Context is the Architecture

Rethinking security platforms in the LLM era

Leon Goldberg · Sola Security · Open Lakehouse + AI, Mountain View · Nov 3, 2026

Leon Goldberg

CTO, Sola Security

Previously: Chief Architect, Cider Security (acquired by Palo Alto Networks)

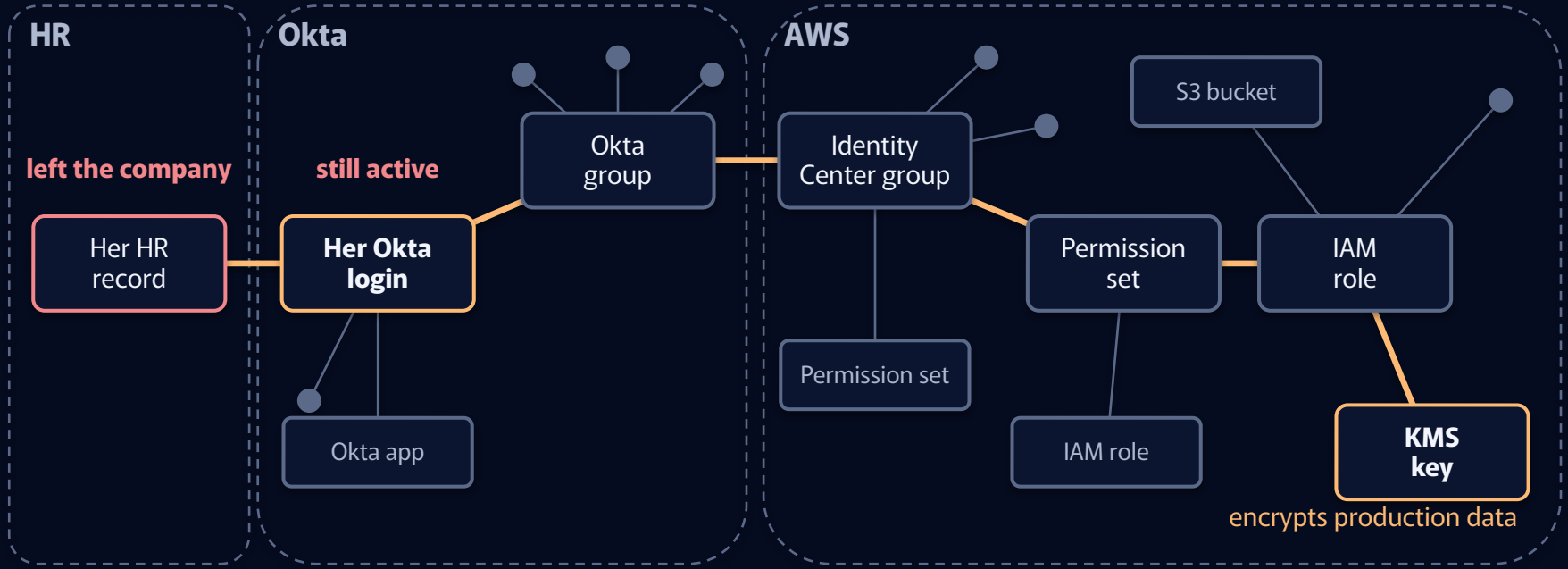


Sola is the enterprise security brain.

It connects the tools a company already runs into a living model of every identity, host, cloud resource, SaaS app and AI agent.

Every question gets an accurate, complete answer, backed by evidence.

She left three weeks ago. Her access didn't.



HR, Okta, AWS. None of them sees this path.



" Defenders think in lists.
Attackers think in graphs.
As long as this is true, attackers win."

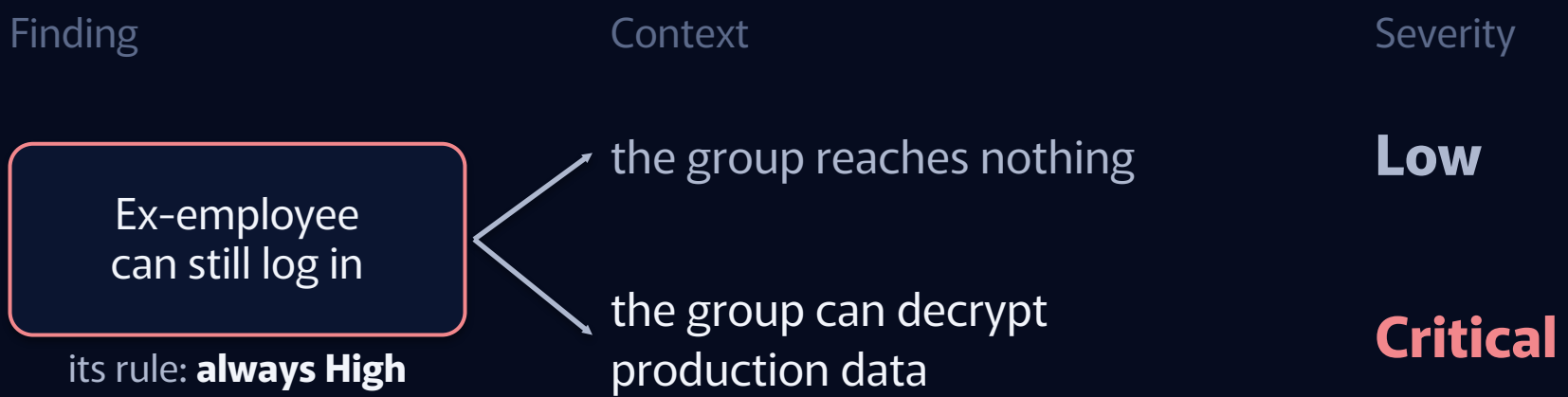
John Lambert, Microsoft, 2015

01

Why a graph

An identity tool flags her login. How bad is it?

Same finding, opposite severity



The environment, not the finding, determines how bad it is.

Security has been a graph problem for 20 years

2005	**MulVAL** attack graphs from Datalog rules
2016	**BloodHound** attack paths through Active Directory
Today	**Cloud and identity products** each ship their own graph

Attackers move in minutes, now with agents

29 min

average time to spread after a break-in
fastest: 27 seconds

CrowdStrike Global Threat Report 2026

80–90%

of one espionage campaign's hacking, done by an AI agent

Anthropic, November 2025

Breadth: every product. **Granularity**: every resource, every attribute.

We model the environment, not the policy

Policy-first



Model only what the rules read.
Context is what the rule author foresaw.

Environment-first



Model everything the company runs.

Before LLMs, a model of everything had no reader. Now it's required to keep pace with adversaries.

Joining at question time misses things

+34%

Correct answers
same model, same tasks, context added

−70%

Exploration queries
same study

Claude Code sampled 40 of ~5,000 buckets: “no bucket policies.”
65 granted read to anyone.

So we build the graph ahead of time.

arXiv 2606.02674 (correctness, queries) · arXiv 2609.30345 (buckets)

02

Why a lake

We built on graph databases at Cider and Palo Alto. Ours lives in Iceberg.

Graph databases buy speed with memory

They're built for fast lookups, and that speed is paid for in memory and compute.

Hops are random reads
Each hop is a lookup, fast only from memory.

Every copy pays again
Each replica needs the same memory again.

Every write maintains the structure
A live graph updates links and indexes, edge by edge.

Per GB, per month

~$3.80
kept hot in memory, per copy

~$0.03
in S3 Tables

Rough estimate, AWS list prices: x2gd.16xlarge (1 TiB, $5.344/h) = $3.81/GiB-month; S3 Tables $0.0265/GB-month

Analytic formats lift most of that burden

Keep the graph in Iceberg; query it in place with PuppyGraph.

	Graph database	Iceberg + PuppyGraph
Storage	in every server	one columnar copy on S3, shared with SQL
Memory	holds the hot graph	caches files, in RAM and on local disk
A hop	lookups, node by node	vectorized, in parallel across workers
Writes	record by record, updating links	whole files in parallel; drops are metadata-only

We trade millisecond lookups for a graph that grows at storage prices.

The graph is just Iceberg tables

One table per node type and per edge type, on S3 Tables

```
okta_user/                                  node
member_of/                                  edge: src_id, dst_id
okta_group/                                 node
  ___tenant_id___=acme/                     each customer, a partition
    ___revision___=2026-10-27T11:00/        each run, a partition
```

Storage, compaction and retention are plain Iceberg.

PuppyGraph is a graph query engine

Reads the Iceberg tables in place: no ETL, no second copy.

```
MATCH (u:okta_user)
  -[:MEMBER_OF]→(g:okta_group)
  -[:MAPS_TO]→(a)
RETURN u, a
```

1. **Map**
   MEMBER_OF reads the member_of/ table
2. **Prune**
   pushdown and min-max stats skip files
3. **Scan**
   only needed columns, vectorized
4. **Run**
   in parallel; a hub's edges split across workers

Our precomputed path queries: **0.45 s** median, **1.5 s** p95

A new run is just new files: the next query reads them.

PuppyGraph docs and blog; Data Engineering Podcast #510. Latency: Sola prod, 15,105 path queries, 3 days.

Right data, right store, right time



The lake holds everything. Other stores are specialists.



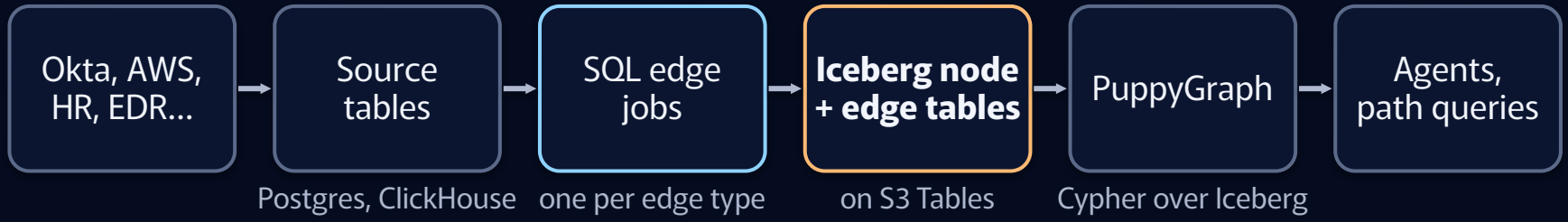
03

Building it

How tables become edges, and edges become paths

It all starts as tables

~1,400 kinds of source tables, from dozens of products



The graph is derived from these tables, and it stays tables.

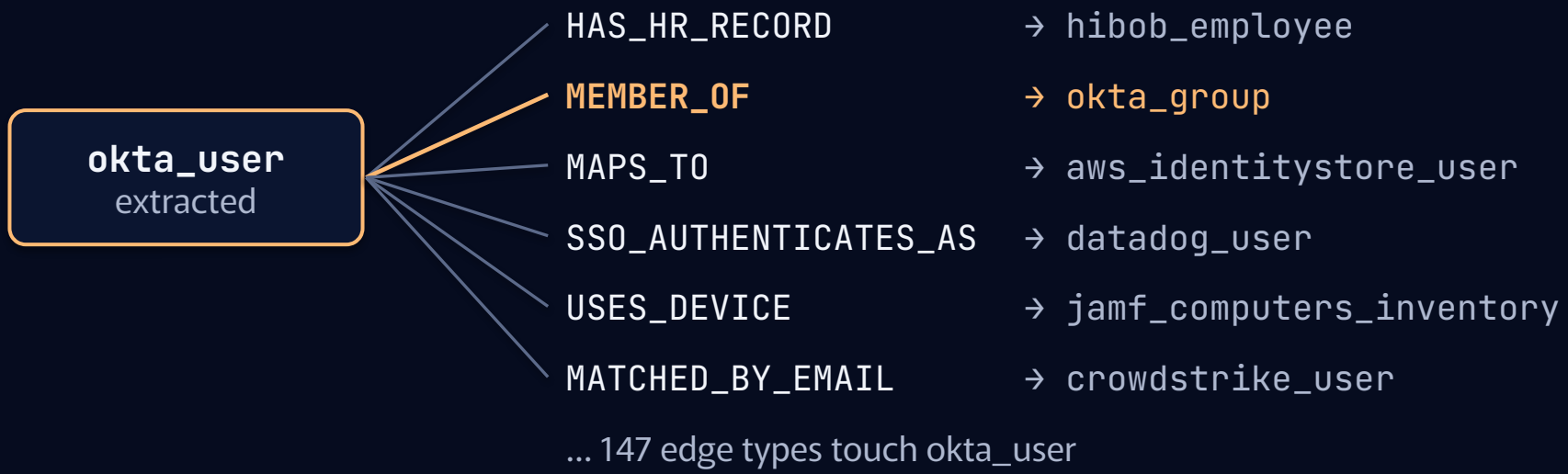
Every edge type is a SQL query

```
okta_group  -[MAPS_TO]→  aws_identitystore_group
SELECT o.name AS src, a.name AS dst
FROM okta_group o
JOIN aws_identitystore_group a
  ON LOWER(o.name) = LOWER(a.name)
```

No golden record: the match is just another edge.

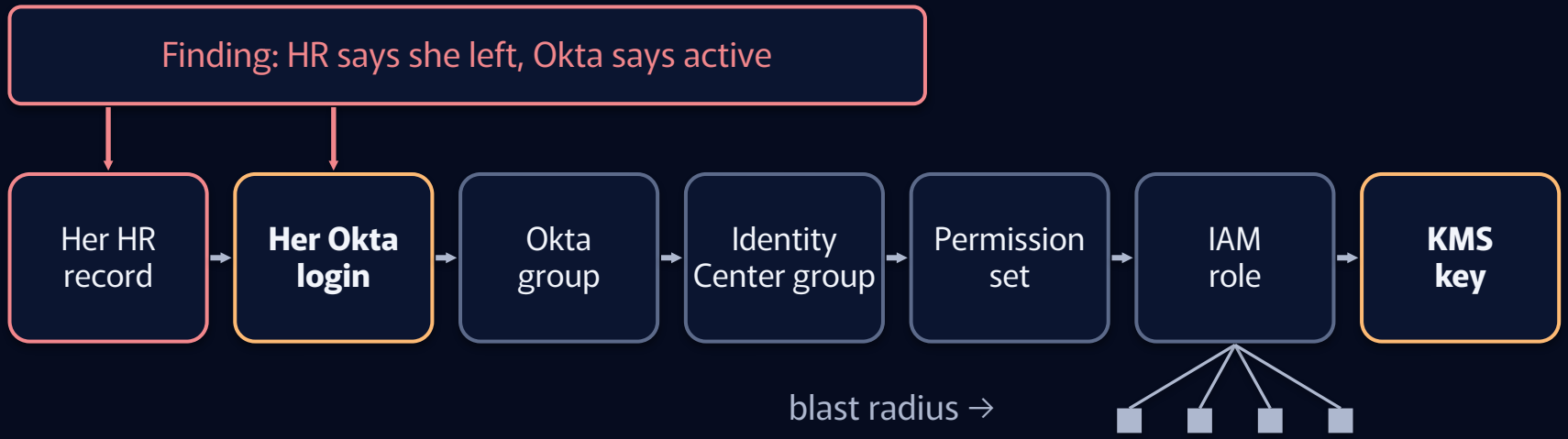
Graph ETL follows each resource's adjacency

500M+ edges a day



Unit of work: one resource type and its adjacency.

Paths are computed before anyone asks



Computed in bulk on Iceberg as data lands, then stored in the context graph.

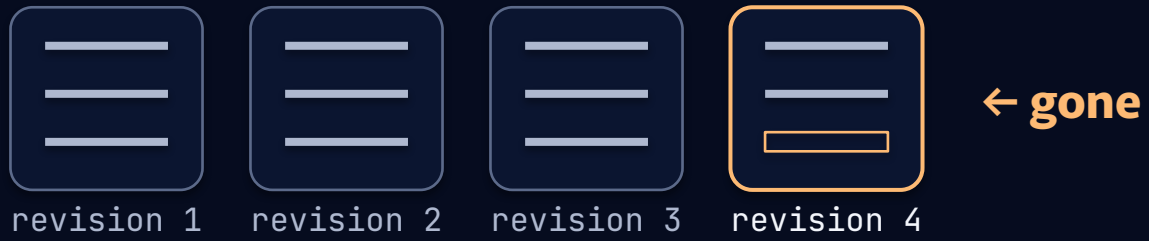
04

Keeping it correct

Configuration, events and content, each written its own way

Configuration: append a full revision

Users, groups, roles, keys



Arrives	a full listing, every run
We write	a new partition, published when complete
Old data	dropped with its revision

Missing from the newest revision means deleted.

Events: append each batch

Alerts, command results



Arrives	a stream, never edited
We write	a new partition per batch
Old data	aged out by day

An event never changes, so its edges are joined once.

Content: replace the record

Findings, tickets, files



one finding, three versions

Arrives	only what changed
We write	the new version of each record
Old data	deleted by key, in the same commit

Iceberg gives two ways to do it.

Merge-on-read or copy-on-write?

Who pays for a change: the writer, or every reader

	Merge-on-read	Copy-on-write
A change writes	a delete file and the new row	new copies of the files it touches
A read pays	merging deletes, until compaction	nothing extra, in any engine
Pick it when	changes are frequent and scattered	reads dominate, changes are rare

We choose per table, by how it changes and how it's read.

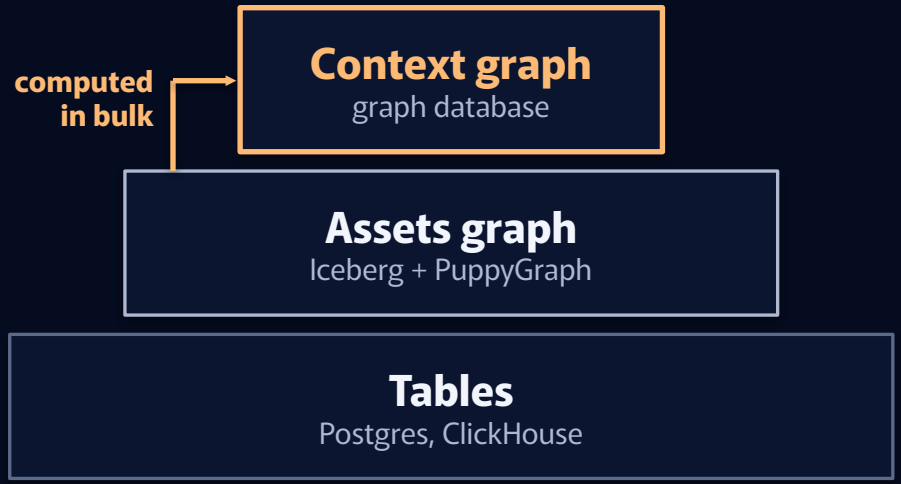
05

Using it

How agents use the graph

Answers move up and down the pyramid

One question: how bad is her login?



The risky subset: attack paths, blast radius
↳ her path to the KMS key

Every resource and edge, for reach
↳ what else her IAM role reaches

All the raw data: counts, attributes, records
↳ the day HR says she left

One answer can draw on all three.

For new questions, the agent writes the query

Scope

Code pins every query to published runs.

```
MATCH (g)-[e]→(a)
WHERE g.revision IN $pub
  AND e.src_revision IN $pub
```

Discover

The agent asks for schema as it goes.

```
types: okta_user, okta_group …
okta_group edges:
  ← MEMBER_OF   okta_user
```

Refine

Too big for the window? Cut, and the agent narrows.

```
RETURN b            → cut
RETURN count(b)     → 4,812
WHERE b.public      → 12 rows
```

The model writes the query; code decides what it sees.



06

Results

Back to the question: how bad is it?

With context, 3 in 4 HIGH findings changed grade

Vendor HIGH findings → Grouped by situation → **One agent per group**



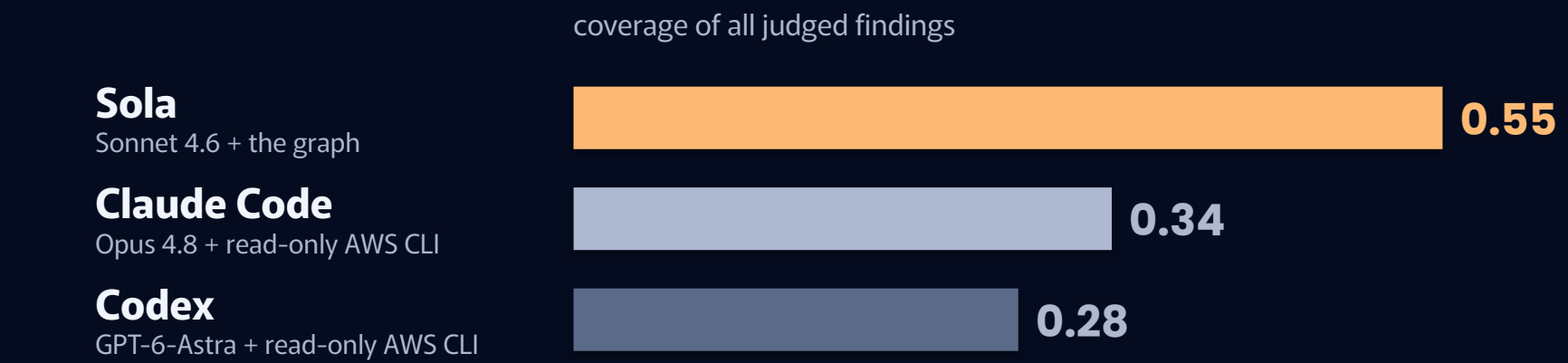
62 of 138 vendor rules went down on some resources, up on others.

Checked read-only against live infrastructure: **99.4% held.**

Sola, 2026: 9,967 HIGH findings from a CNAPP and a CWPP, 8 environments; 4,140 of 4,164 live checks

Coding agents aren't enough

28 cloud investigations, one live AWS environment

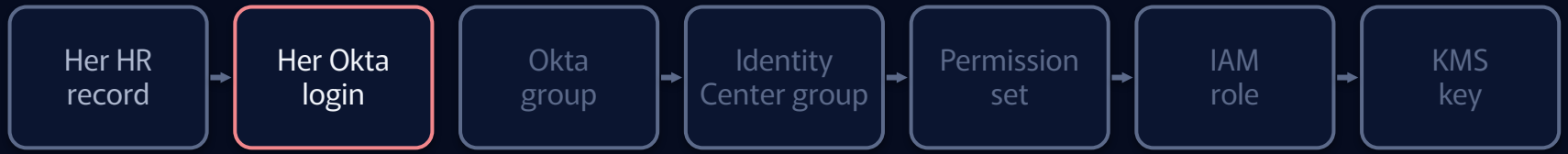


Won **24 of 28** tasks, at **1/18** of Claude Code's cost per task.

A smaller model with the graph beat a bigger one without it.

arXiv 2609.30345 (Sola, 2026). Cost per task: $0.0073 vs $0.1297 for Claude Code.

Context is the Architecture



Her path topped the queue. Her login was turned off.

1. Model the environment, not the policy.
2. Store the graph as tables; query it as a graph.
3. Write each kind of data the way it changes.
4. Do the heavy graph work before the question.

Graph databases at scale: the sources

Speed is memory

Neptune: "If the cache hit ratio is below 99.9% … consider upgrading the instance type." Neighbors are linked, not stored together (Stonebraker & Pavlo, 2024).

Sharded

JanusGraph: a network call "on each hop of a traversal" can largely nullify placement gains; vertex cuts are "not production-ready". NebulaGraph stores each edge twice. Natural graphs are power-law (PowerGraph, OSDI 2012).

One primary; replicas keep up

Neptune: "The primary writer instance coordinates all write operations … and is vertically scalable." Replicas smaller than the writer can't keep up and restart during bulk loads.

In the field

A large cloud-security vendor runs "hundreds of Neptune clusters … over more than 20 AWS Regions" (AWS case study). BloodHound moved customers to PostgreSQL "after over a year's effort"; post-processing >50% faster in many cases.

An edge is partitioned by its source's run

```
okta_group/
  ___revision___=2026-10-27T11:00/
aws_identitystore_group/
  ___revision___=2026-10-27T10:40/
maps_to/
  ___src_revision___=2026-10-27T11:00/
```

Why not partition by both runs?

Every pair of source and destination runs would need its own partition.

So the destination's run is checked when the query runs.

The write path: the numbers

Commit size

Edge jobs flush up to ~128 MB per commit with pyiceberg.

Concurrency

Jobs commit to the shared tables in parallel, optimistically.

Copy-on-write

A keyed change rewrites only the files that hold those keys, in one snapshot.

Delete files: the mechanics

Position deletes, written by us

One delete file per touched data file.

Why position, not equality

We pay the key lookup once a day at write time, instead of on every read.

Copy-on-write, by key

A change rewrites the files holding its keys. Any engine reads them correctly, with or without delete-file support.

Compaction

S3 Tables runs it, not us. Tables are Iceberg v2.

The read path: the details

One revision set

The guard pins every label to one set of published runs. Run ids are unique, so that is safe.

Unknown labels

Refused before they run; the engine would answer them with zero rows.

Result caps

A query returns at most 200 rows; the tool flags the cut.

Template reruns

Templates rerun as their inputs land, at most once a day.

Why not a sharded graph database?

We have built on graph databases before

At Cider and Palo Alto Networks. At Sola we started on Iceberg and PuppyGraph.

We still run a graph database

A small, curated graph that agents search, where a graph database's indexes win.

Update in place costs a sweep

Removing what a source stopped listing means stamping every row and sweeping the rest.

The lake gives versions

A run is a version, published by a row in Postgres and retired by a partition drop.

PuppyGraph in our deployment

A label per table

One label per node table and one per edge table.

The agent writes Cypher

Results are capped at 200 rows; a mistyped label is refused before it runs.

Dialect traps

No count(DISTINCT); no mixed CASE inside aggregates.

The agent loop in detail

Routing

SQL for totals; precomputed results for risk and reach; the lake graph for other relationships.

Precomputed: flagged paths only

“How many” and “riskiest” then widen to the full population in SQL.

Evaluation

Real questions, graded on precision and grounding.

History: runs, not snapshots

What is kept

Full-state runs for about a day. Events for 7 days, evolving entities for 365 (rolling out).

Not Iceberg time travel

Snapshots expire after an hour; runs are kept longer.