



Agentic AI Foundation

AGNTCon + MCPCon North America

Where the **agentic stack** is being built.

Building an Agentic Eval Pipeline

Battle-tested lessons with EvalBench



Kurtis Van Gent

kvg@google.com



Haoyu Wang

whaoyu@google.com



Haoyu Wang
Staff Software Engineer
Google Cloud Databases



Kurtis Van Gent
Senior Staff Software Engineer
Google Cloud Databases

MCP Core Maintainer
Transport WG Lead

Agenda

Why CUJ-based evals	00
What a great CUJ eval needs	01
Reliable eval infrastructure	02
What evals taught us	03
Wrap-up & Q&A	04

Managed MCP servers for Google Cloud databases



Fully managed

No infrastructure to provision. Google handles hosting, scaling, and security.



Agent-first design

Built specifically for the reasoning patterns of modern LLMs.



Common governance for enterprise data

Managed service providing observability and governance required for enterprises.



Simplified discovery of data sources

Easily browse and access data sources through Gemini CLI and IDEs.



Data protection

Model Armor provides secure access management and identity control.

MCP Toolbox for Databases

Open Source & Extensible

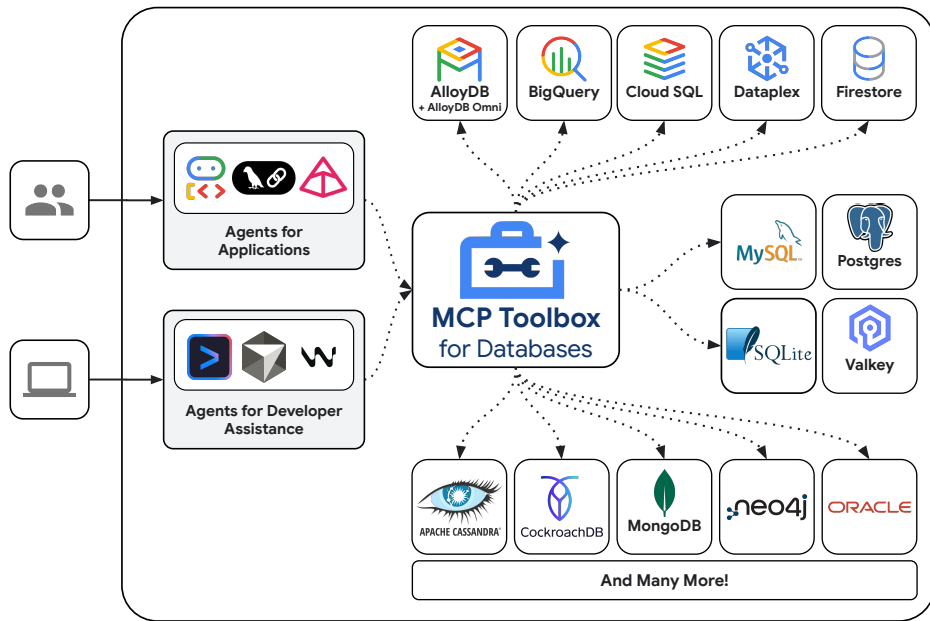
- ★ 16k+ Stars on GitHub, +180 contributors
- 🌐 40+ different databases

Highly Customizable

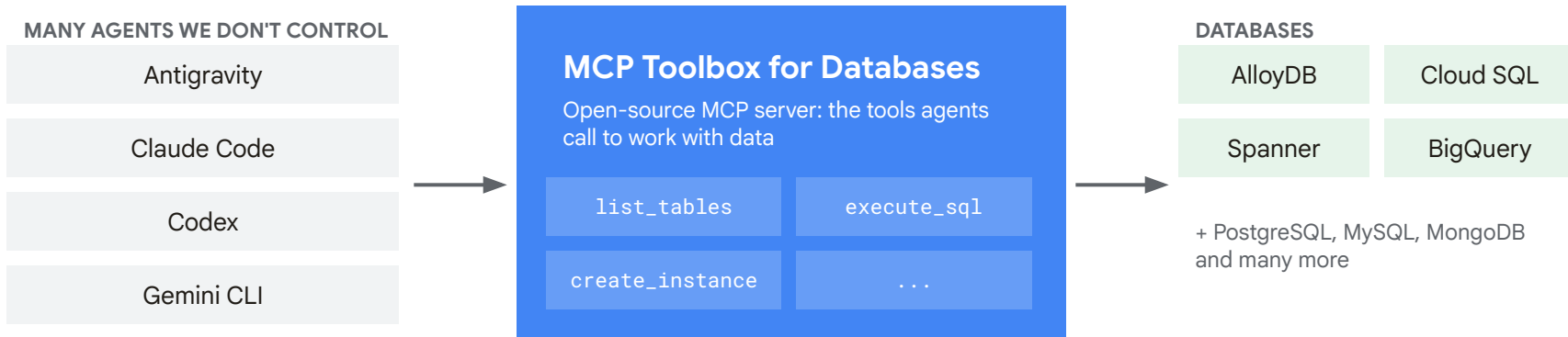
- Pre-built tools for common operations
- Custom tools for specific, narrow access for production-grade applications.**

Scalable and Secure

- Secure connection management & credential handling
- Built-in connection pooling and OpenTelemetry support



We built MCP Tools & Skills; Agents decide how to use it.



WHAT WE CONTROL: THE TOOLS

Unit and integration tests prove every tool works: right query, right result.

WHAT WE DON'T: THE AGENT

Which tool / skill it picks, in what order, and whether the user's job actually gets done.

So: To know if our tools really work for users, we have to evaluate whole user journeys.

Green tests prove the code works, not that the agent uses it

THE TEST PASSED

✓ Unit tests on the tool code are 100% green.

but →

THE AGENT FAILED

✗ The agent ignored the tool entirely and guessed the answer.

✓ Final string output is an exact match.

but →

✗ The agent wasted 30 seconds searching for irrelevant info before generating output.

✓ 5 manual demo runs looked great.

but →

✗ Run #6 failed silently in an edge case.

Why we focus on CUJ-based evaluation

None of these failures broke the build. Regressions can hide in your **tools**, your **skills**, or **the agent itself**, and only evals that measure the whole journey will catch them.

WHAT IS A CUJ?

A Critical User Journey: an end-to-end task a real user needs done, in their words.

"Create a Postgres instance"

"Load my orders data"

"Show my top customers"

"Clean up when I'm done"

It mirrors real usage

Users bring goals, not single tool calls. The journey is the unit of value.

It works across any agent

The same CUJ runs on every harness, so results compare apples to apples.

It catches hidden regressions

A tool, skill, prompt or model change shows up as a broken journey.

It measures what matters

Goal completion, cost and latency per journey, not per call.

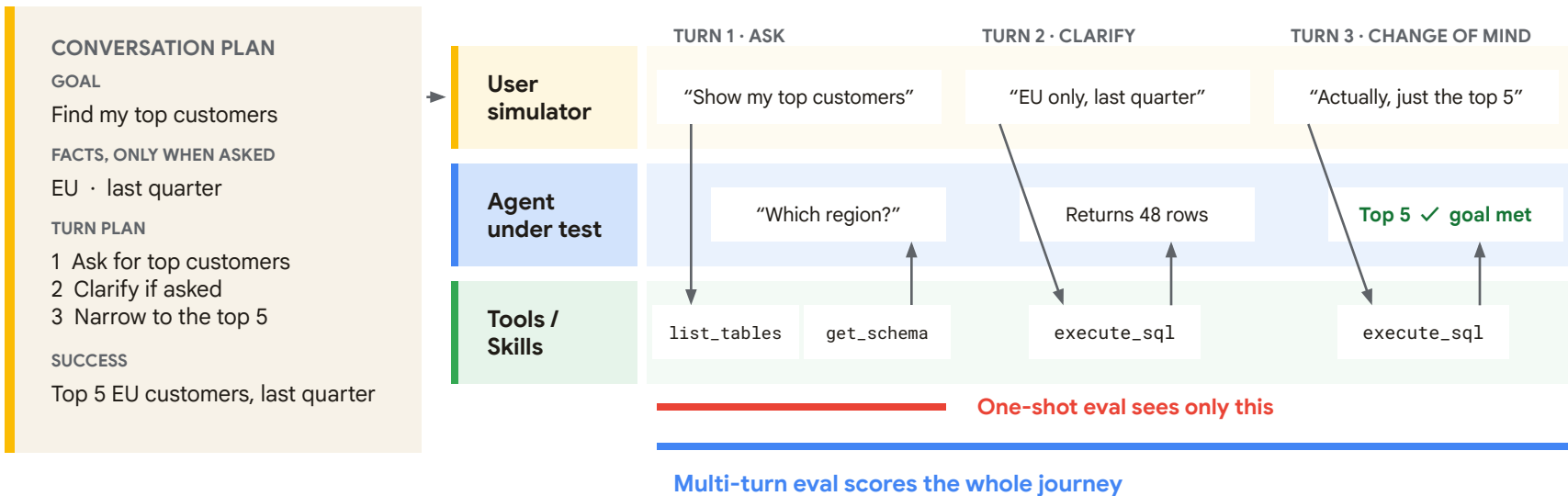
Takeaway: Tool tests tell you a tool works. CUJ evals tell you the user's job got done.

01

What a great CUJ eval needs

- 1 Multi-turn journeys, driven by a user simulator
- 2 Grade the whole run: path, outcome, output
- 3 Judges you can trust

Multi-turn journeys need a user simulator driven by a plan



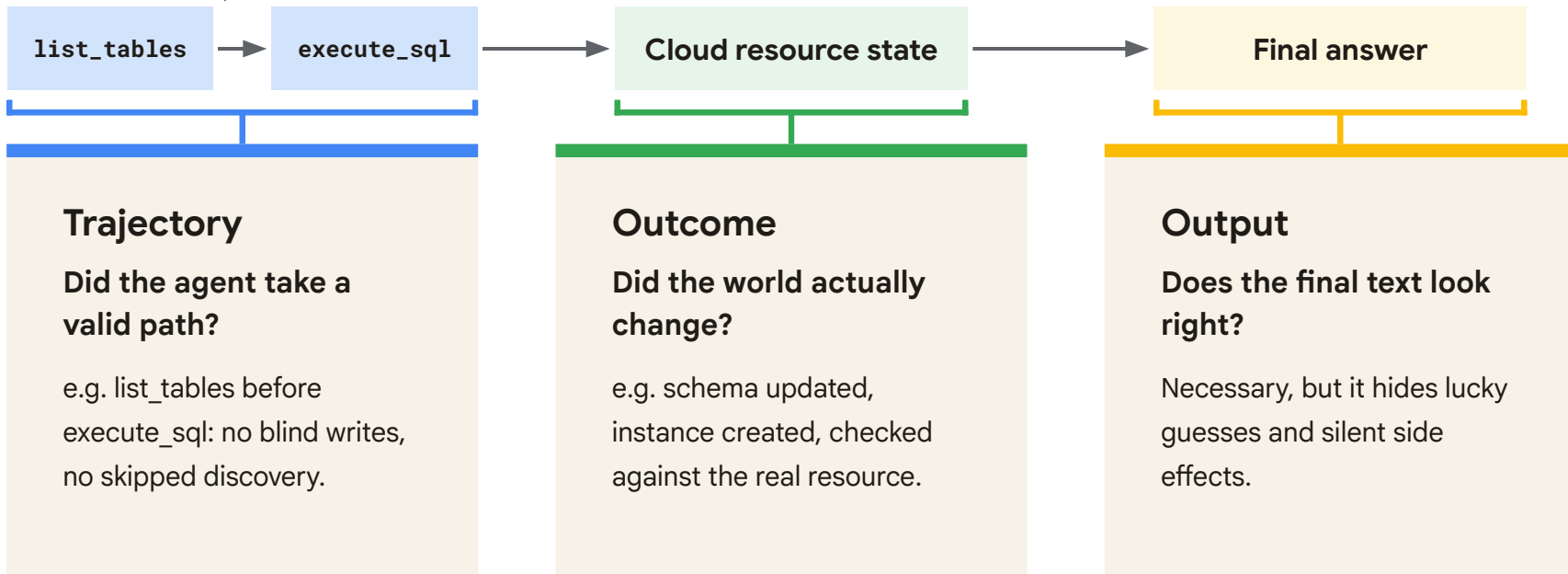
THE USER SIMULATOR

An LLM that plays the user. It follows the plan, reveals facts only when asked, and reacts to what the agent actually says. The agent never sees the plan.

- ✓ Repeatable every run
- ✓ Adapts to the agent
- ✓ Success criteria feed scoring

Grade the path and the outcome, not just the final sentence

ONE AGENT RUN, LEFT TO RIGHT



An agent can claim success without achieving it.
Check what it answered, how it got there, and what actually changed.

Calibrate LLM judges and make them show their work

✓ LLM judges scale grading to subjective intent

✗ But unexplained or uncalibrated, they are another black box

Deterministic scorers

Execute the golden query

Compare result sets

Check strict API and tool-call correctness

+

LLM judges that show their reasoning

Grade subjective intent. Every score comes with its reasoning:

JUDGE VERDICT

Intent fulfilled: 2 / 5

Why: filtered to region = 'EU' but ignored 'last quarter', so older rows leaked into the answer.

+

Calibrated against human raters

Expert-labeled examples

Judge scores the same set

Refine rubric & prompt

Measure agreement

ACTIONABLE FEEDBACK

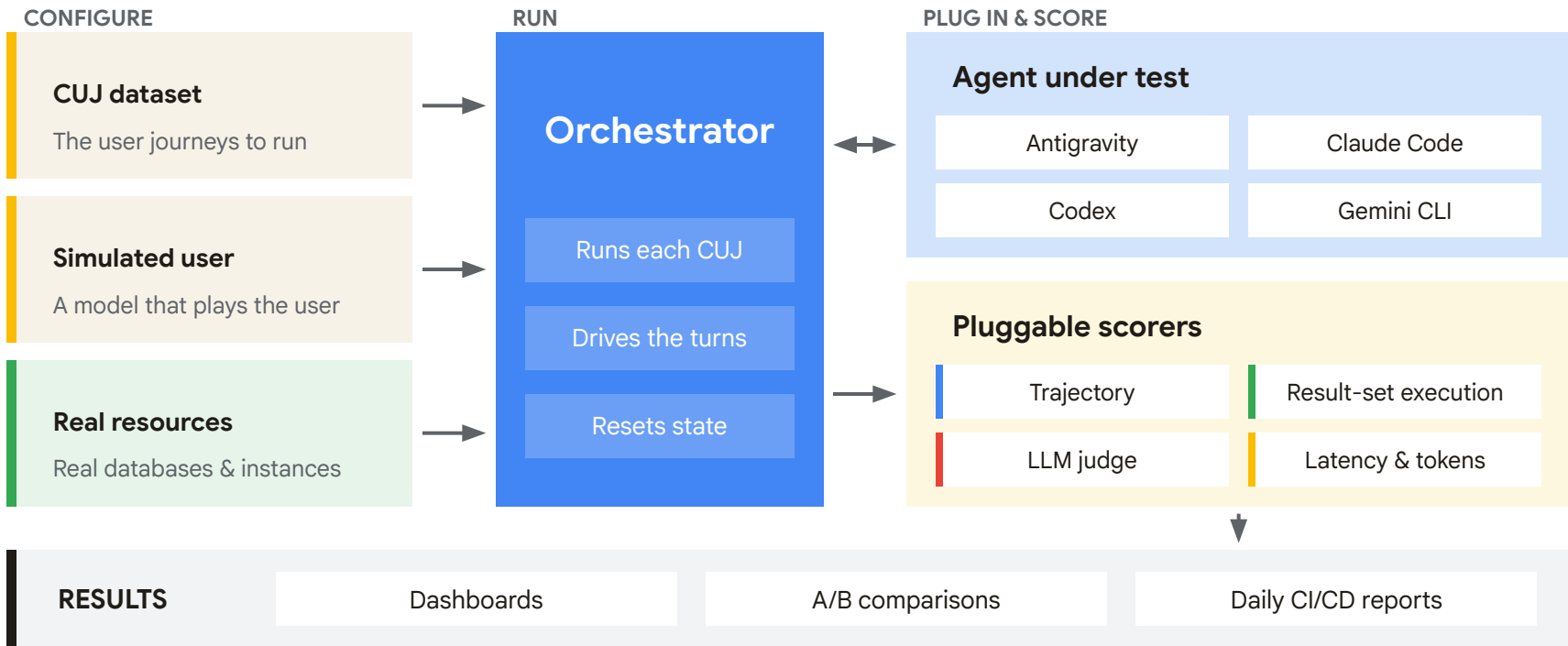
What failed (hard checks) + why it matters to the user (judge reasoning)

02

Reliable eval infrastructure

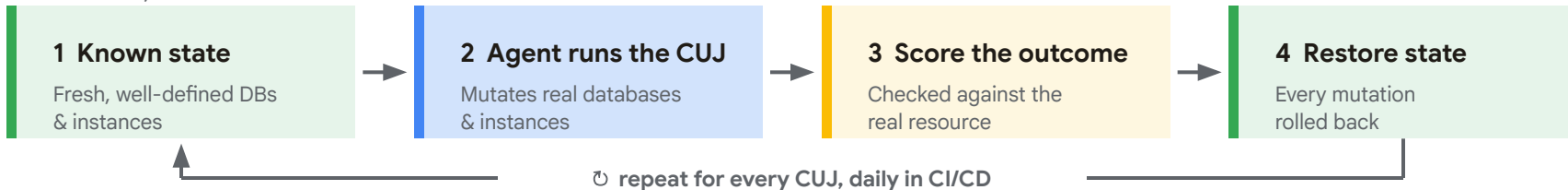
-
- 1 A harness built from swappable parts
 - 2 Real resources, clean state every run
 - 3 Daily in CI/CD, across any agent
-

A great eval harness is built from swappable parts



Evals hit real resources and reset to a known state

EVERY RUN, EVERY DAY



Daily CI/CD runs

Scheduled pipelines run the full CUJ suite every day, so regressions surface within a day of the change.

Real cloud resources

Agents act on actual databases and instances, not mocks, so outcomes are verified on the real thing.

Clean state, every run

Mutating operations are rolled back to a well-defined state between runs, which keeps results reproducible.

AGENT-AGNOSTIC

Antigravity

Claude Code

Codex

Gemini CLI

CASE STUDIES

03

What evals taught us

Token cost: what loads upfront

Pure skill impact via ablation

Agents that don't know how to wait

Token cost is decided by what loads before any work

CASE A · TOOL SCHEMA LOADING

Preload every tool schema?

Eager: full catalog upfront

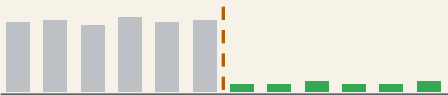


~20K tokens

Deferred: load on demand



< 1K tokens



Daily CI/CD catches the step when a release defers loading

→ Defer schemas; track tokens on every run

CASE B · PLUGIN SIZE

Bundle every skill into one plugin?

Load every description

+35%

context, every session

Cap the context budget

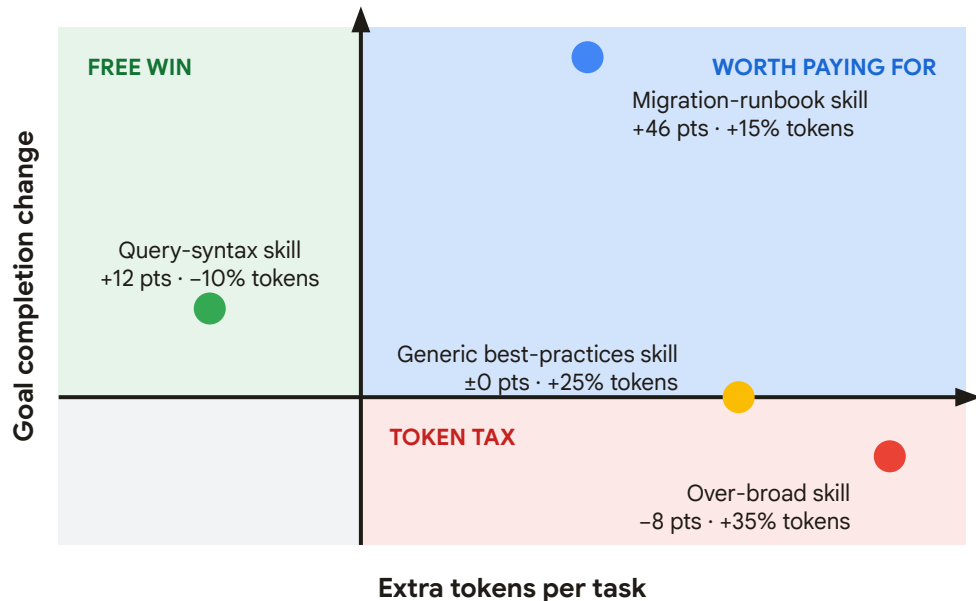
Silent drops

agent guesses tools by name

→ Size plugins along product boundaries

Takeaway: What loads upfront, not what gets used, drives cost. Measure it on every run.

Does a skill earn its tokens? Ablation tells you



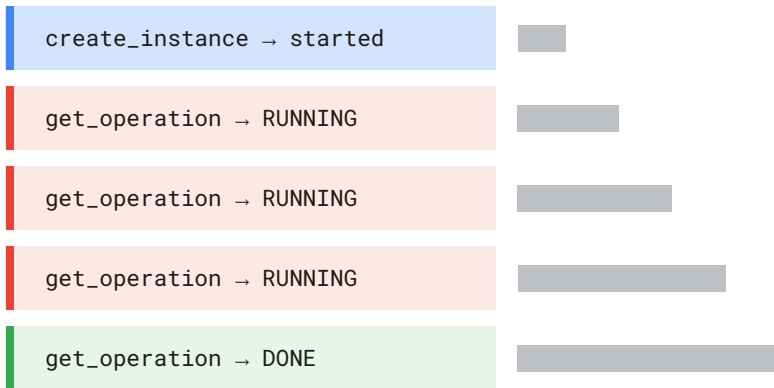
Ablate one skill at a time

- 1 Same agent, same model
- 2 Same CUJ dataset
- 3 Toggle one skill on / off
- 4 Compare completion & tokens

Takeaway: A skill that doesn't move goal completion is a token tax. Ablate it before you ship it.

Agents don't know how to wait, and polling burns tokens

WHAT NAIVE POLLING LOOKS LIKE



Every poll is a full agent step: the model re-reads the whole history just to see “not done yet.”

In-loop polling

anti-pattern

Each check is a model step. Tokens, latency and context grow with every poll.

Server-side blocking

trade-off

Fewer steps, but the tool holds the connection open and risks timeouts.

Harness-managed wait

recommended

The agent suspends; the harness resumes it when it's done.

✓ MCP Tasks are a good fit

The tool returns a task ID right away: call now, fetch later. The client tracks it, not the model.

Takeaway: Make long-running tools MCP Tasks. Evals expose polling as spikes in steps and tokens.

What a great agent eval needs



Real user journeys

Multi-turn CUJs with a simulated user



Grades the whole run

Trajectory, outcome and output



Judges you can trust

Deterministic checks + calibrated, explained LLM judges



Real resources, clean state

Every run starts from a known state



Any agent, swappable parts

Same CUJs across every harness



Runs every day

CI/CD tracks completion, tokens and latency

EvalBench does this work, and it's open source

github.com/GoogleCloudPlatform/evalbench

Run it locally with the CLI, with UVX, or in Docker

★ Star on GitHub

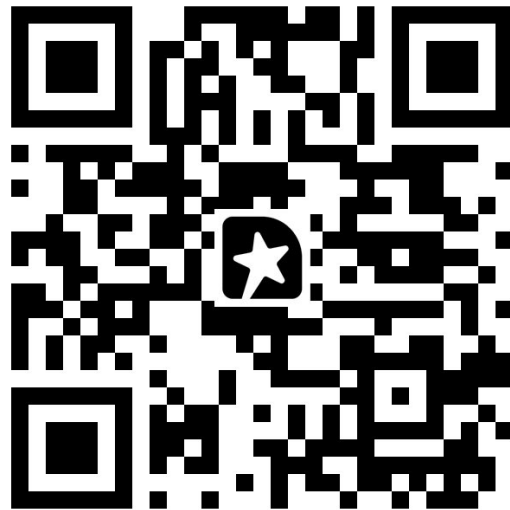
Thank you!

Questions?

Kurtis Van Gent (kvg@google.com)

Haoyu Wang (whaoyu@google.com)

github.com/GoogleCloudPlatform/evalbench



Building an Agentic Eval Pipeline: Battle-Tested Lessons with
EvalBench

Please rate this session