

How we taught Copilot to remember – and when to forget



Alejandro Carderera
Staff Applied Researcher
GitHub



Cooper Nederhood
Senior Applied Researcher
GitHub



Alejandro Carderera

STAFF APPLIED RESEARCHER/GITHUB

Memory / Copilot Code Review / Evaluations



Cooper Nederhood

SENIOR APPLIED RESEARCHER/GITHUB

Memory / Context / Evaluations

Session Agenda

01 The cold start problem
02 Modeling how developers work
03 Evaluating agent memory
04 What's next?
05 Q&A

The cold start problem

Remembering prior work

Ben is a dev working on a large codebase, he's on-call this week, and his days are packed! Fortunately, he has Copilot to help him with his work.

Monday

Quiet and productive

Progress on planned tasks.

Tuesday

Incident declared

Investigating and building critical context.

Wednesday

Issue resolved

Finding root cause and deploying fix.

Thursday

Back to planned work

Return to planned work.

Friday



A familiar incident, but no memory

A related incident occurs, but the agent remembers nothing.

The cold start problem

We work alongside agents as colleagues, but without the ability to remember past work, it's like a coworker whose memory is wiped every day! They have no recollection of:

Past incidents

No recollection of what caused the incident that the agent investigated alongside Ben

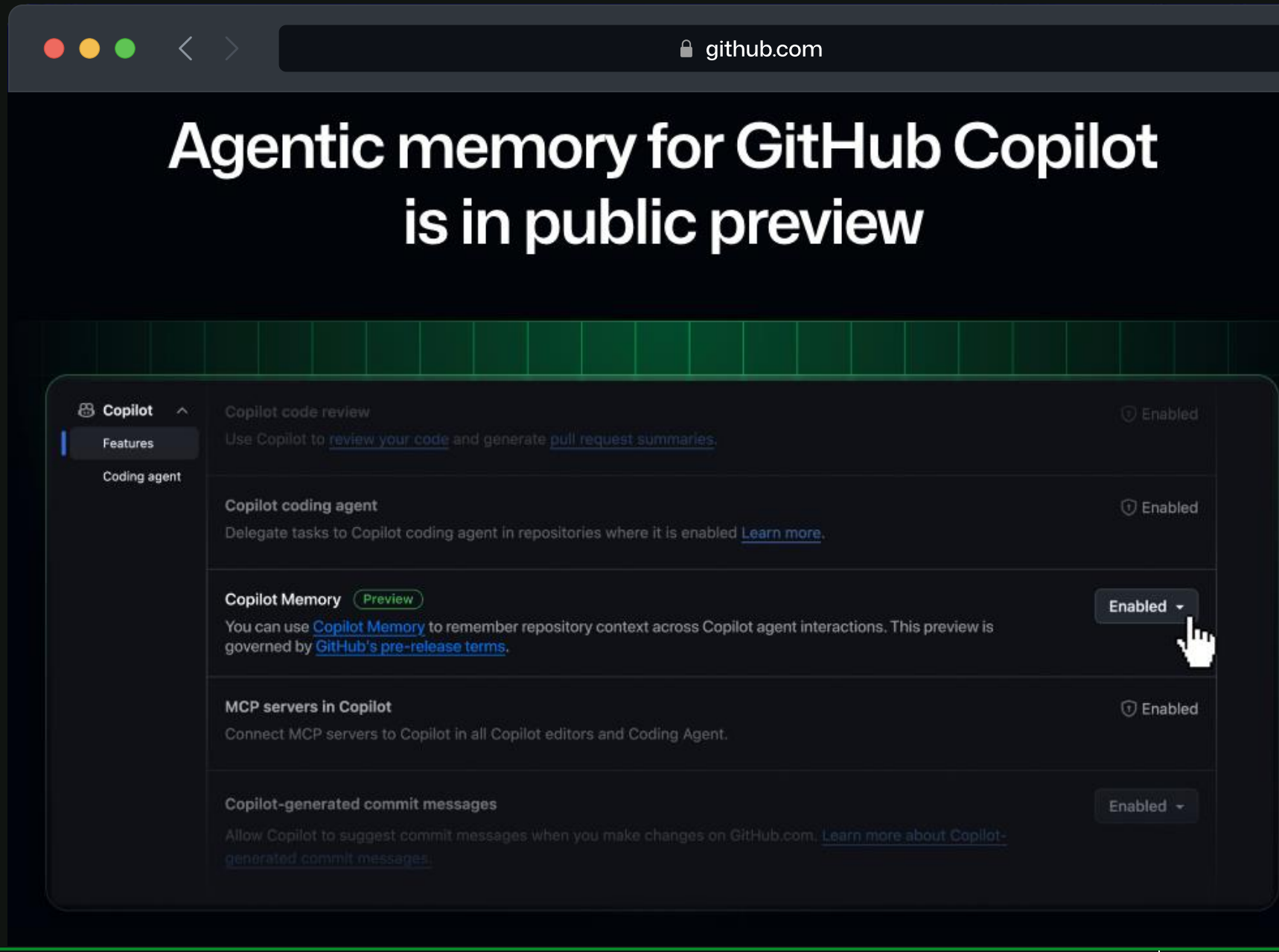
Past workflows you've used

No recollection of the data analysis that Ben did to measure latency improvements, and the key levers that improved it

Your user preferences

No recollection that Ben hates em-dashes and likes direct and succinct language

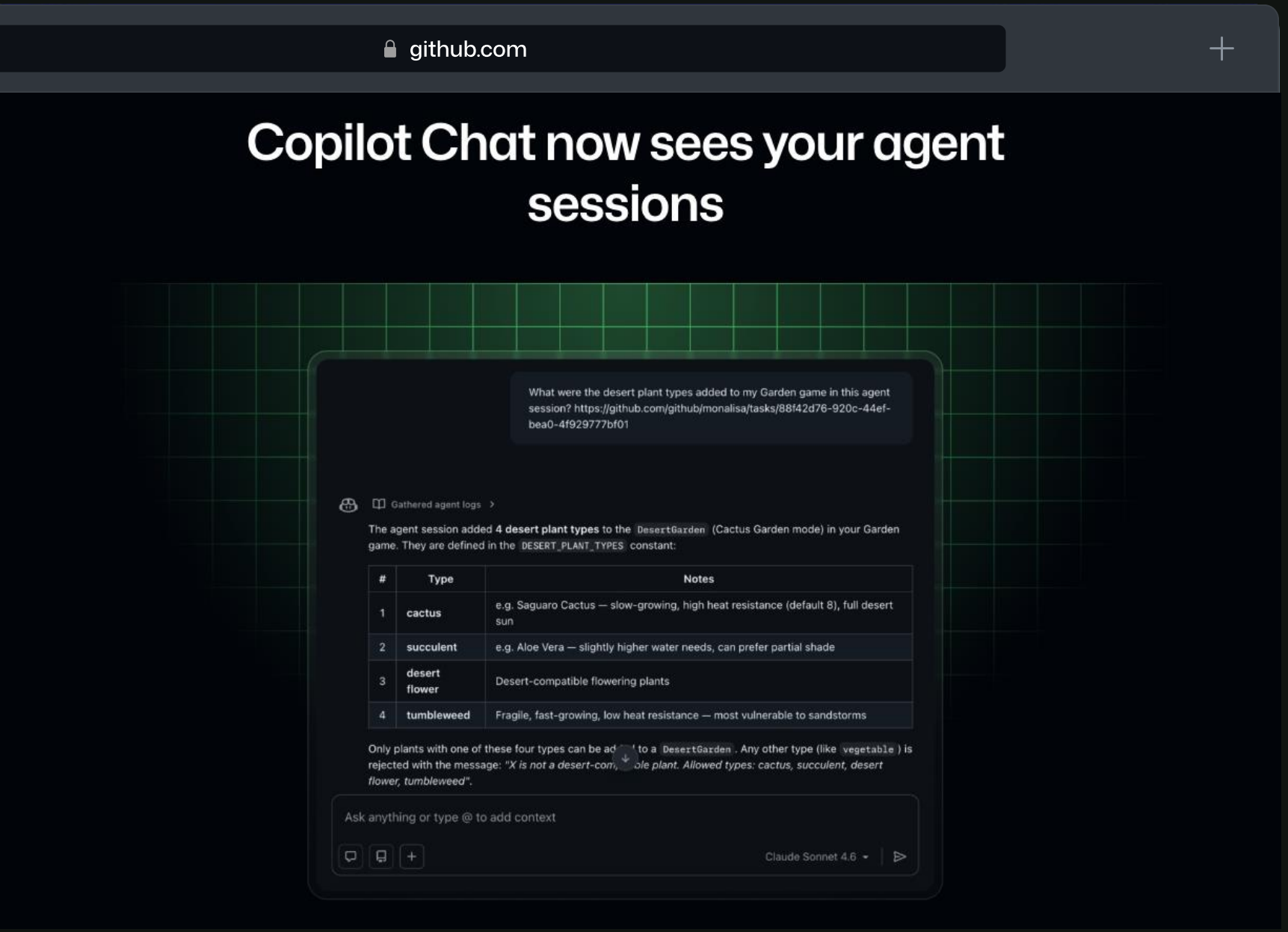
Some ways to make agents remember



Public preview – Jan. 26'

Allowing agents to store repo and user-based snippets of information during a session

Some ways to make agents remember



Release – Jun. 26'

Letting your agents access other sessions

This talk is about the system we built to eval these features

Modeling how developers work

Developer work is sequential

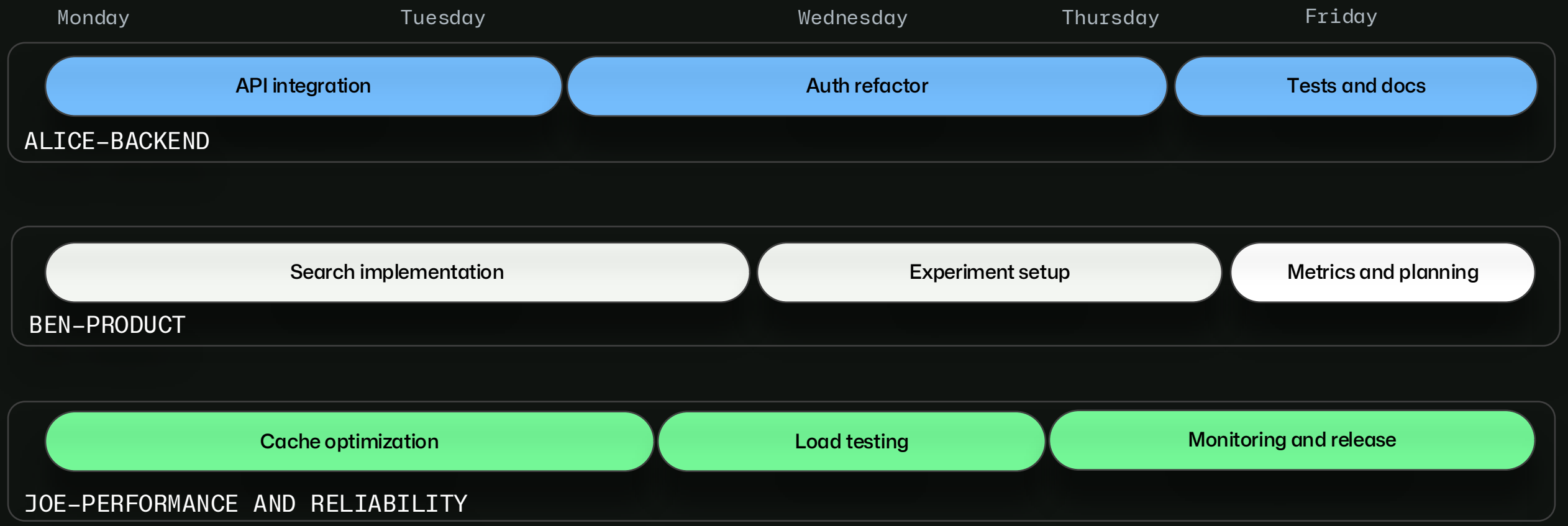
Many benchmarks currently measure one-off pieces of work and grade agents on those standalone tasks. This is fundamentally different than how we operate as developers

A solo developer's weekly schedule of work on a feature might look like:

Monday	Tuesday	Wednesday	Thursday	Friday
Implement core backend functionality	Hook up telemetry	Build frontend	Bug Fixing	Test, polish, and run online experiment

Developer work is collaborative

Most projects are built by teams working together and building on top of each other, not by solo developers.



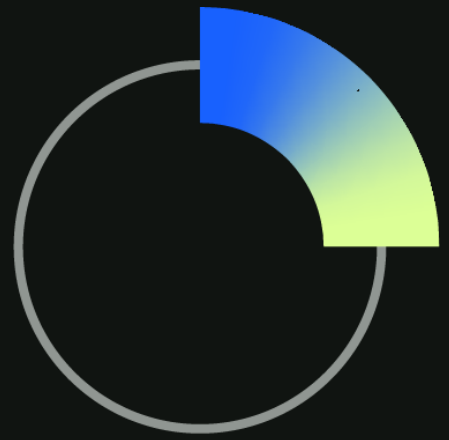
Developer work is (often) messy

The repos on which developers build are often large, complex, and sometimes messy!



PRs that touch 6 or more files

30%



Repos above 100MB

25%

BENCHMARK REQUIREMENTS

How can we build a benchmark that tests how well agents learn over time?

Sequential

The ordering of the tasks matter

Collaborative

Few developers/agents work in isolation

Messy

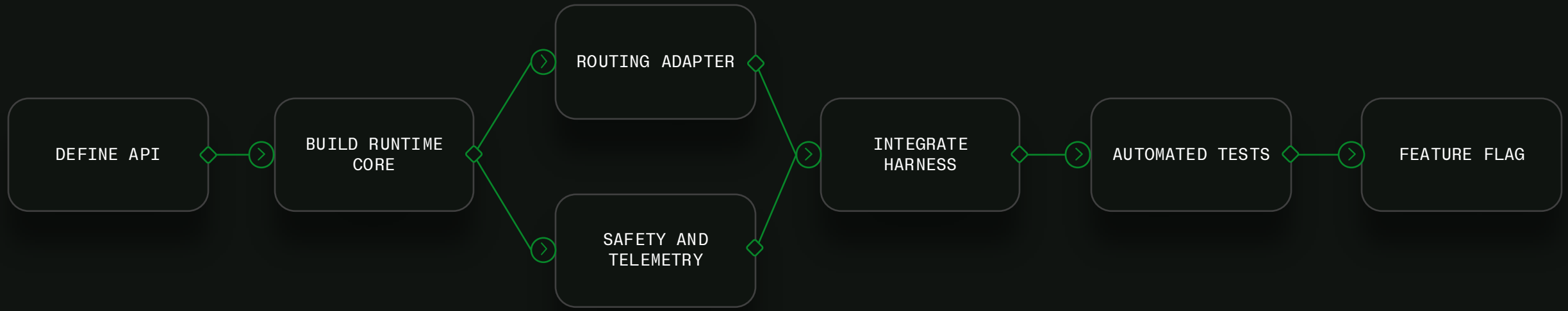
Repos are often complex and intertwined with other dependencies

Extensible

This benchmark needs to test various repo scenarios and be easily extensible

Software development as a graph

Evaluate agents completing tasks sequentially, and accumulating experience along edges. An idealized example:



Two questions remain:

1. How do we build these abstract graphs?
2. How do we build benchmarks to represent these nodes?

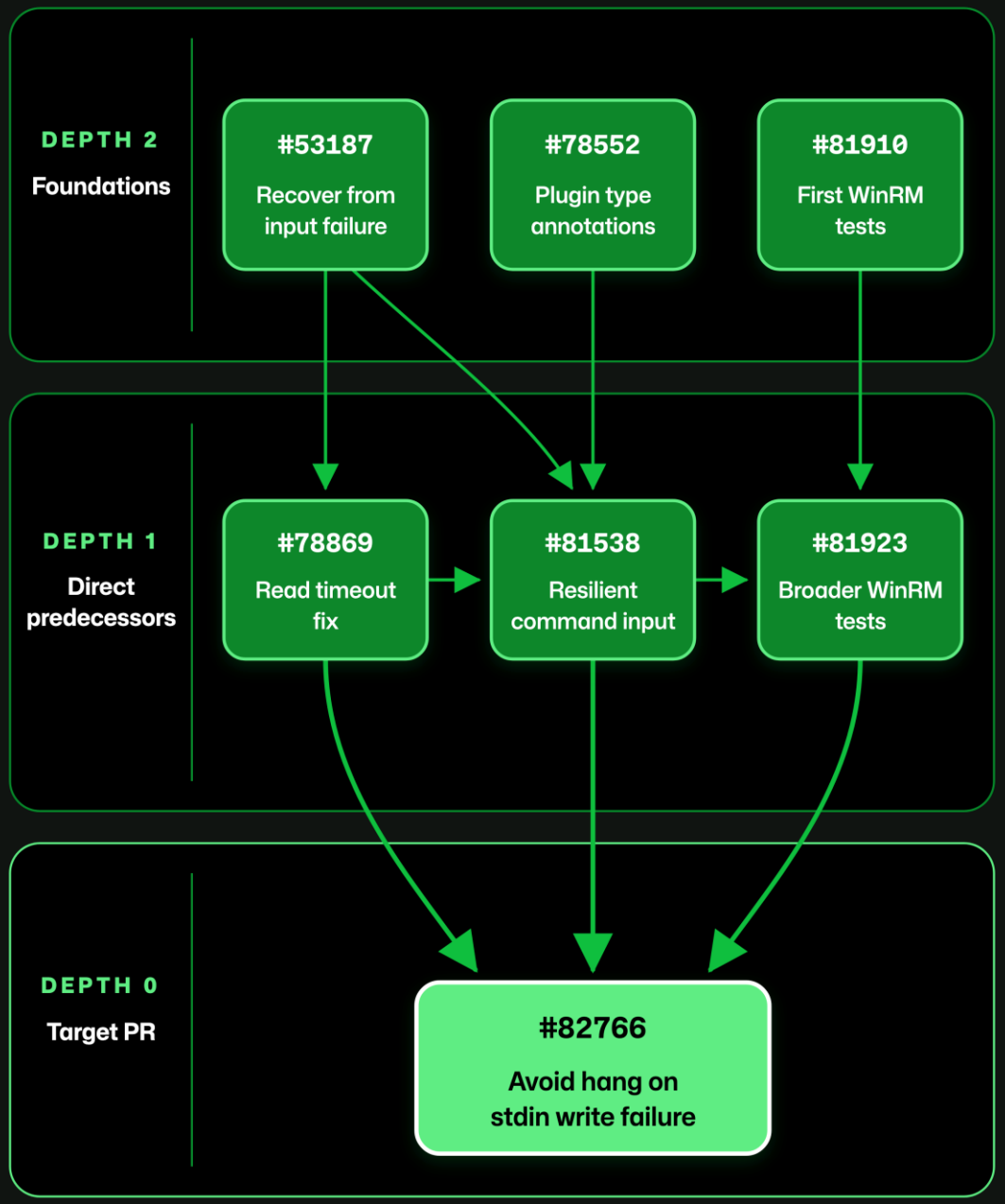
A graph of GitHub PRs

Create candidate nodes

Given a PR use GitHub's semantic issue search capabilities to generate a list of n related PRs

Greedily add nodes

Add edges connecting the target PR to related PRs, then repeat for each node



How do we build the nodes?

BUILDING THE NODES

Benchpress

- Turns real software changes into standardized coding challenges
- Creates everything needed to evaluate an AI coding agent: instructions, tests, environment, and known solution
- Works with repositories hosted on GitHub and Azure DevOps
- Makes it easier to compare AI coding agents fairly and consistently

From real fixes to repeatable tasks

01

A real code change

Start with a merged PR from a developer's repository.

02

AI builds the task

Write the problem and tests. Recreate the starting setup.

03

A runnable challenge

A self-contained task agents can attempt in the same setup every time.

Each task becomes one node in our graph.

What do Benchpress tasks look like?

WHAT THE AGENT GETS

Stop a code-checking tool from crashing on certain invalid inputs.

The agent starts from the version before the fix.

HOW WE CHECK IT

Tests of the behavior

Does bad input report an error without crashing?

A known fix

Used to validate the challenge, not supplied as the agent's answer.

Simplified from a real Roslyn (Microsoft's C#) compiler fix.

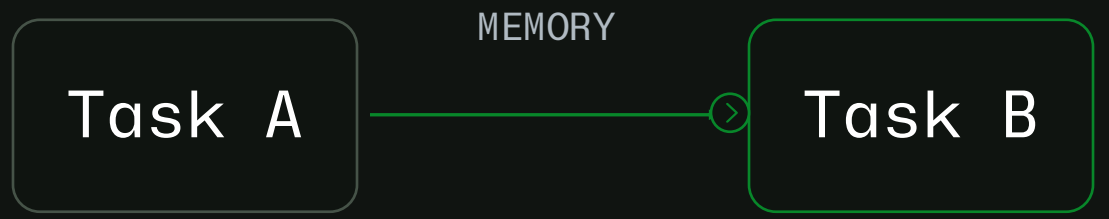
Reliable tasks make memory measurable

Without memory



No earlier experience

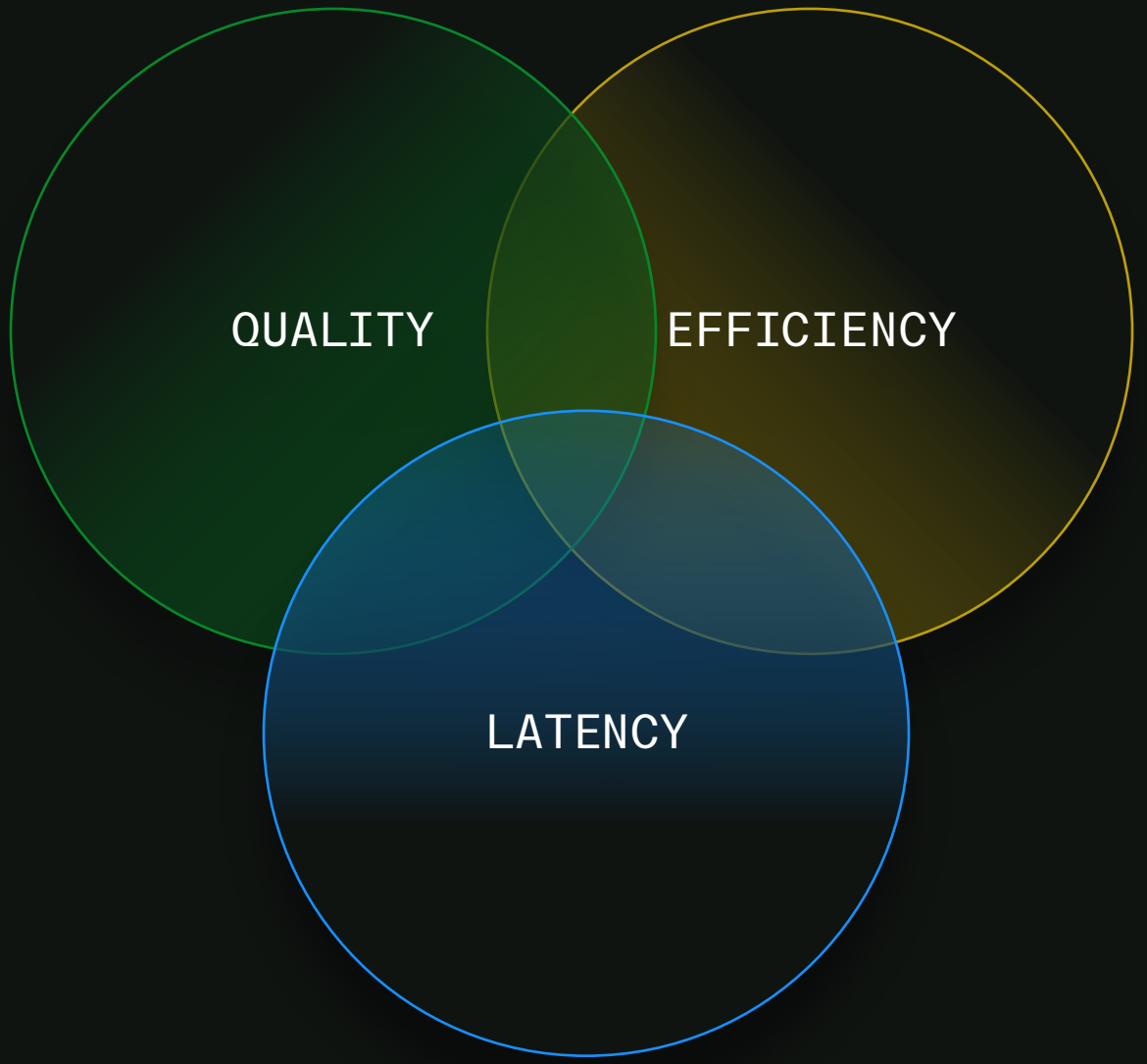
With memory



Experience from a related task

Allows us to answer: What impact does memory have on task B?

What can we now measure with this?



Does accumulating knowledge and giving the agent the ability to ”remember” improve:

- How well it performs a task
- How fast it can perform a task
- How efficiently it can perform a task

Evaluating agent memory

The three components of a memory system



What to remember?

Copilot is already powerful, so how can we make it *smarter?*



How to deliver it?

How can we make sure Copilot *gets* this information when it needs it?



How to update it?

Even the best knowledge goes stale, so how do we update it?

EXAMPLES

Real pull requests from VS-Code

These PRs are related.

There *should* be some learnings between them.

Let’s evaluate how Copilot learns!

Merged agents/block4a-reconcile-v2-chat-catalogs → main

agentHost: reconcile older-build edits into normalized chat catalogs and convert every session (Block 4a) #341904

Description Files 31 Reviews ✓ 4/4 Checks ✓ 36/36 Commits 24

Problem

Since #340980 and #341086, Copilot CLI, Claude and Codex sessions keep a **legacy** chat catalog (authority 1), and conversion to the normalized catalog (authority 2, `chats_v2`) is disabled for them. That's because a stable build without normalized-catalog code (1.140 and 1.141) writes only the legacy stores, and on a normalized session those edits were lost on the next upgrade:

- created chats dropped from the list;
- deleted chats came back;
- archive, read and order changes reverted.

Moving chats between sessions (Block 4) only works on normalized catalogs, so this blocks it.

Change

OCTOBER 16, 2026, AT 02:51:36

PR #341904

This PR adds a way to update old chat records, retrying when they aren’t ready.

Merged agents/fix-catalog-conversion-retry-storm → main

agentHost: stop retrying chat catalog conversions that wait for an impossible re-projection #342077

Description Files 6 Reviews Checks ✓ 48/48 Commits 2

Fixes a production regression from Move Chats Block 4a (#341904). There is no separate issue; it was found in today's Insiders agent host logs.

Root cause

Block 4a C4 removed the `requiresLegacyChatCatalog` gate, so older provider-scheme sessions (`codex:/`, `copilotcli:/`, `claude:/`) now go through chat catalog conversion. For about 110 of them, the provider can't vouch for the session. Either it is registered with an unreadable catalog (`providerUnavailable`), or it no longer knows the session (`sourceUnresolvable`, parked). Their aggregate therefore never becomes verified, and every conversion ends `notReady: the session aggregate is unverified or awaits its re-projection`. Each attempt then fed a loop:

1. **Opened the session DB before checking.** `_migrateChatCatalog` ran `reconcileLegacy` and `runMigrationExclusive`, which calls `tryOpenDatabase`, before `migrateChatCatalogV2` read the cheap central state.
2. **Marked the payload dirty on `notReady`.** This bumped the dirty revision so an in-flight

OCTOBER 16, 2026, AT 17:15:00

PR #342077

This PR fixes that same retry logic so it waits until the records are actually ready.

EXAMPLES

Turn the PRs into test cases

Use Benchpress to turn these PRs into Harbor-compatible test cases

Upgrade old chat records automatically

Problem

Some saved chats still use an older format. Changes made by older app versions can also be missed by the new format.

Expected behavior

- Add automatic upgrades for every saved session, preserving changes made by older app versions.
- If an upgrade needs refreshed session data, defer it and retry after that data is refreshed.

OCTOBER 16, 2026, AT 02:51:36

PR #341904

We solve this task first and pass state to the next test case.

Stop retrying chat upgrades before they are ready

Problem

The chat upgrade process from Task 1 keeps retrying some records even when their required session data cannot be refreshed yet.

Expected behavior

- Fix the existing retry logic so it waits until the required session data is confirmed ready.
- Leave blocked upgrades waiting without repeated attempts, then resume when their data becomes ready.

OCTOBER 16, 2026, AT 17:15:00

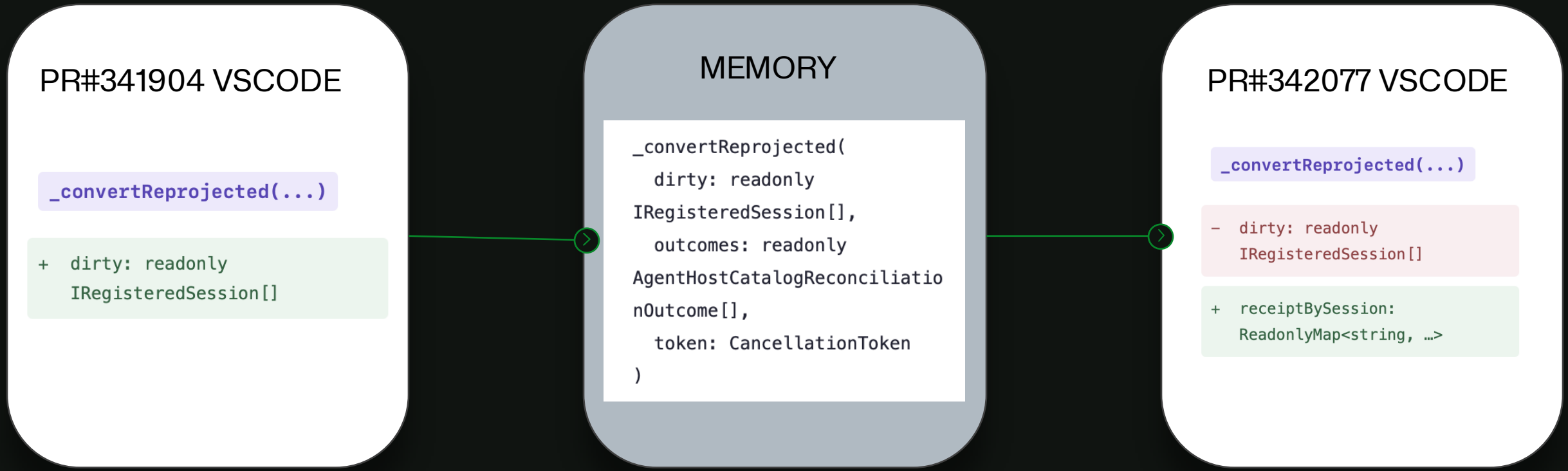
PR #342077

We measure performance on this task with and without the upstream memory.

Learning what to remember

⊗ Bad: exhaustive documentation

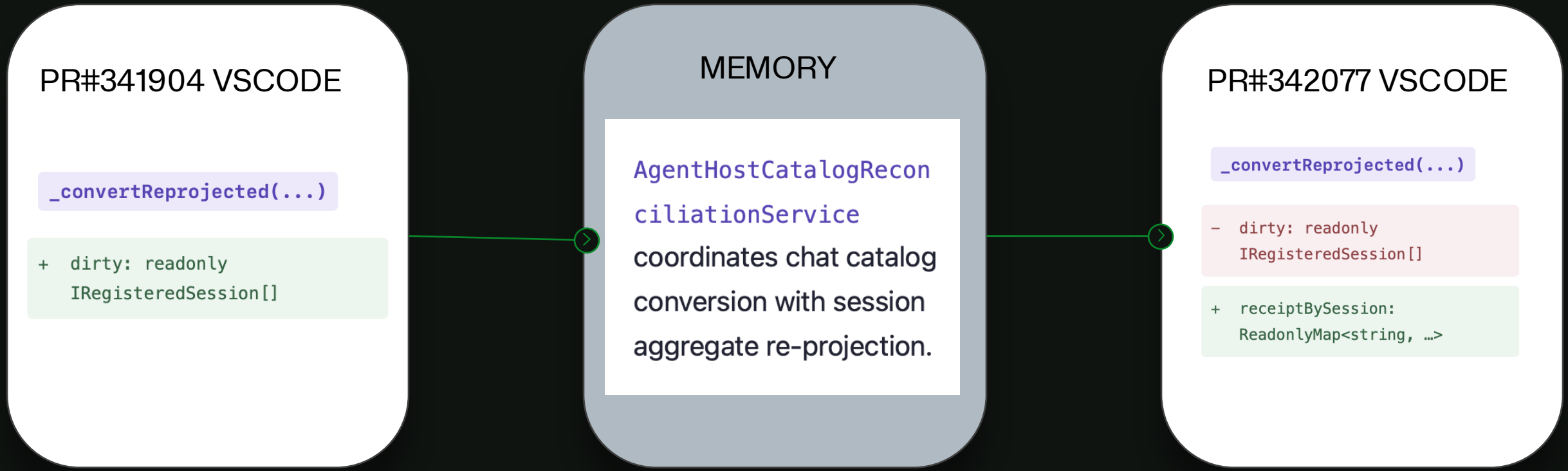
Granular knowledge is immediately stale



Learning what to remember

Good: core architectural patterns

Broad knowledge is still useful



Learning what to remember

⊗ Bad: simple explanations of code from few files

Copilot already can learn this with one file view

PR#341904 VSCODE

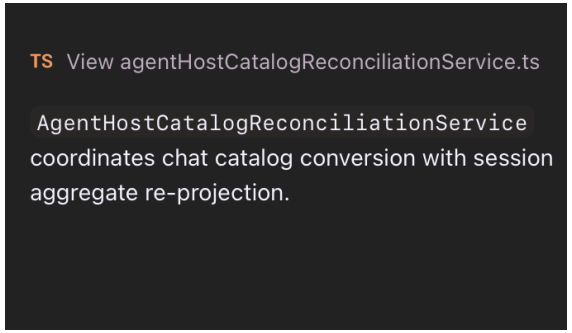
```
_convertReprojected(...)
```

```
+  dirty: readonly
   IRegisteredSession[]
```

MEMORY

AgentHostCatalogReconciliationService coordinates chat catalog conversion with session aggregate re-projection.

PR#342077 VSCODE



Learning what to remember

Good: multi-hop reasoning requiring *deep* knowledge across the codebase

This saves the agent more time, decreases probability of failure

PR#341904 VSCODE

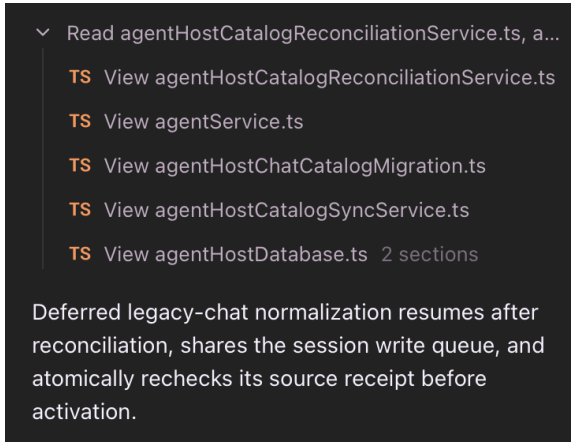
```
_convertReprojected(...)
```

```
+  dirty: readonly
   IRegisteredSession[]
```

MEMORY

Deferred legacy-chat normalization resumes after reconciliation, shares the session write queue, and atomically rechecks its source receipt before activation.

PR#342077 VSCODE



Learning what to remember

Multi-hop reasoning requiring *deep* knowledge across the codebase

User preferences

Gotchas / failure cases

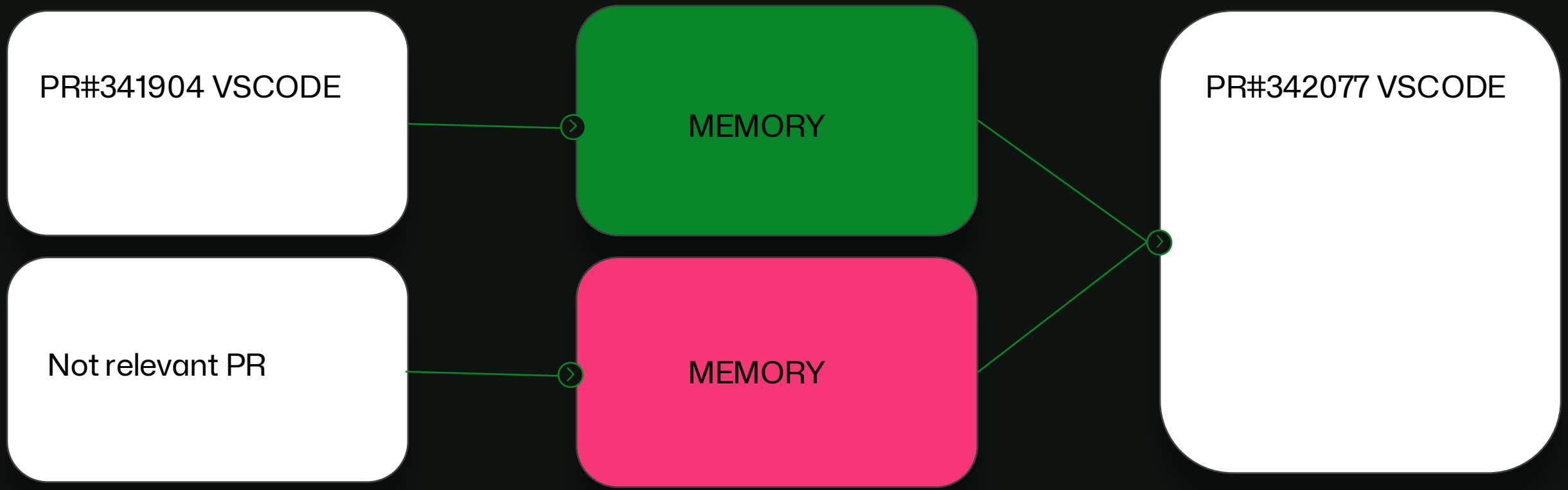
Skill workflows

We can build a useful memory, but not *every* memory is useful *all* the time

So how do we choose the *right* memory?
At the *right* time?

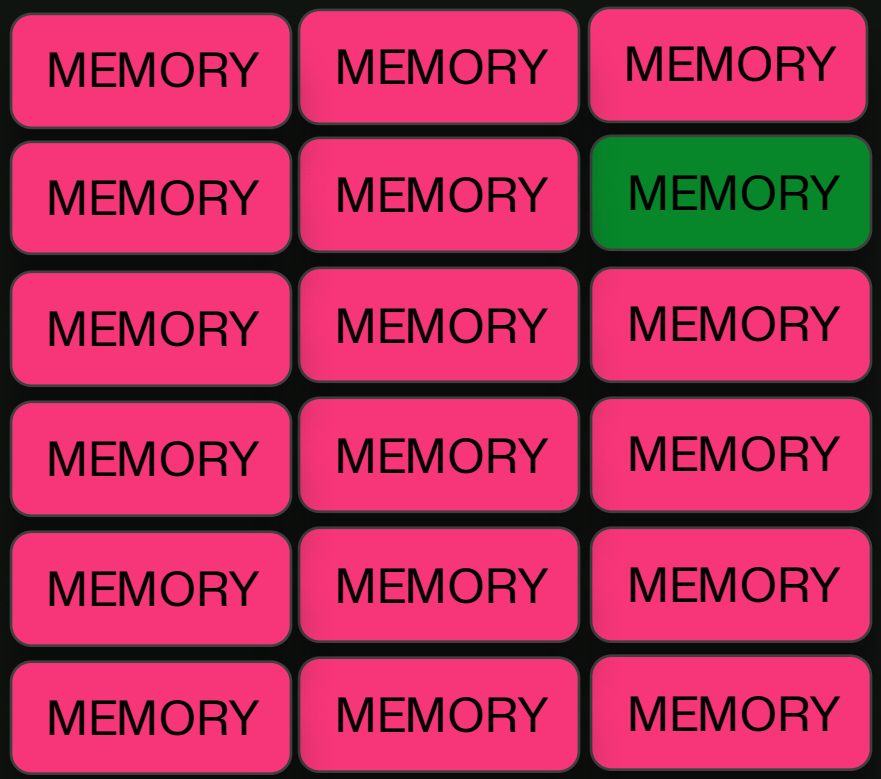
Learning how to deliver a memory

Assemble a graph with *two* upstream tasks: one relevant, one *not* relevant
Can Copilot *find* and *use* the right memory?



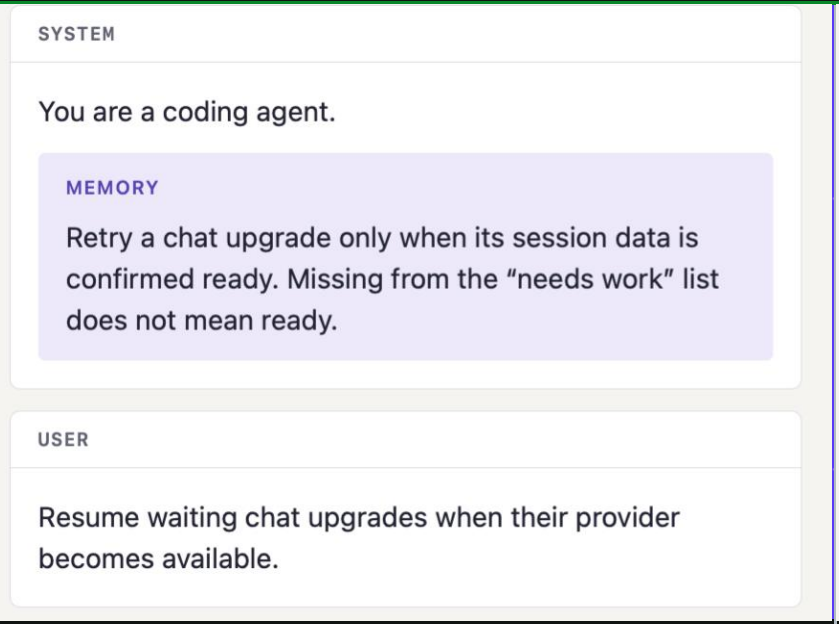
Learning how to deliver a memory

Can we still find *the right memory* when our memory bank gets bigger?



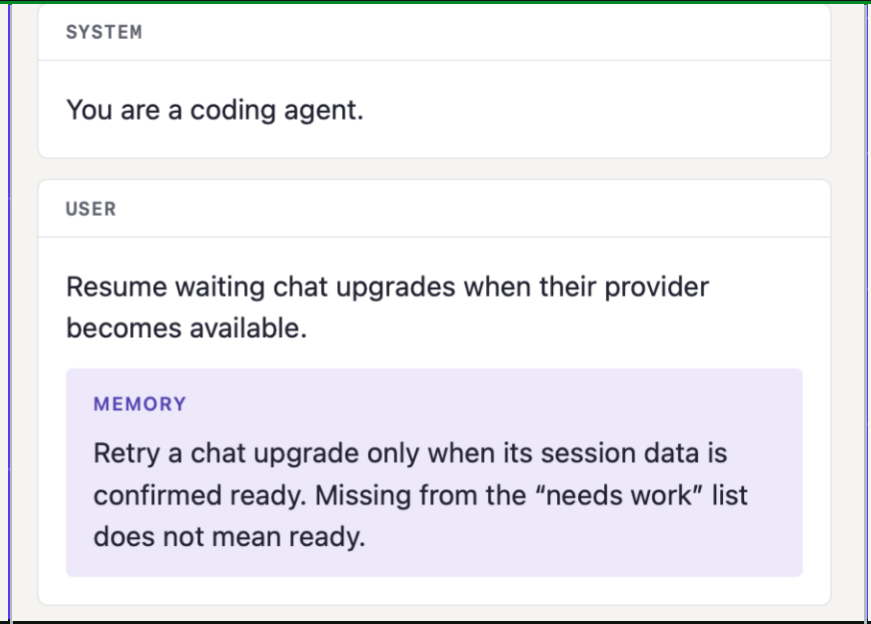
PR#342077 VSCODE

Learning how to deliver a memory



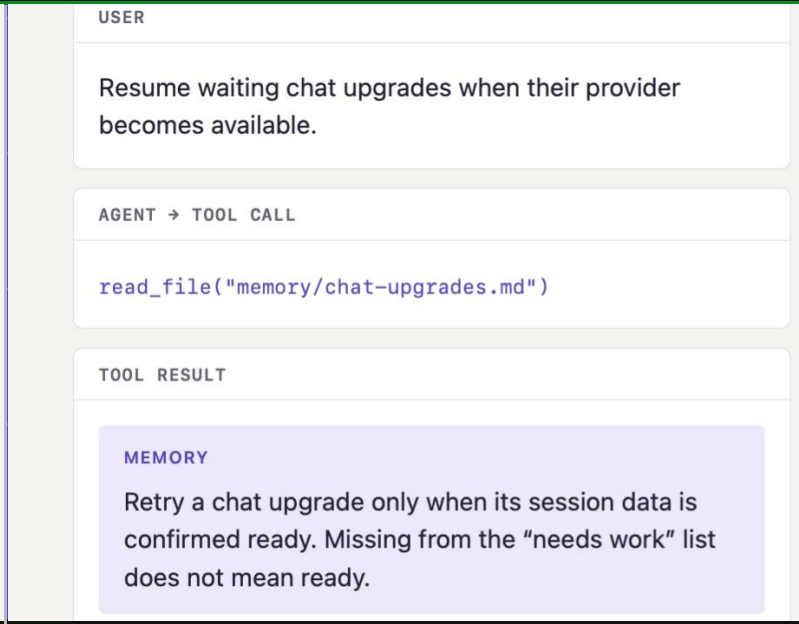
System message

Inject the memory in the system instructions.



Append to user message

Inject the memory below the task.



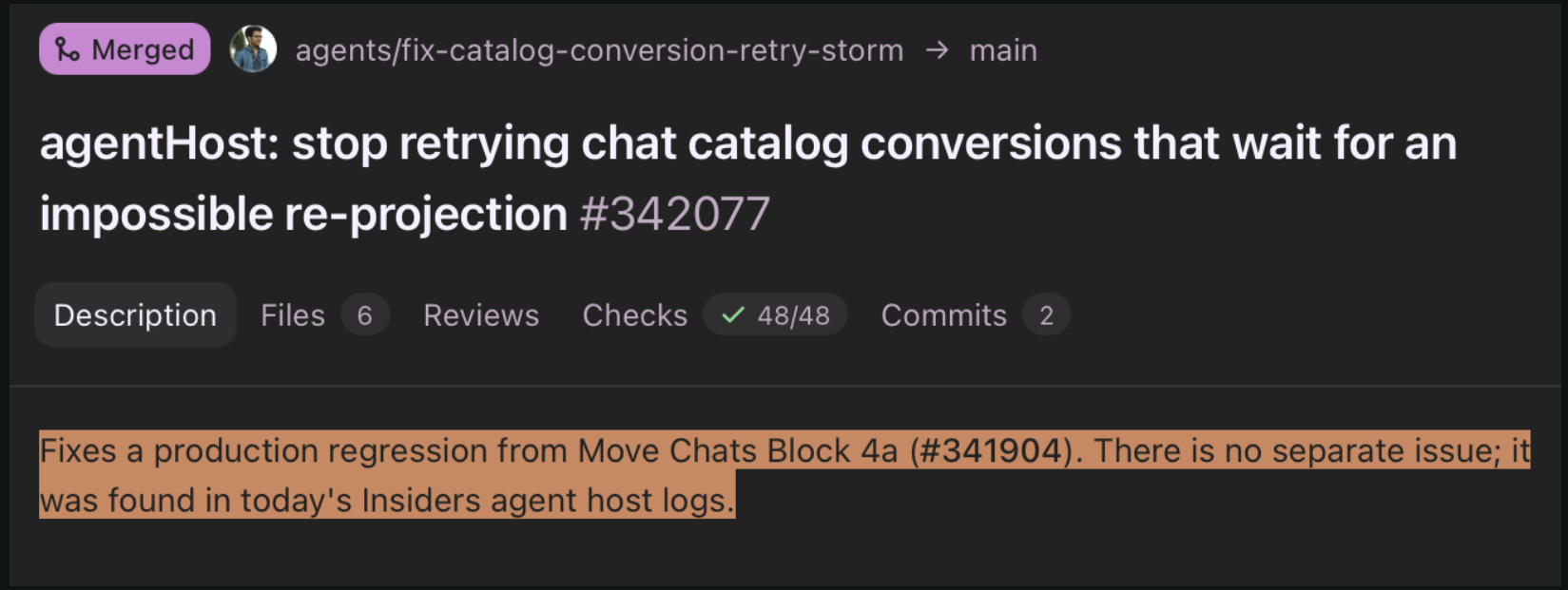
Read through a tool

The agent retrieves memory during the task.

Code changes and memories go stale. So how do we measure how our system *updates?*

Learning how to update our memory

Our original PR actually introduced a regression



Learning how to update our memory

Add a *third* task to our sequence, which tests that we have healed.

Regression

PR#341904 VSCODE

Regression: Blocked sessions were mistaken for ready ones, causing repeated upgrade attempts.

Fix

PR#342077 VSCODE

Fix: Require proof that session data is ready before retrying an upgrade.

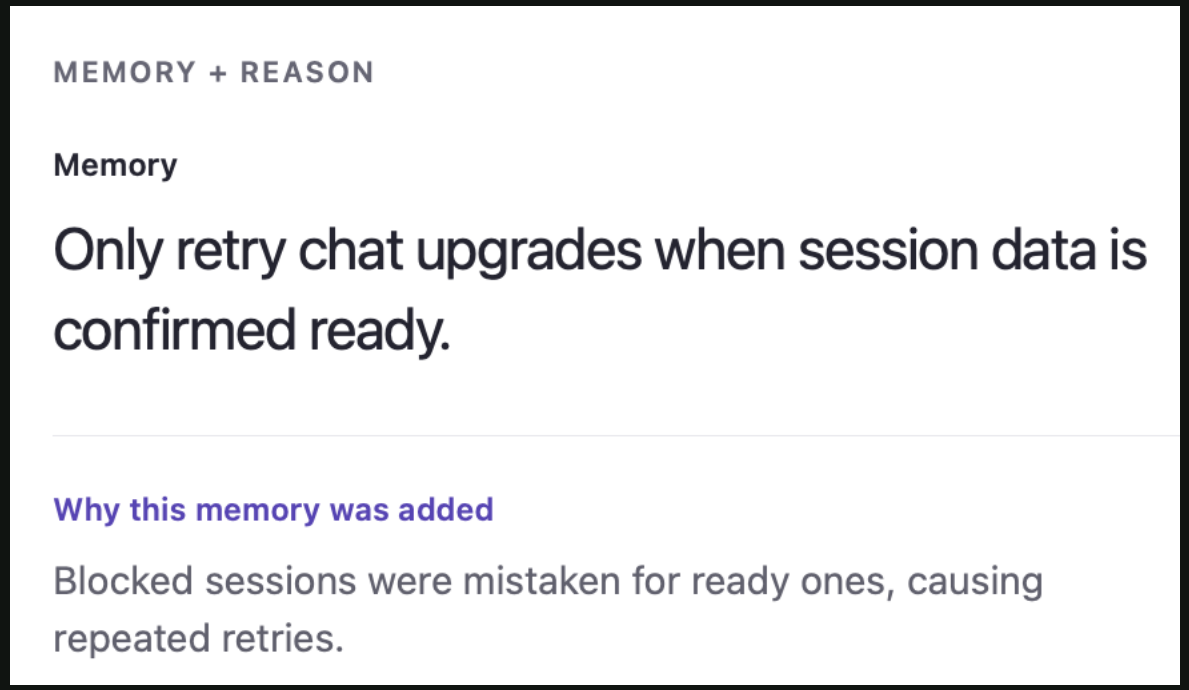
Test healed

PR#342127 VSCODE

Memory test: Check whether the agent preserves that readiness rule when adding provider-recovery behavior.

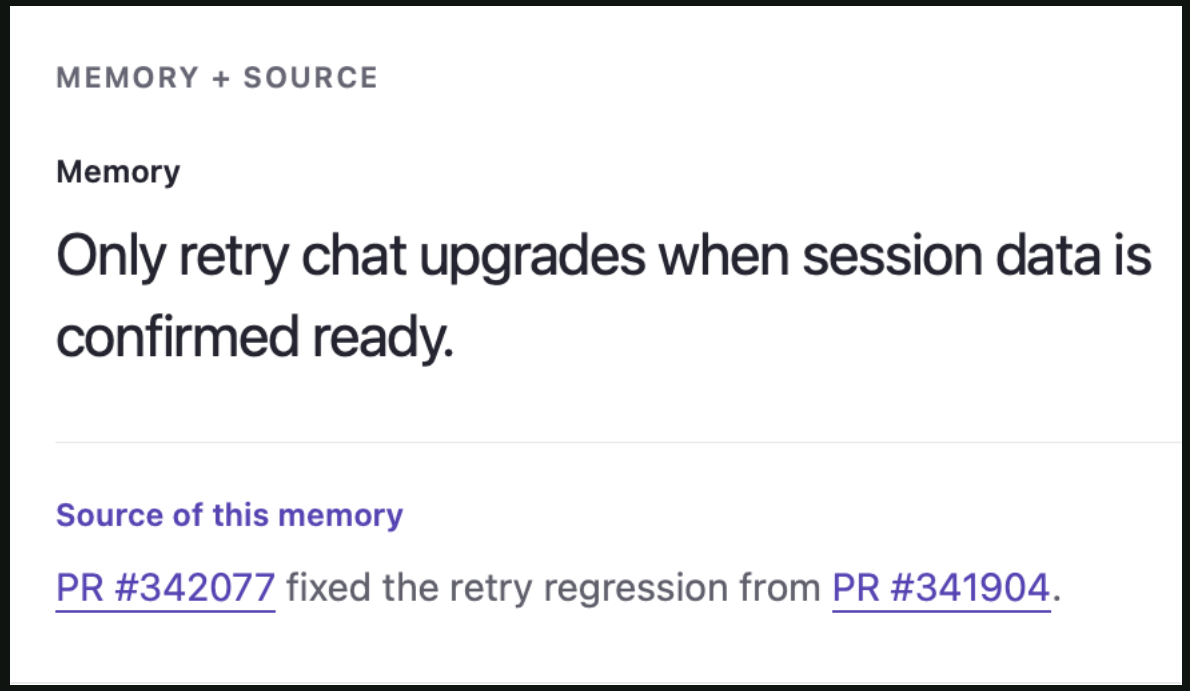
Learning how to update our memory

Provenance – recording *why* we remembered something



Learning how to update our memory

Provenance – recording *where* the memory came from?



What's next?

Agents are always improving, rendering past learnings outdated.

Evaluations keep the learning loop running by revealing what works, what breaks, and what to improve next.



Thank you

How was this session?

We do actually read these. It's how Universe gets better every year.
Scan to review.

