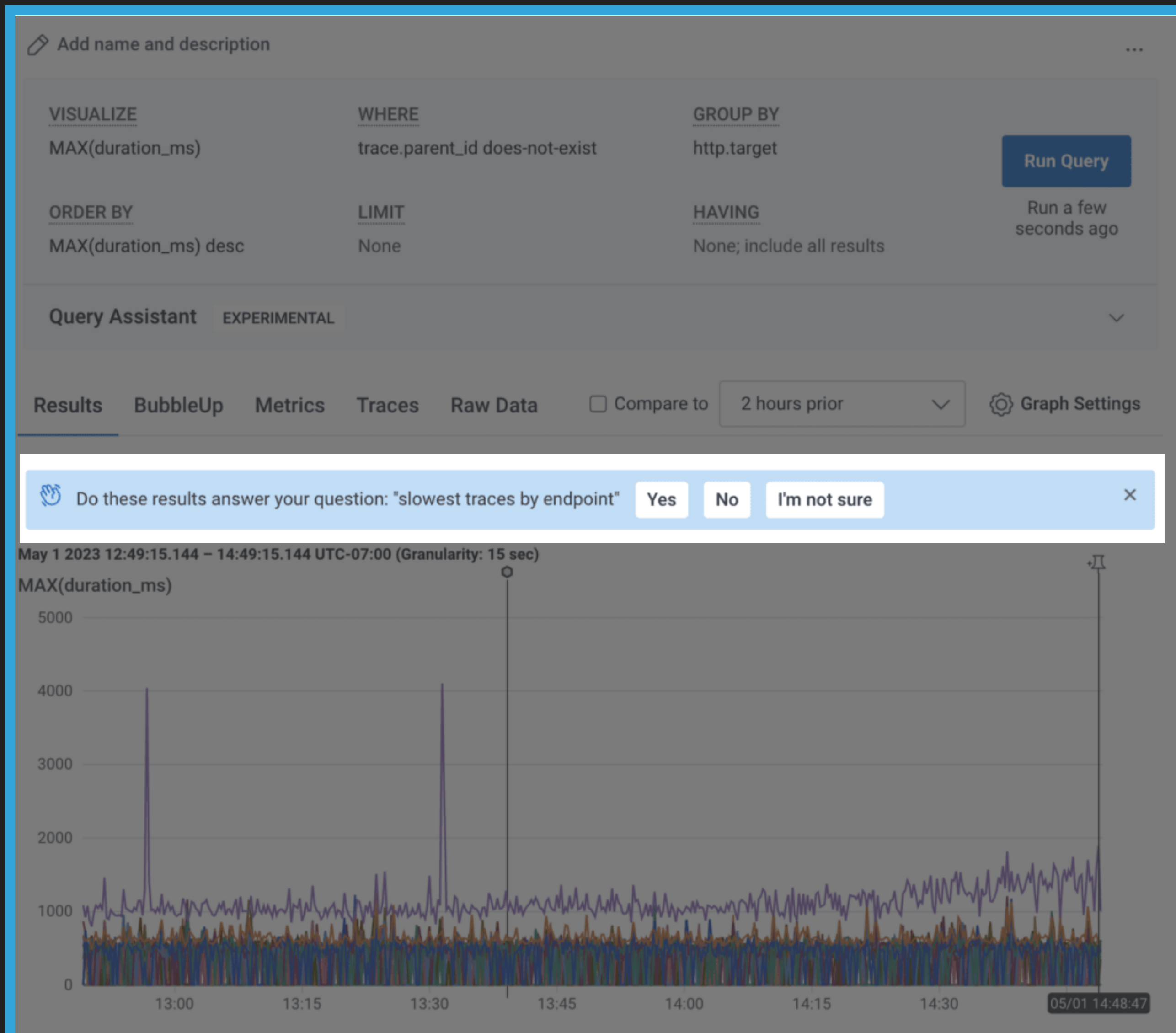
A 'Run Query' button is visible, with a timestamp 'Run a few seconds ago' below it.



"What's the 95th percentile latency on the /checkout endpoint?"



Query Assistant: goals

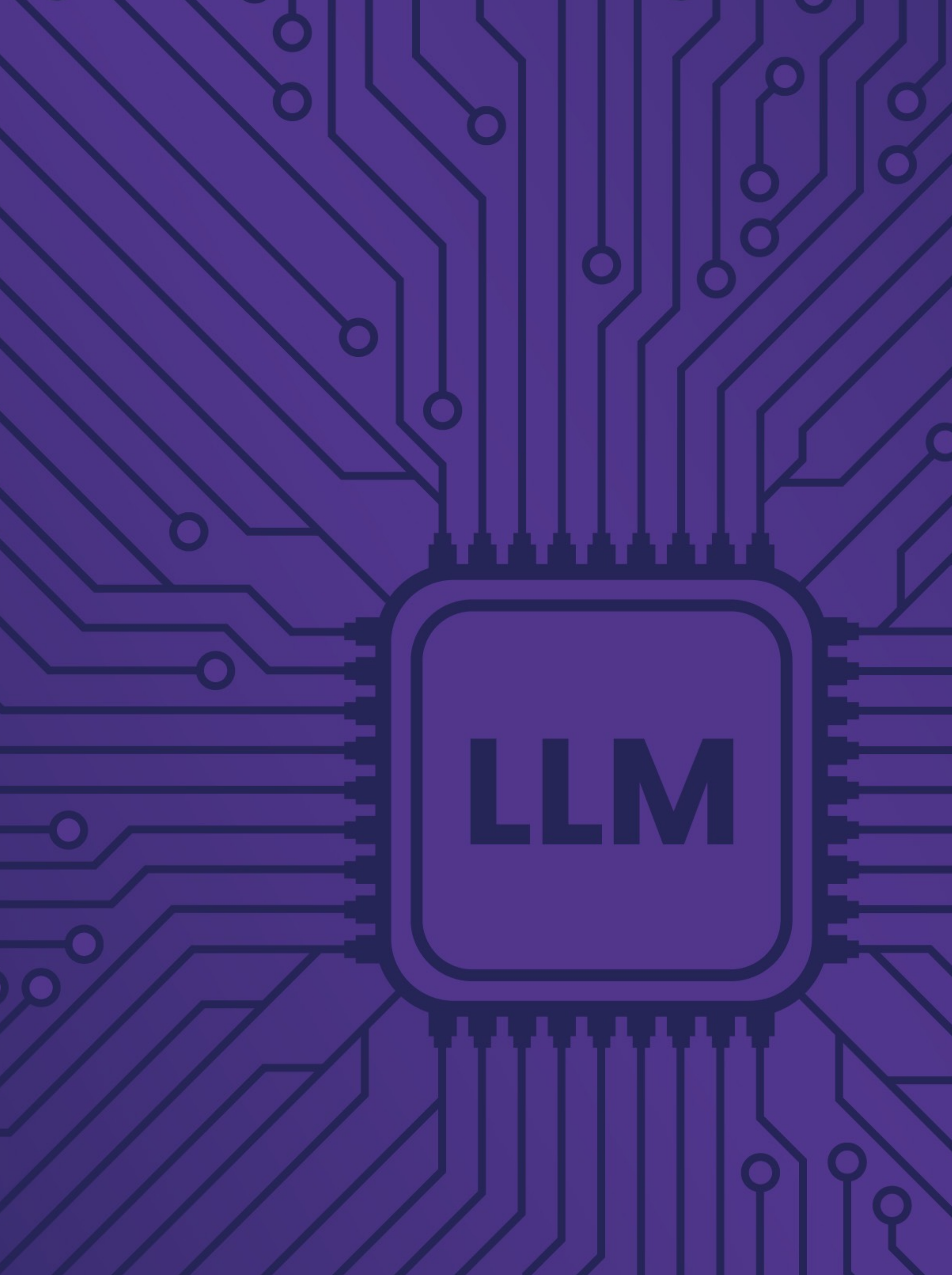


Laws of building on LLMs

- ▶ Failure will happen—it's a question of when, not if.
- ▶ Users will do things you can't possibly predict.
- ▶ You will ship a "bug fix" that breaks something else.
- ▶ You can't really write unit tests for this (nor practice TDD)
- ▶ Latency is often unpredictable
- ▶ Early access programs won't help you

<https://honeycomb.io/blog/hard-stuff-nobody-talks-about-llm>





OK, so

**How *do* we go
forward?**

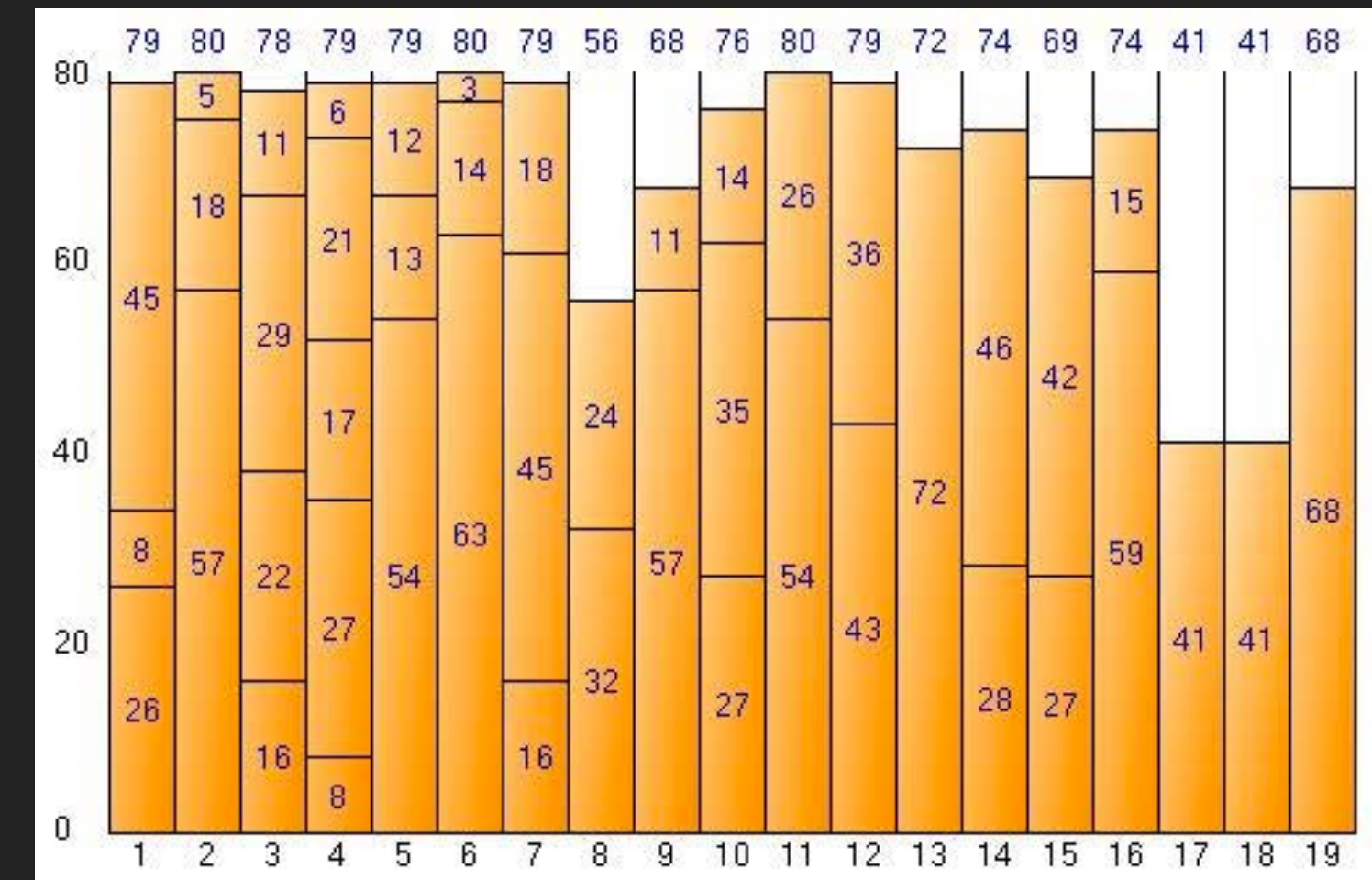
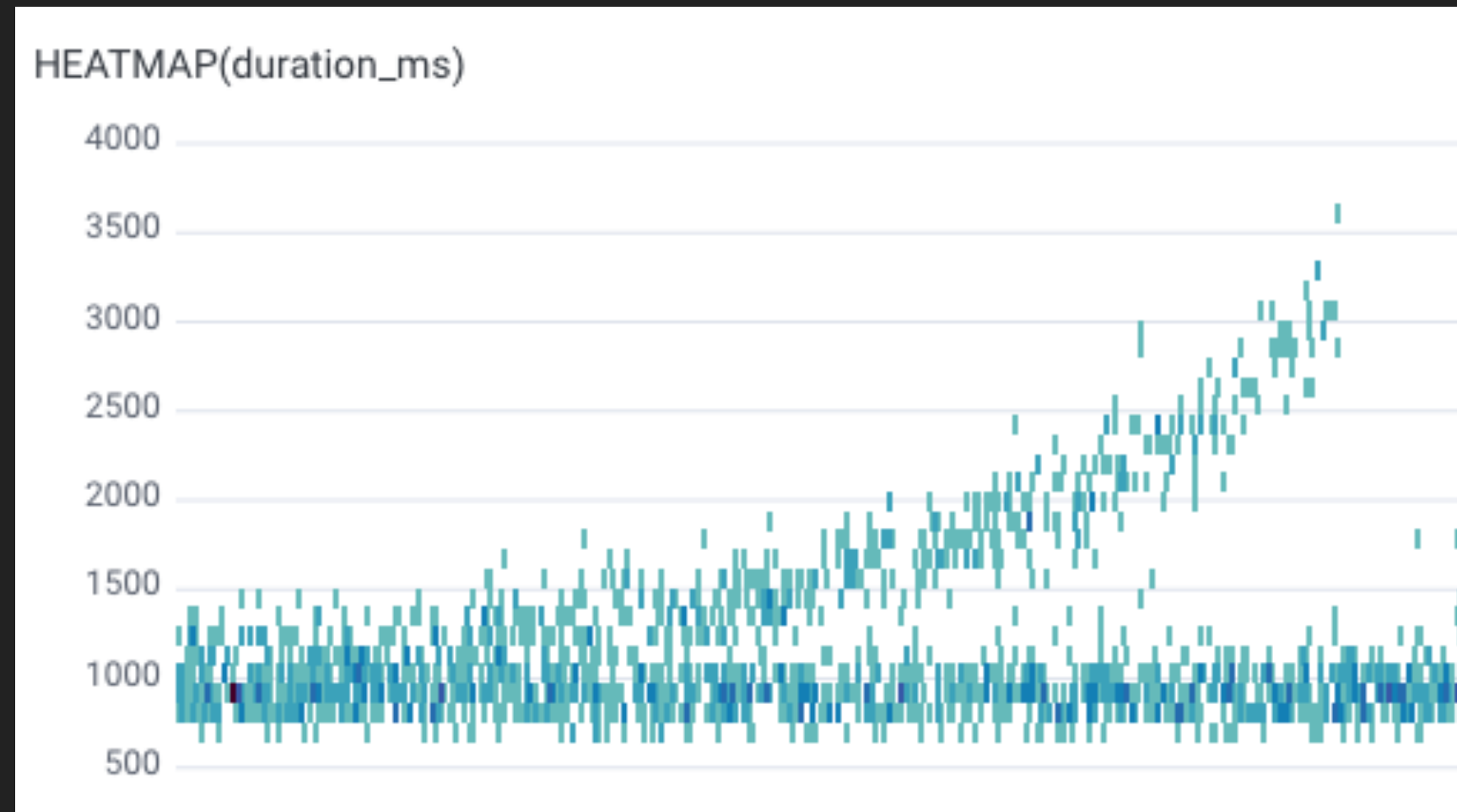
Instrumentation \approx docs and tests



capture data for your hypotheses



Instrumentation \approx docs and tests



capture data for your hypotheses



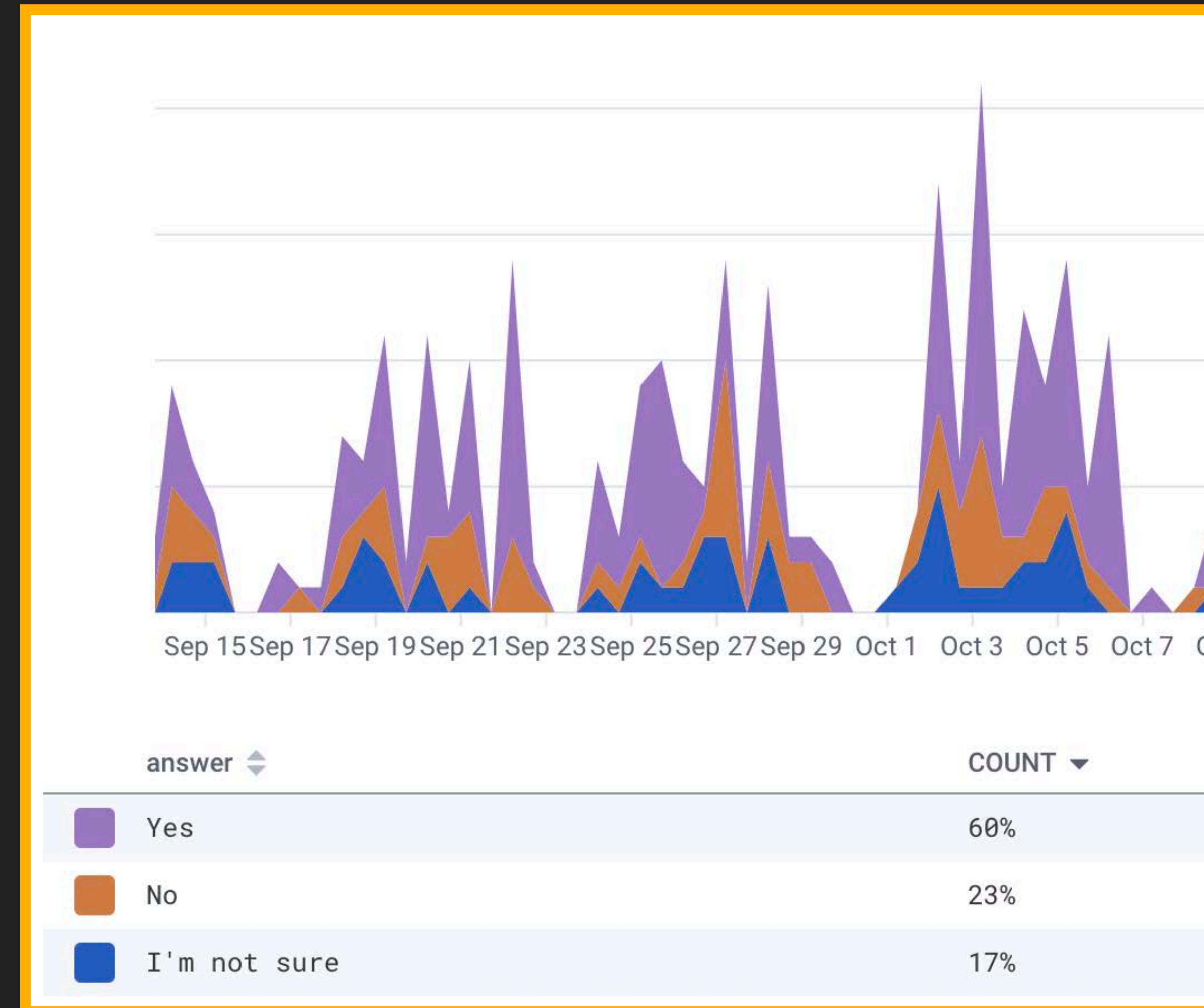
Instrumentation for LLMs

- ▶ user/team IDs
- ▶ full user input string
- ▶ add'l product context for prompt
- ▶ token usage
- ▶ LLM latency
- ▶ full LLM response
- ▶ parse and/or validation errors
- ▶ user feedback



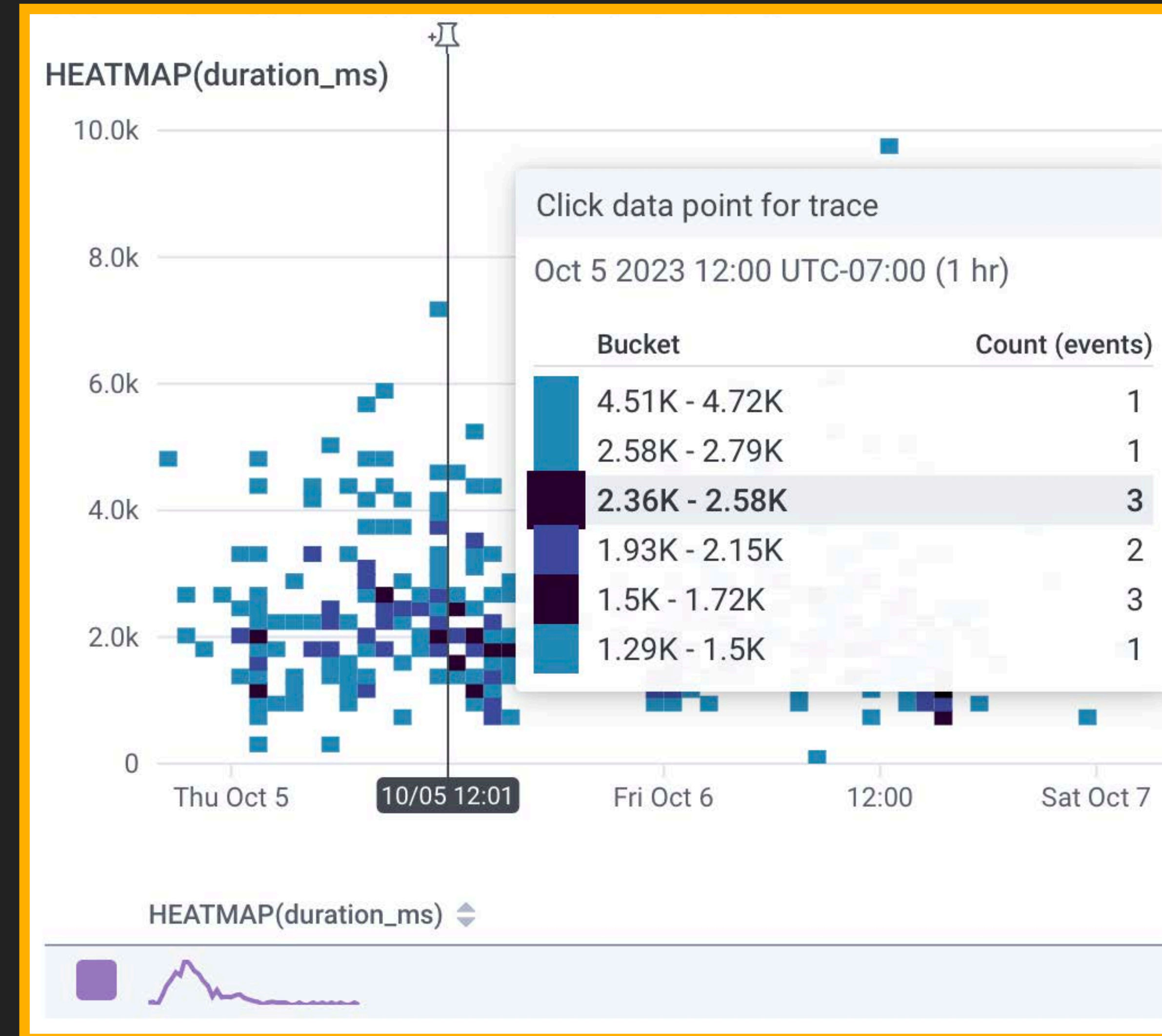
Instrumentation for LLMs

- ▶ user/team IDs
- ▶ full user input string
- ▶ add'l product context for prompt
- ▶ token usage
- ▶ LLM latency
- ▶ full LLM response
- ▶ parse and/or validation errors
- ▶ **user feedback**



Instrumentation for LLMs

- ▶ user/team IDs
- ▶ full user input string
- ▶ add'l product context for prompt
- ▶ token usage
- ▶ **LLM latency**
- ▶ full LLM response
- ▶ parse and/or validation errors
- ▶ user feedback




Instrumentation for LLMs

- ▶ **user/team IDs**
- ▶ full user input string
- ▶ **add'l product context for prompt**
- ▶ **token usage**
- ▶ LLM latency
- ▶ full LLM response
- ▶ parse and/or validation errors
- ▶ user feedback

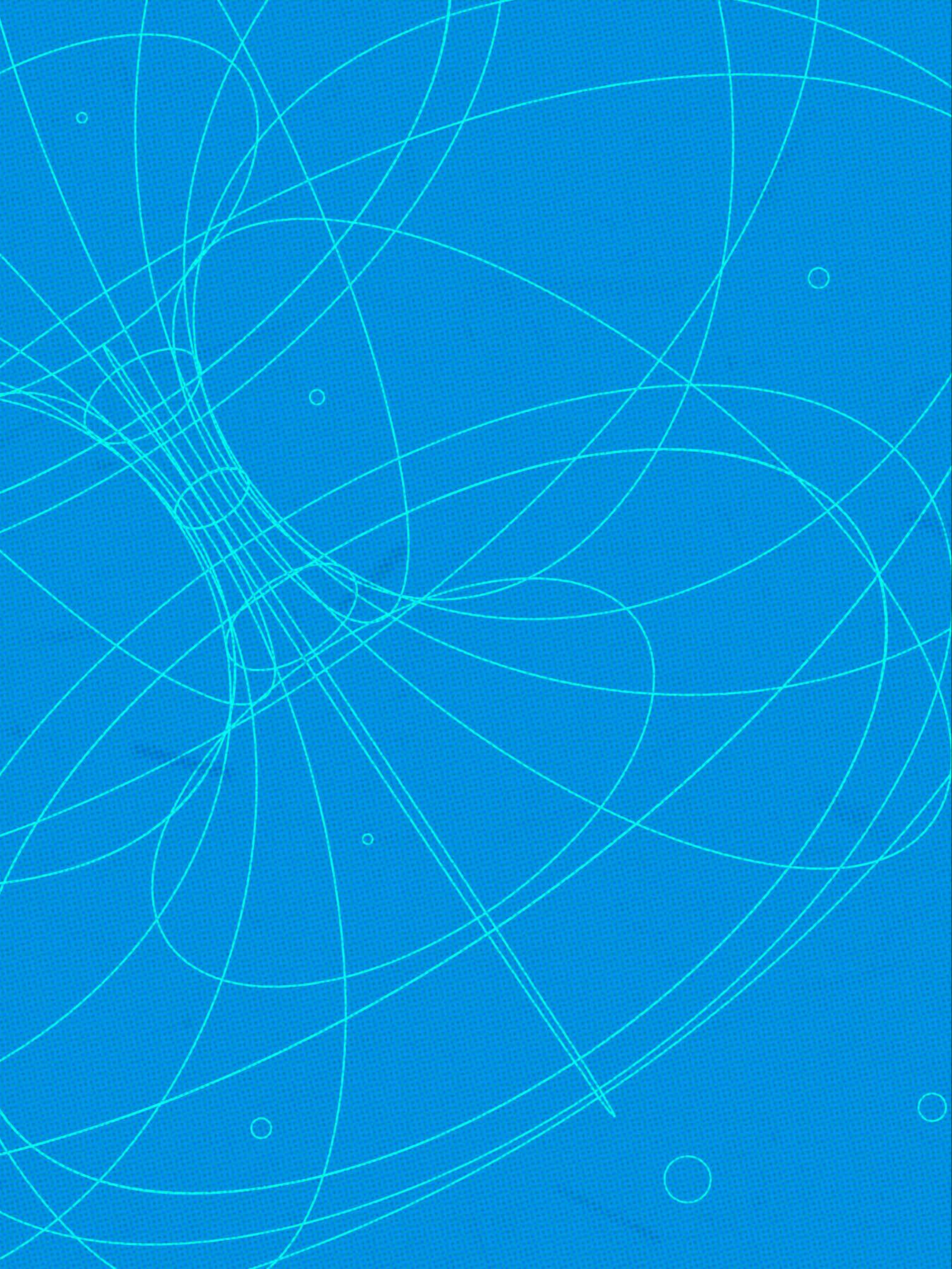


Instrumentation > EXCEPTIONS

- ▶ user/team IDs
- ▶ full user input string
- ▶ add'l product context for prompt
- ▶ token usage
- ▶ LLM latency
- ▶ full LLM response
- ▶ parse and/or validation **errors**
- ▶ user feedback



	app.nlq.user_input	error	app.nlq.response
	ff05f572-6587-4831-abe4-5bffa2108011	ML response does not contain valid JSON	I'm sorry, but I cou
	p95 duration_ms converted to seconds where name starts-with test_unit_dao grouped by name	received unexpected field "unit"	{"breakdowns":["name
	perform the current query grouped by ip	unexpected end of JSON input	{"breakdowns":["user
	slow requests	Post "https://api.openai.com/v1/chat/completions": context canceled	
	rds free storage available	ML response does not contain valid JSON	{"breakdowns": ["amazonaws.com/AWS/
	events with highest count	Post "https://api.openai.com/v1/chat/completions": context canceled	
	where os.device.sdk starts with a 2 or 3	unexpected end of JSON input	{"breakdowns":["app. {"column":"app.device



Tapping into

Emerging behaviors

DEV

WRITE → TEST → COMMIT → WRITE → TEST → COMMIT
→ WRITE → TEST → COMMIT → WRITE → TEST → COMMIT
→ WRITE → TEST → COMMIT → WRITE → TEST → COMMIT
→ WRITE → TEST → COMMIT → WRITE → TEST → COMMIT



DEV

write lots of code
service ownership
developers on call
test in production



DEVD

identify levers impacting logical branches in code
(debuggability + reproducibility)

compare **expected** vs **actual**

fail fast / fail first;
embrace fast feedback loops

BROD

instrument code with intention

inspect results after changes go live; watch for deviations

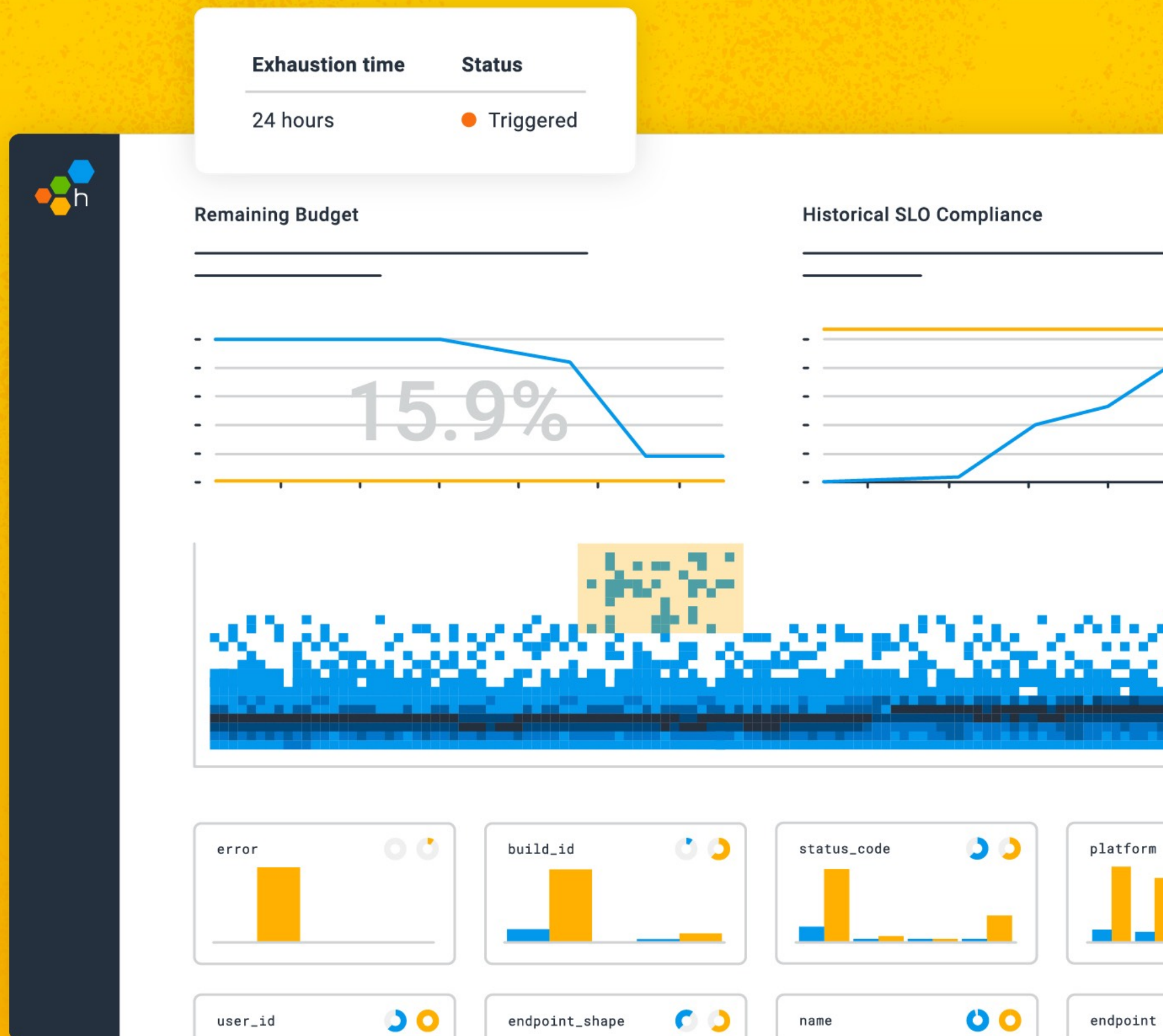
ship to prod quickly (CI/CD);
expect to iterate



**A truth in all software systems,
but never more true than with LLMs:**

Software behaves in unpredictable, emergent ways,
and the important part is observing your code
as it's running in production, while users are using it.





Let's zoom in on

Service Level Objectives

SLOs: a quick definition

Service Level Objectives codify what it means to **"deliver great service"**

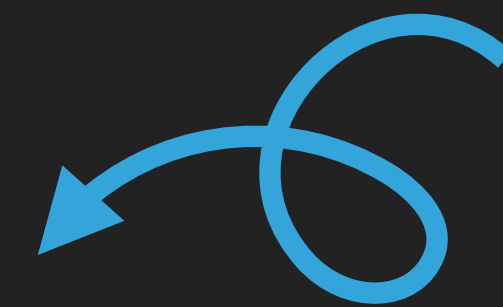
- ▶ "Key user flows like cart checkout should complete quickly and reliably"
- ▶ "99.9% of shopping cart checkout attempts complete error-free in < Xs"



Laws of building on LLMs

- ▶ Failure will happen—it's a question of when, not if.
- ▶ Users will do things you can't possibly predict.
- ▶ You will ship a "bug fix" that breaks something else.
- ▶ You can't really write unit tests for this (nor practice TDD)
- ▶ Latency is often unpredictable
- ▶ Early access programs won't help you

- ▶ **LINK TO: hard things about hard things blog post**



Remember this?

Degradation will happen.

SLOs can help.



SLOs for developing with LLMs

Historical SLO Compliance

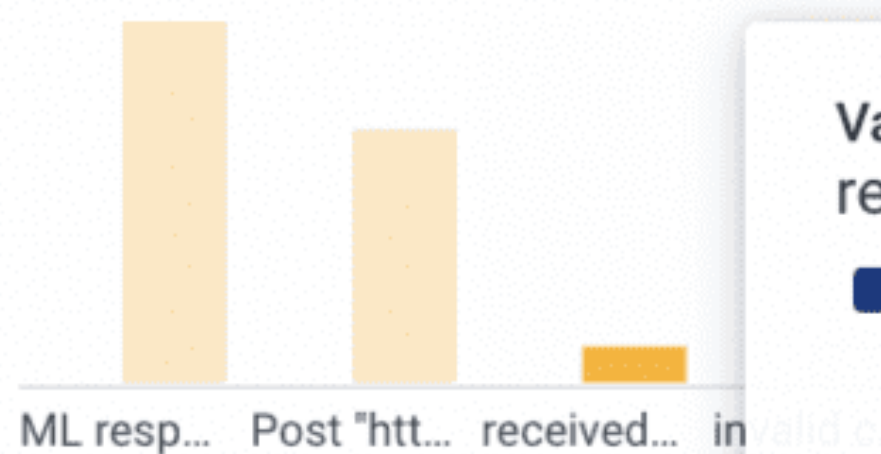
For each day of the past 7, how often this SLI has succeeded over the preceding 7 days.



Dimensions

Try to `{..}` GROUP BY columns that look most different between the ■ successful and ■ failed.

error



app.nlq.response

Value
received unexpected field "value"

■ Successful ■ Failed
0% 5%

Click column for actions



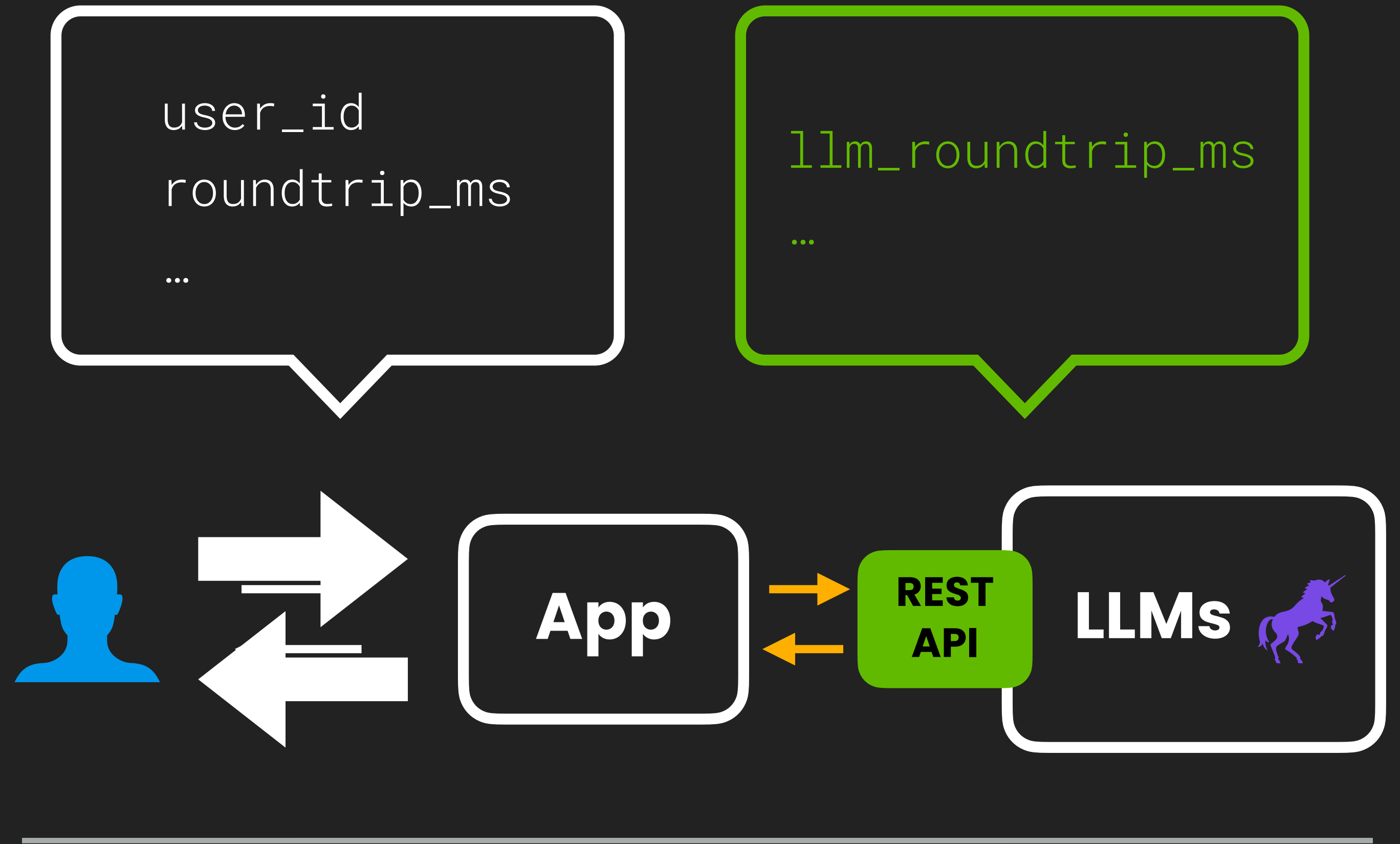


Some more stories

**From others in
the wild**

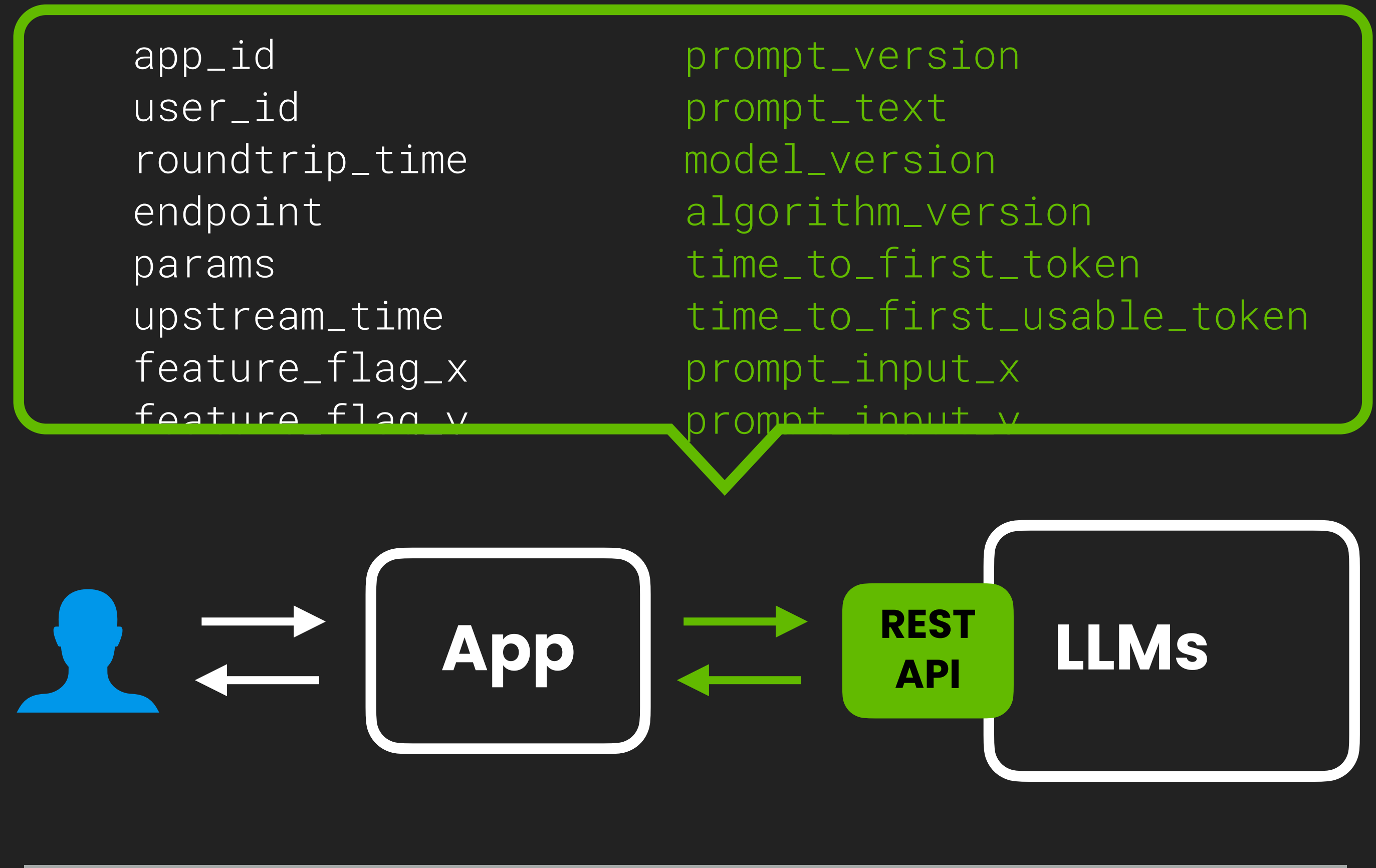


duolingo





INTERCOM



So in the end:

- ▶ Incorporating LLMs breaks many of our existing tools for ensuring correctness + a good user experience
- ▶ Observability can help! Instrument + observe from the outside in
- ▶ Capture all the metadata to be able to debug and analyze unexpected behavior in LLMs
- ▶ Embrace the unpredictability of user input + LLMs: run in production and plan to iterate *fast*



More resources:

<https://honeycomb.io/blog/hard-stuff-nobody-talks-about-llm>

<https://honeycomb.io/blog/improving-llms-production-observability>

<https://honeycomb.io/blog/llms-demand-observability-driven-development>

<https://honeycomb.io/blog/we-shipped-ai-product>

thanks!

q?

@cyen

@honeycombio

