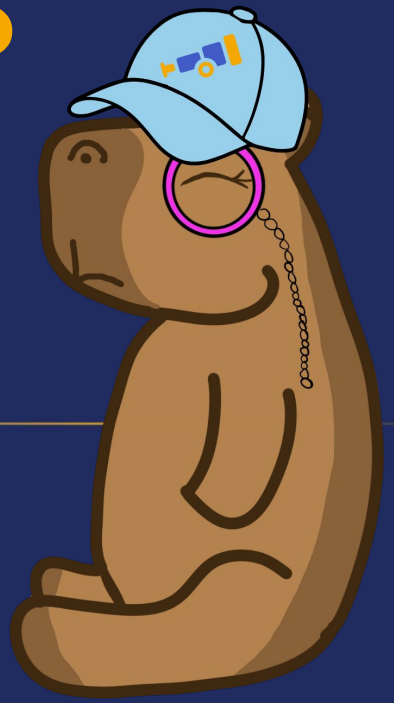
OpenTelemetry

OBSERVABILITY SUMMIT EUROPE

Your Agent Did WHAT?

Forensic observability for systems that don't leave obvious footprints



Adriana Villela
dynatrace

Kasper Borg Nissen
dash0

BEFORE WE START

Who are we?



Adriana Villela

Principal Developer Advocate

dynatrace

CNCF Ambassador · AAIF Ambassador

OpenTelemetry Community Manager

End User SIG maintainer

O'Reilly Author, *Fundamentals of Observability (Video)*

Capybara enthusiast

`Geeking Out podcast`



Kasper Borg Nissen

Director of Developer Relations

dash0

CNCF Ambassador · AAIF Ambassador

MergeForward Ambassador

Golden Kubestronaut

Author, *OpenTelemetry for Dummies*

Former Co-Chair, KubeCon + CloudNativeCon EU & NA

`Cloud Native Nordics`

TECHNOLOGY IS A MOVING TARGET

We keep changing what we have to understand

Monolith

Microservices

Event-driven

LLM applications



TECHNOLOGY IS A MOVING TARGET

The consequences of changing tech

- New tech tends to fix one thing, and break another
- In some ways, building software got easier
- Becoming “proficient” in new tech got harder
- In the age of LLM applications, understanding WTF is going on just got harder



How do we make sense of what our agents are doing to our systems...when things go




10 PM
DO YOU KNOW WHERE YOUR CHILDREN ARE?
AGENTS

01

Setting the stage: LLM Applications

UP NEXT

What the heck is an LLM app?

Terminology refresher

Anatomy of an agent

Complexity added by LLM

Good intentions, bad outcomes



BRUSHING UP ON SOME TERMINOLOGY!

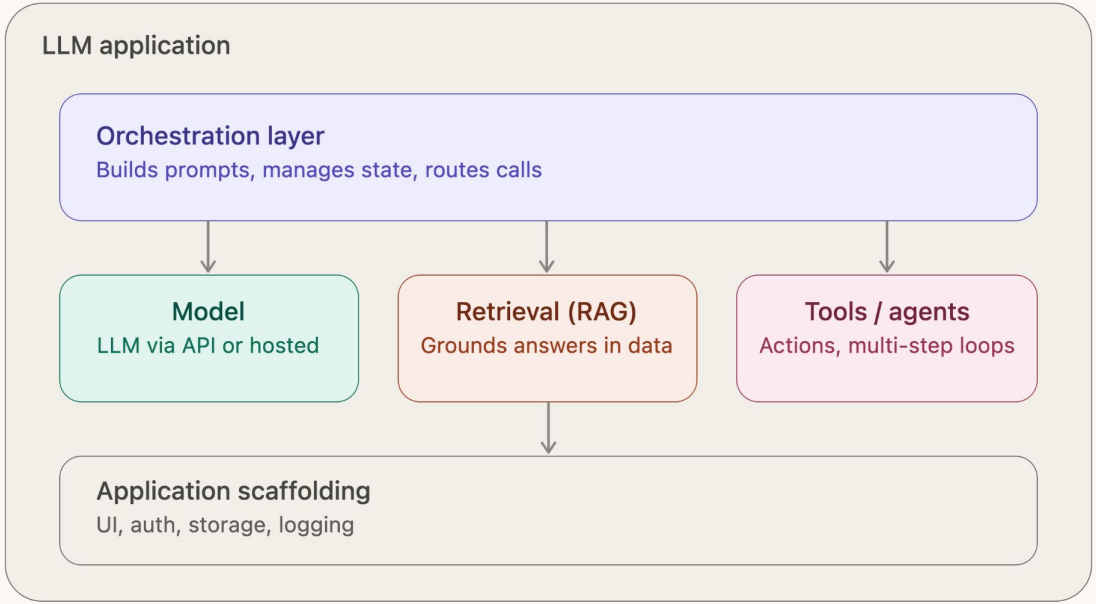
Terminology refresher

`provider`	Whoever hosts and serves the model. e.g. Anthropic, OpenAI, your own Ollama.
`model`	The thing that generates. e.g. Claude Sonnet, GPT, Qwen.
`MCP`	How an agent is handed tools it did not ship with. An API designed for models rather than people.
`agent`	Made up of a model, instructions, tools, and a loop.
`harness`	What runs the agent: memory, lifecycle, observability. Goose, Claude Code and Copilot are both agent and harness.



AGENTS ARE JUST ONE PART OF THE LANDSCAPE

What the heck is an LLM application??



** Existing instrumentation covers the bits in the application scaffolding



BRUSHING UP ON SOME TERMINOLOGY!

Anatomy of an agent

↺ AGENT

The loop. Keeps calling the model until it decides it is done.

→

✦ MODEL

Picks the next step. Billed per token, in and out.

→

⇆ MCP SERVER

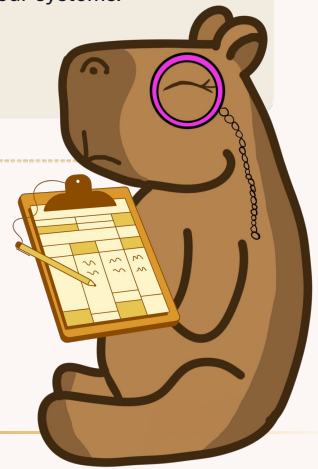
How an agent is handed tools it did not ship with.

→

⚙ TOOLS

The functions it can actually run against your systems.

↺ the agent decides when to stop



CLAUDE HAS ENTERED THE CHAT

LLMs make things more...complicated

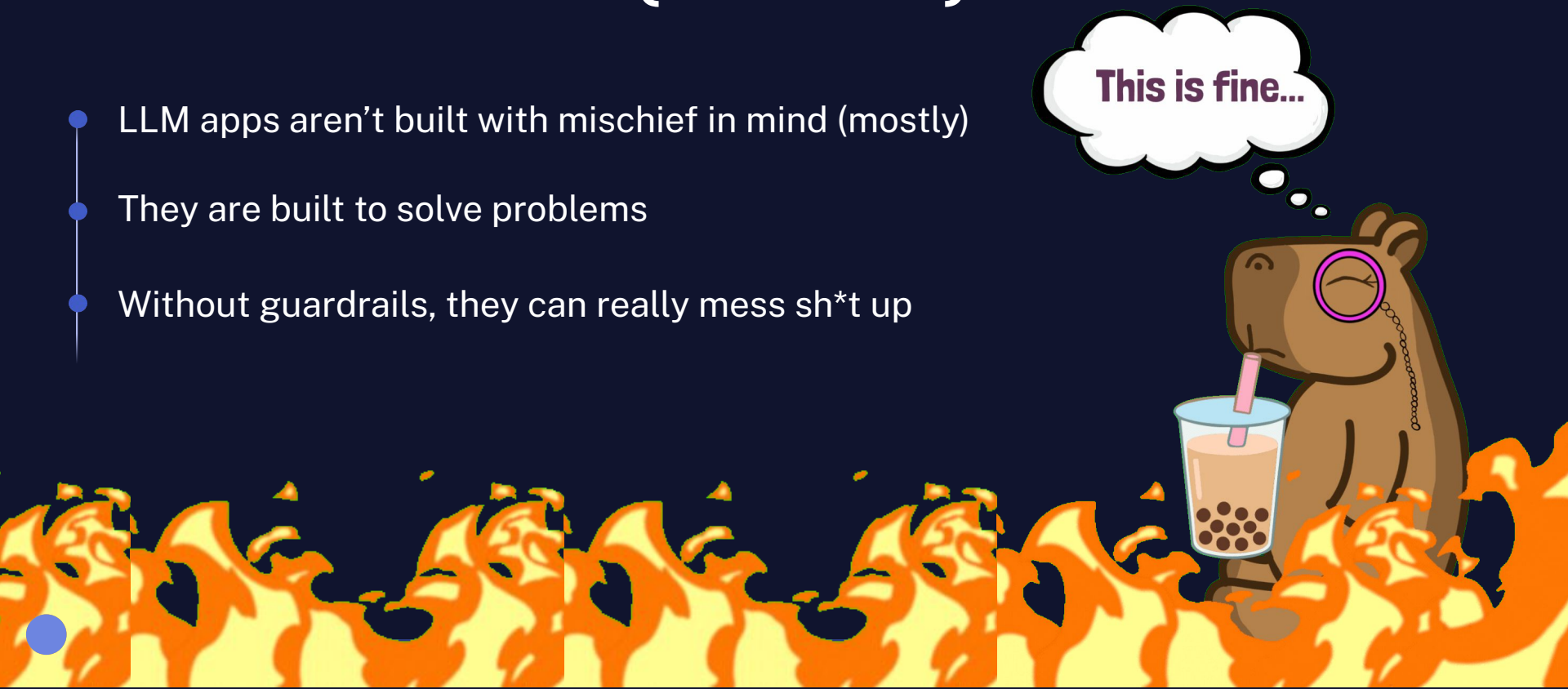
- Non-deterministic: run many times == different results each time
- Dynamic call graph: You don't know the path to be taken ahead of time
- Unpredictable costs: Token use and costs may vary
- Opaque decisions: No stack trace to help you understand WTF happened



OOOPS, I DID IT AGAIN...

Good intentions with (sometimes) bad outcomes

- LLM apps aren't built with mischief in mind (mostly)
- They are built to solve problems
- Without guardrails, they can really mess sh*t up



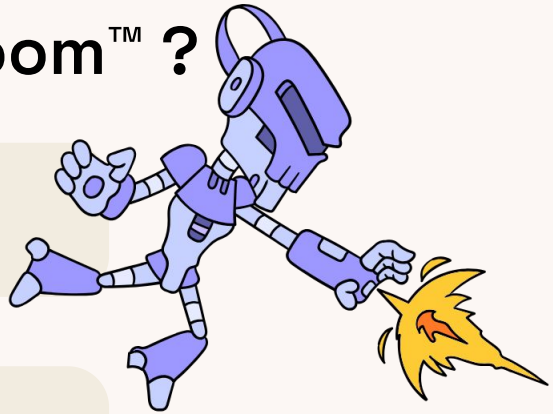
WHEN LLMs MESS UP

What led it down the Path of Doom™ ?

What did it decide?

What tools did it use?

Why OH WHY???





02

Instrumenting LLM applications

UP NEXT

Speaking the language of AI Observability

Agent instrumentation

LLM API instrumentation

Competing naming standards

DECODER RING

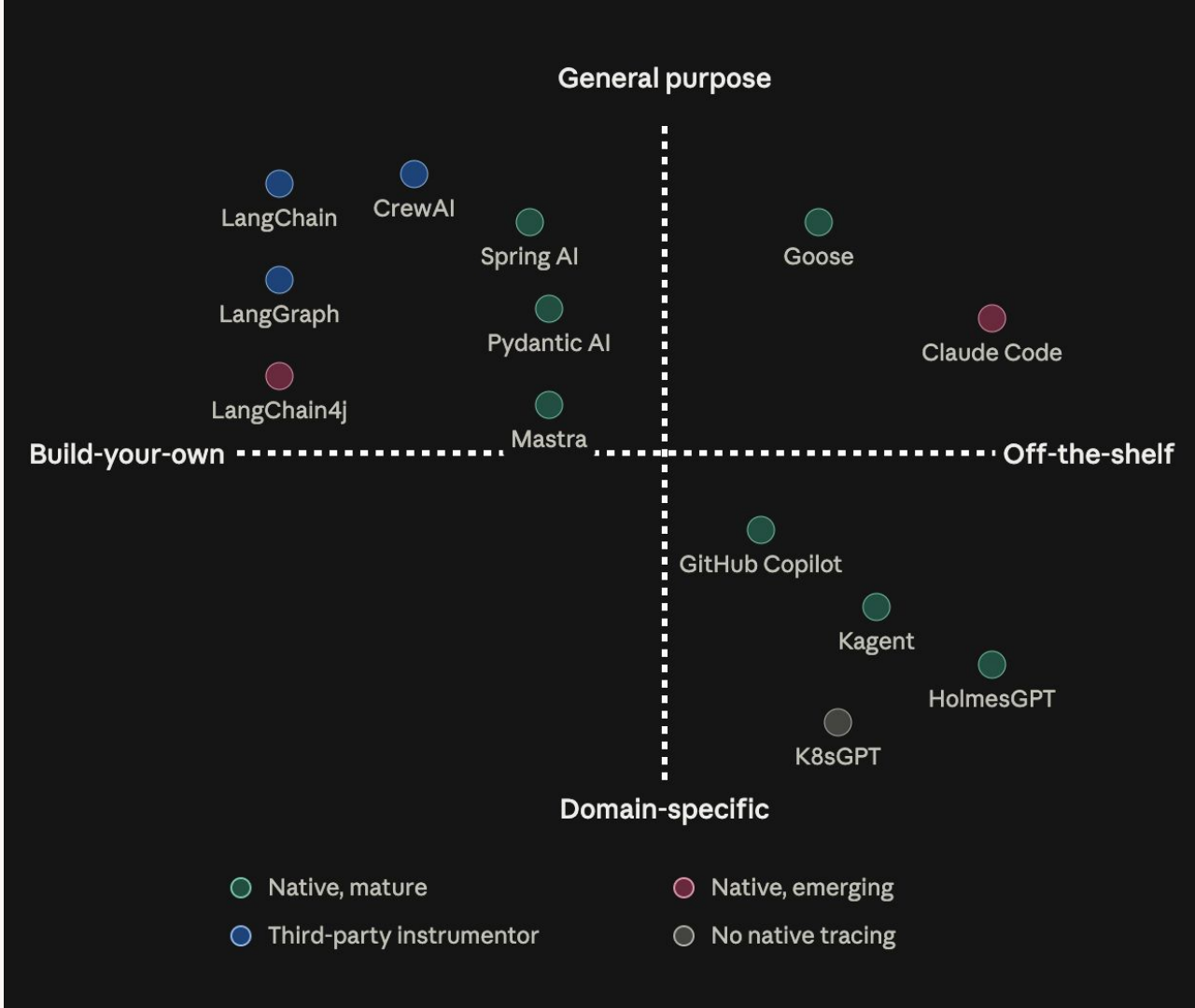
Speaking the language of AI observability



Traditional Observability	LLM Observability
Stack trace	Reasoning chain
Status code	Quality signal
RPS & latency	Tokens, cost, blast radius
Static call graph	Dynamic tool invocation
Replayable request	Non-deterministic run



Agent Frameworks vs Off-the-Shelf Agents



AGENT INSTRUMENTATION

Agent Frameworks vs Off-the Shelf Agents



AGENT FRAMEWORKS

Assemble your own agent

Instrumentation: 🫠

OFF-THE-SHELF AGENTS

Declarative (YAML)

Native instrumentation



BEHIND THE SCENES

LLM API calls





TOOLS LANDSCAPE

Instrumenting LLM API calls



OpenInference
Arize

OpenAI · Anthropic · Gemini · Groq · Mistral · Bedrock · Cohere · Vertex AI · any provider via LiteLLM



OpenLLMetry
Traceloop

OpenAI · Anthropic · Gemini · and other major providers, auto-detected from installed SDKs



OpenLit
OpenLit

OpenAI · Azure OpenAI · Anthropic · Gemini/Vertex AI · Cohere · Mistral · Groq · Bedrock · Ollama · HuggingFace · GPT4All · and more



GenAI OTel Instrumentation
OpenTelemetry

OpenAI · OpenRouter · CometAPI · Anthropic · Google · AWS · Azure · Cohere · Mistral · Together · Groq · Ollama · Replicate · SambaNova · and more



HOW IT WORKS

Instrumenting LLM API calls



Wrap SDK client

Intercept each provider API call

Capture span data

Prompt · completion · tokens · latency

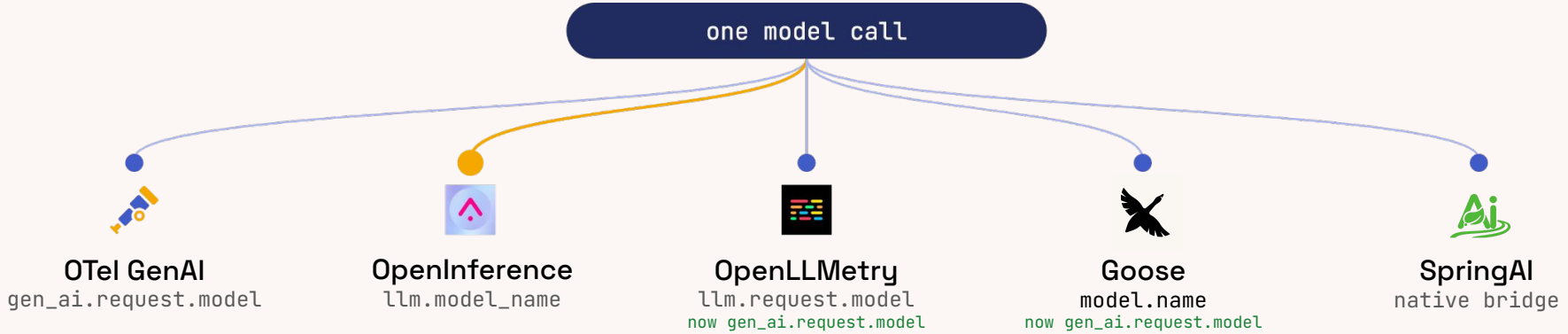
Export via OpenTelemetry

Span sent to your collector / backend



COMPETING STANDARDS

Why can’t we all just get along??

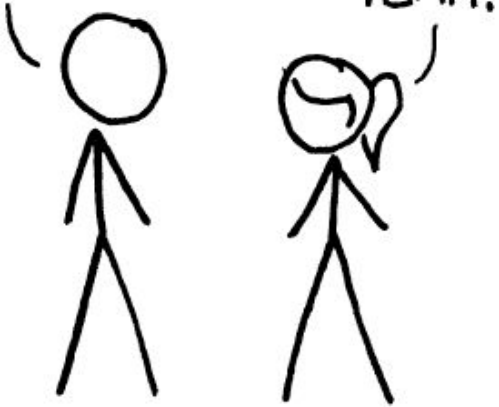



HOUSTON,
WE HAVE
A PROBLEM!
APOLLO 13

HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)

SITUATION:
THERE ARE
14 COMPETING
STANDARDS.

14?! RIDICULOUS!
WE NEED TO DEVELOP
ONE UNIVERSAL STANDARD
THAT COVERS EVERYONE'S
USE CASES.



SOON:

SITUATION:
THERE ARE
15 COMPETING
STANDARDS.

03

OpenTelemetry as standard for LLM apps

UP NEXT

Why OpenTelemetry?

We know what we need to capture

Semantic conventions to the rescue!

Converging on OpenTelemetry GenAI semconv

GenAI Normalizer Processor

LET’S NOT REINVENT THE WHEEL

Why OpenTelemetry?

- Vendor neutral
- Open source
- Already a standard



HALF THE WORK IS DONE, AT LEAST!

We already know what we need to capture



WHAT'S IN A NAME?

Semantic Conventions to the rescue!

- Shared vocabulary
- Standardized names
- A living specification

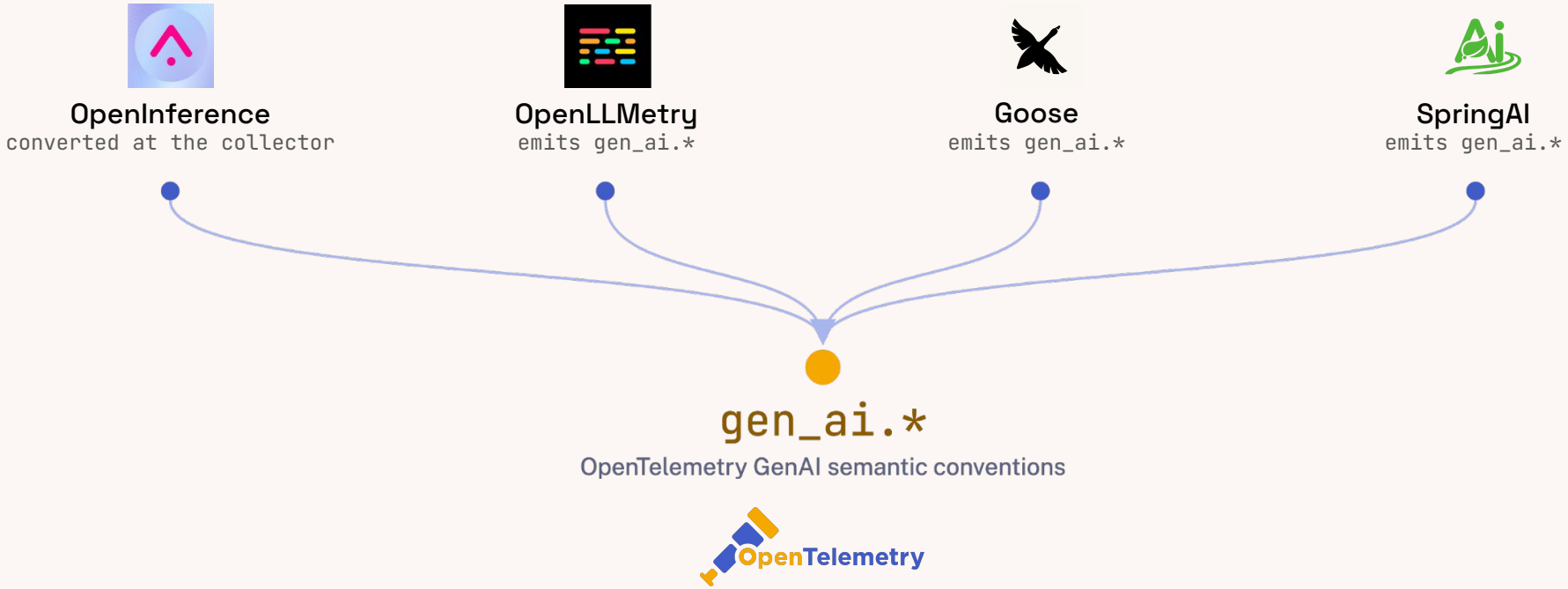


OTel GenAI Semantic Conventions	LLM Observability Data
gen_ai.usage.reasoning.output_tokens	Reasoning chain
gen_ai.evaluation.name, gen_ai.evaluation.score.value, gen_ai.evaluation.score.label, gen_ai.evaluation.explanation	Quality signal
gen_ai.usage.input_tokens, gen_ai.usage.output_tokens, gen_ai.token.type	Tokens, cost, blast radius
gen_ai.tool.name, gen_ai.tool.call.id, gen_ai.tool.call.arguments, gen_ai.tool.call.result, gen_ai.operation.name (value execute_tool)	Dynamic tool invocation
gen_ai.request.seed, gen_ai.request.temperature	Non-deterministic run



THERE'S GOOD NEWS!

Converging on OpenTelemetry GenAI semconv (mostly)



SIDE QUEST

A tale of two donations

OPENLLMETRY · TRACELOOP

Offered for donation in February 2025.

No traction. Closed sixteen months later

github.com/open-telemetry/community/issues/2571

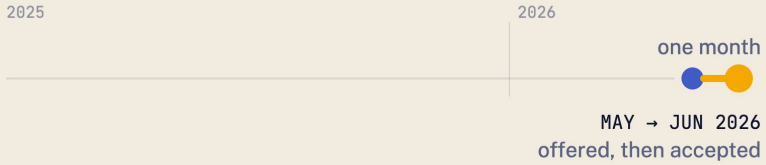


OPENINFERENCE · ARIZE

Offered for donation in May 2026.

Accepted by the governance committee in June 2026.

github.com/open-telemetry/community/issues/3467



ONE PROCESSOR TO UNIFY THEM ALL

Meet the GenAI Normalizer Processor



```
llm.model_name
llm.token_count.prompt
openinference.span.kind
```

→

gen_ai_normalizer
contrib >0.153.0

→



```
gen_ai.request.model
gen_ai.usage.input_tokens
gen_ai.operation.name
```

```yaml
processors:
  gen_ai_normalizer:
    sources:
      - name: openinference
        remove_originals: false
      - name: openllmetry
        remove_originals: false
```





EXAMPLE DATA FROM A PYTHON LLM APP USING OPENINFERENCE

GenAI Normalizer processor in action

OPENINFERENCE, IN

- llm.provider
- llm.model_name
- llm.token_count.prompt
- llm.token_count.completion
- openinference.span.kind
- llm.input_messages.*
- llm.output_messages.*
- llm.system



OTEL GENAI, OUT

- gen_ai.provider.name
- gen_ai.request.model
- gen_ai.usage.input_tokens
- gen_ai.usage.output_tokens
- gen_ai.operation.name
- gen_ai.input.messages
- gen_ai.output.messages
- llm.system

04

Demo time!

UP NEXT

Cast of characters

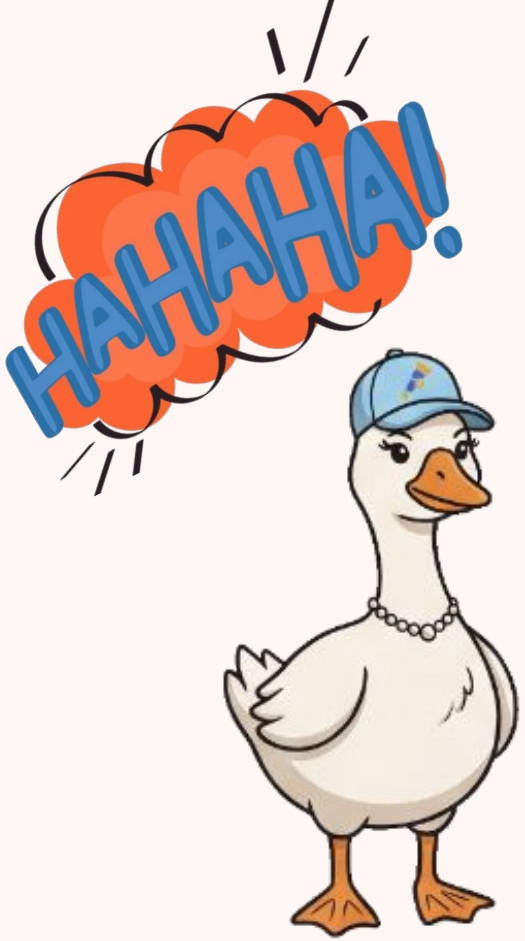
Picture this

Demo overview

GenAI data, three ways

PICTURE THIS

A rogue agent has deleted rows in the database!!



PICTURE THIS

Platform engineers have dispatched AI SRE tools to investigate the incident



Beaver SRE

Python · OpenInference



Otter SRE

Python · OpenLLMetry

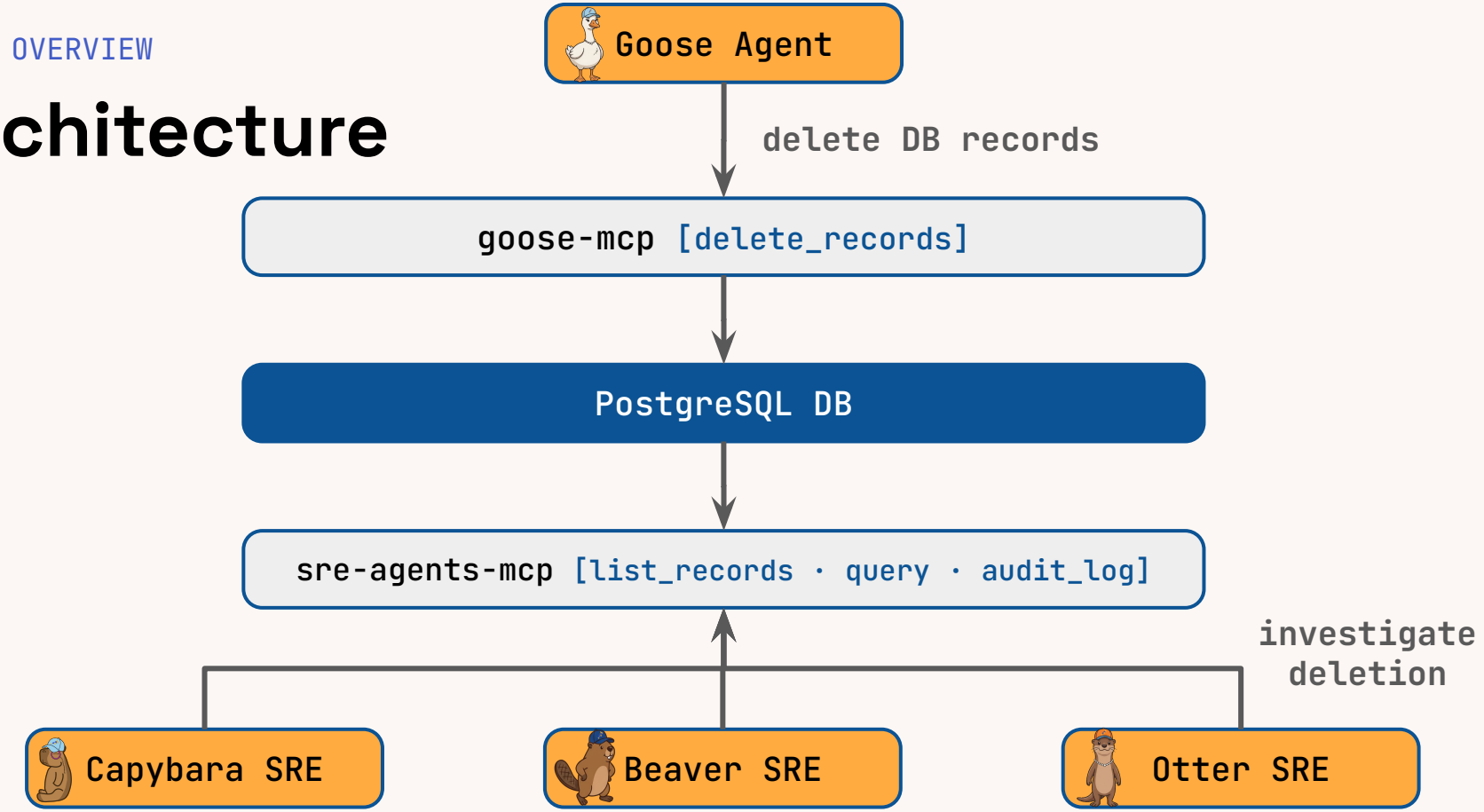


Capybara SRE

Java · langchain4j

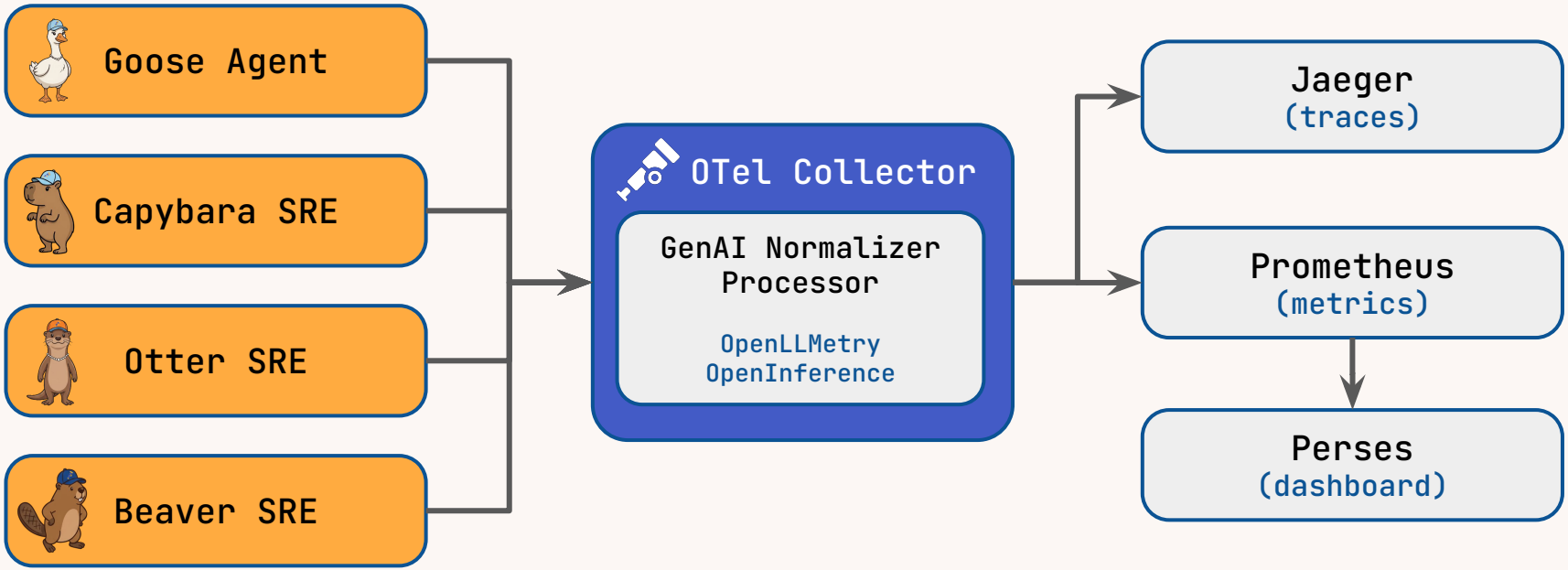
DEMO OVERVIEW

Architecture



DEMO OVERVIEW

Telemetry Collection



DEMO



UPON FURTHER EXAMINATION...

GenAI data, three ways

	capybara-sre	beaver-sre	otter-sre
the model	gen_ai.request.model	llm.model_name	gen_ai.request.model
the provider	gen_ai.provider.name	llm.provider	gen_ai.provider.name
what went in	gen_ai.prompt	llm.input_messages.*	gen_ai.input.messages
what came back	gen_ai.completion	llm.output_messages.*	gen_ai.output.messages
tokens in	gen_ai.usage.input_tokens	llm.token_count.prompt	gen_ai.usage.input_tokens
tokens out	gen_ai.usage.output_tokens	llm.token_count.completion	gen_ai.usage.output_tokens
the tools it had	-	llm.tools.N.tool.json_schema	gen_ai.tool.definitions



05

Forensic analysis

UP NEXT

The 63 millisecond clue

Some useful attributes

Three things you still cannot get

CAPYBARA SRE · INVESTIGATING THE INCIDENT

The 63 millisecond clue



BEAVER SRE · THE AUDIT_LOG TOOL CALL

The culprit hidden in the span!

gen_ai.tool.name	audit_log
gen_ai.tool.description	"…which client and database role did it"
gen_ai.tool.call.id	toolu_01TByoxFYgfPYdApe1X3gVXX
gen_ai.tool.call.arguments	{"limit": 20}

THE RESULT RETURNED FROM THE TOOL CALL:

```
AuditEntry[at=07:11:16, operation=DELETE, username=pepper, plan=free, client=goose (db
role: deploy_svc)]
AuditEntry[… username=nibbles …]
AuditEntry[… username=biscuit …]
```



Goose DID IT!



… the goosechase is over!

What did goose do? What was captured?



What did the telemetry tell us?

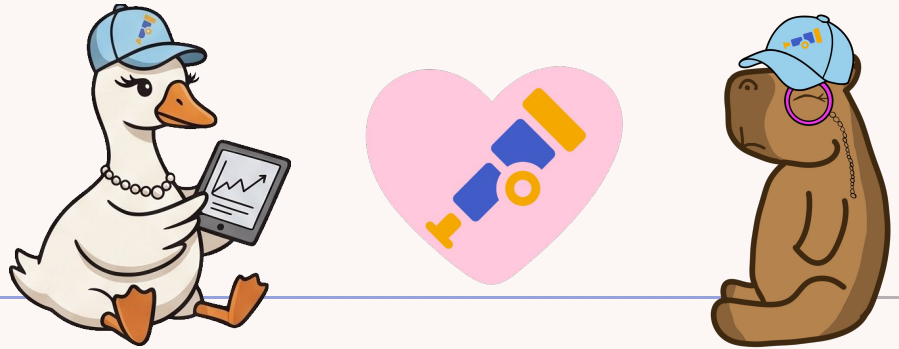
Turns		finish_reason
1	`reasoning` "Look at the records, delete free-plan accounts, report how many" `tool_call` list_records {}	`tool_call`
2	`text` "3 free-plan accounts. I'll proceed to delete them now." `tool_call` **delete_records {"plan":"free"}**	`tool_call`
3	`text` "Done. Removed 3 free-plan accounts, 2 remaining."	`stop`



FOOD FOR THOUGHT

Goose is the culprit, but you don't see the full picture until you see the Goose telemetry

Context matters! Connecting your telemetry lets you see the full picture



THE GEN_AI SEMANTIC CONVENTIONS

What can they answer today?

Question?	Attribute	Opt-in?
What kind of step?	`gen_ai.operation.name`	
Which model, whose?	`gen_ai.request.model` · `provider.name`	
What was it told?	`gen_ai.input.messages`	✓
What did it decide?	`gen_ai.output.messages`	✓
Act, or stop?	`gen_ai.response.finish_reasons`	
Which tool?	`gen_ai.tool.name` · `tool.call.id`	
Asking for what?	`gen_ai.tool.call.arguments`	✓
What actually happened?	`gen_ai.tool.call.result`	✓
Same conversation?	`gen_ai.conversation.id`	



LIMITING FACTORS

We can prove what, but not necessarily why

`gen_ai.tool.call.reason` doesn't exist (we wish it did)

The LLM doesn't share everything with us (HOW RUDE!)



HOW CAN YOU EVALUATE THE QUALITY OF THE AGENT LOOP?

Enter LLM-as-judge (aka just another model call)

IT READS

The incident prompt
Every tool call, in order
What the agent reported



CapybaraJudge

IT RETURNS

```
log record  event.name = gen_ai.evaluation.result
  gen_ai.evaluation.name          root_cause_corretness
  gen_ai.evaluation.score.value   1.0  // a metric
  gen_ai.evaluation.score.label   pass // a gate
  gen_ai.evaluation.explanation   "explanation…"
```

06

What did we learn?

UP NEXT

Becoming an AI observability wizard

PARTING WORDS OF WISDOM

Becoming an AI observability wizard

1. Instrument conventional application code
2. Instrument agents
3. Instrument LLM API calls
4. Follow gen_ai.* semantic conventions where possible
5. LLM-as-judge for validation
6. Propagate that (telemetry) context!



And remember: be a good open source citizen and contribute back to projects when you see a gap!



COME AND ARGUE WITH US ABOUT ATTRIBUTE NAMES

Cheers!



Adriana Villela

Principal Developer Advocate

dynatrace



Kasper Borg Nissen

Director of Developer Relations

dash0

Demo: `github.com/kaspernissen/your-agent-did-what`



TELL US WHAT YOU THINK!

Feedback



Your Agent Did What? Forensic Observability for Systems That Don't Leave Obvious Footprints