## Leveraging SRE and Observability Techniques

### for the Wild World of Building on LLMs

@cyen
@honeycombio





### LLMs $\approx$ like APIs we know and $\checkmark$



- 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 \vertice









### LLMs != like APIs we know and \(\circ)

### unit tests

reproducible (AKA mockable)

explainable (AKA debuggable)

### Normal APIs LLMs

can conceivably scope the range of inputs

deterministic + (ideally) idempotent

based on spec, can understand how a change in input  $\rightarrow$ change in output

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



Extra context



### LLMs: how do we define "correct"?





### "early access"

### staging env

### integration tests

### unit tests







## observability





## observability

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











### Observability: what's in the box?





### Observability: what's in the box?



response\_status\_code

### prompt\_text

prompt\_version

### app\_metadata

user\_id

 REST LLMS 
LLM\_response
 error\_code

roundtrip\_ms



### Observability: co feedback loops

# $IDEATE \rightarrow WRITE \rightarrow TEST \rightarrow RELEASE \rightarrow OBSERVE$ $\uparrow TEST \rightarrow IDEATE \rightarrow ID$





Why believe me?

### Query Assistant: timeline

### 6 weeks of development

### May 2023

### **O** 8 weeks of iteration



Product Updates

### **Observability, Meet Natural Language Querying with Query Assistant**

By Phillip Carter | Published May 3, 2023

users with the highest shopping cart totals	RUN QUERY
multiment and how the state of	MMM
	users with the highest shopping cart totals



### Query Assistant: goals

### **New Query**

+ ORDER BY	+ LIMIT	+ HAVING	
Query Assistant EXPERIMENTAL	L		^
Can you show me			Get Query
slow requests what are my er	rors? latency distribution by statu	s code	
VICUALIZE	WHERE		
HEATMAP(duration_MS)	http.status_code exists	http.status_code	Run Query
I AND LAND LAND AND AND AND AND AND AND AND AND AND			
P90(duration_MS)			Run a few seconds ago
P90(duration_MS)	LIMIT	HAVING	Run a few seconds ago

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



### Query Assistant: goals

🔗 Add name and description

VISU	JALIZE				WHERE					GROUP	BY			
MAX	(duratio	n_ms)			trace.pa	rent_id do	es-not-e>	list	ł	nttp.targ	get			
ORD MAX	ER BY	n_ms) de	esc		LIMIT None				<u> </u> 1	HAVING None; in	clude all i	results		S
Que	ery Assi	stant	EXPER	IMENTAL										
Resul	ts B	ubbleUp	M	etrics	Traces	Raw [	Data	Cor	mpare t	0 2	hours pric	or	~	© (
Ø D	)o these	results a	nswer	your que	estion: "slo	west trace	es by end	point"	Yes	No	l'm not	sure		
May 1 20	23 12:49:	15.144 – 1	14:49:1	5.144 UTC	C-07:00 (Gra	nularity: 15	i sec) O							
5000	iration_n	15)												
4000														
3000														
2000	_													Mar
1000	which	hum	mp m	Malin	Ampun	When	handa	Mall	white h	mych	MANNA	W Mh	Annal 1	WPU III
0		13:00		13:15	13	1144 11 1:30	13:45		14:00		14:15		14:30	



Run a few econds ago

 $\sim$ 

Graph Settings





### 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





# How do we go forward?

OK, SO

### Instrumentation ~= docs and tests



### capture data for your hypotheses



App

### Instrumentation ~= docs and tests



### capture data for your hypotheses





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

nt

13 GenerateQueryFromPrompt	2.256s
launchdarkly.BoolVariation	91.3µs
- launchdarkly.NumberVariation	22.9µs
- Iaunchdarkly.StringVariation	22.6µs
launchdarkly.StringVariation	17.2µs
• Schema Store Get	25.8µs
—• Schema Store Get	83.8µs
-3 queryml.FindAllSuggestedQueriesForDataset	4.143ms
- queryml.FindCustomdQueriesForDatas	7.59µs
-• SelectContext	3.506ms
queryml.FindDefaulttedQueriesForData	0.4152ms
-• classifier.GetStarterQueries	0.3516ms
-• classifier.GetStarterQueriesClassifierEr	18.9µs
-• classifier.GetStarteriesClassifierHT	21.9µs
-• classifier.GetStartsClassifierDataba	20.2µs
-• classifier.GetStarterQueriesClassifierE	11.3µs
- classifier.GetStarteriesClassifierNgi	12.0µs
classifier.GetStartierKubernetes Eve	7.89µs
- 1 queryml.MostRelevantColumns	215.8ms
queryml.filterColumnsUsingEmbeddings	215.7ms
-4 queryml.embeddings.Embeddings	201.5ms
- queryml.embeddings.getFromCache	0.9509ms
• queryml.embeddshalEmbeddi	22.5µs
TruncateColumnList	46.4µs
-• openai.Embeddings	199.4ms
queryml.embeddings.saveInCache	0.9284ms
queryml.embeddings.Embeddings	13.90ms
queryml.embeddings.getFromCache	13.75ms
• queryml.embeddshalEmbeddi	9.295ms
• CreateChatPrompt	16.95ms
• CreateCustomExamplesPromptText	89.4µs
TruncateColumnList	0.4441ms
- 1 queryml.GenerateQuery	2.016s
openai.ChatCompletion	2.016s
randar ISON	22 10



- 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



answer 🗢	COUNT
Yes	60%
No	23%
I'm not sure	17%
	answer 🌲 Yes No I'm not sure



- 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



HEATMAP(duration\_ms) =

- 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.team.id 🧅	app.user.id 🜲	app.nlq.input_tokens_us
13	85170	1500-1750
13	48009	1500-1750
62,519	152741	1750-2000
84,002	171917	1750-2000
13	43516	1500-1750
84,002	171918	1750-2000
48,775	142184	2000-3000

### **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

	1	0				
		21:00	Wed Oct 11	03:00	06:00	09:00
50		app.nlq.user_input 🖨	error 🜲			app.nlq.re
		ff05f572- 6587-4831-abe4- 5bffa2108011	ML response	e does not con	tain valid JSON	I'm sor
		p95 duration_ms converted to seconds where name starts-with test_unit_dao grouped by name	received un	nexpected fiel	d "unit"	{"break
		perform the current query grouped by ip	unexpected	end of JSON i	nput	{"break
	8	slow requests	Post "https://ap context car	ions":		
		rds free storage available	ML response	e does not con	tain valid JSON	{"break ["amazo
		events with highest count	Post "https://ap context car	oi.openai.com/ nceled	v1/chat/complet	ions":
		where os.device.sdk starts with a 2 or 3	unexpected	end of JSON i	nput	{"break {"colum



- 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

name 🗸 🍚	0s 0.5s 1s 1.5s 2s 2.5s
<ul> <li>Schema Store Get</li> </ul>	11.4µs
- 2 queryml.FindAllSuggestedQueriesForDataset	0.4712ms
queryml.FindCustomSuggestedQueriesForDataset	5.43µs
queryml.FindDefaultSuggestedQueriesForDataset	0.3775ms
-1 queryml.MostRelevantColumns	503.0ms
2 queryml.filterColumnsUsingEmbeddings	502.9ms
-4 queryml.embeddings.Embeddings	496.7ms
- 1 queryml.embeddings.getFromCache	0.4856ms
queryml.embeddings.unmarshalEmbeddings	21.3µs
TruncateColumnList	65.8µs
openai.Embeddings	495.4ms
• queryml.embeddings.saveInCache	0.4965ms
queryml.embeddings.Embeddings	6.039ms
queryml.embeddings.getFromCache	5.939ms
• queryml.embeddings.unmarshalEmbeddings	4.309ms
-• CreateChatPrompt	19.27ms
<ul> <li>CreateCustomExamplesPromptText</li> </ul>	70.6µs
<ul> <li>TruncateColumnList</li> </ul>	0.3755ms
-1 queryml.GenerateQuery	3.498s
•• openai.ChatCompletion	3.498s
renderJSON	





### Tapping into

### Emerging behaviors

# 

### WRITE $\rightarrow$ TEST $\rightarrow$ COMMIT $\rightarrow$ WRITE $\rightarrow$ TEST $\rightarrow$ COMMIT





### 

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



## 

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

compare expected vs actual

fail fast / fail first; embrace fast feedback loops

# 

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.







### Some more stories

# From others in the wild



## duolingo





### INTERCOM

app\_id user\_id roundtrip\_time endpoint params upstream\_time feature\_flag\_x feature\_flag\_y prompt\_version
prompt\_text
model\_version
algorithm\_version
time\_to\_first\_token
time\_to\_first\_usable\_token
prompt\_input\_x
prompt\_input\_y

App Rest App LLMs





### 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

# 







