AI AGENT MEMORY

I gave AI agent a Dory memory And it's not what you think.

I've been cooking on the AI Agent memory problem for a while now. Got inspired by chats with my mates here at Safewill, and I ended up building something that's actually been helping me day to day.

Here's how I went from missing critical context every time I write a solution design — to having AI surface the blind spots for me in 2 seconds.



THE PROBLEM

Missing context is painfully expensive

When I'm writing a solution design, it's so easy to miss stuff. Another team's initiative depends on the same code I'm changing. A feature flag is still running a 50/50 experiment. Someone already tried this approach and it didn't work.

The unknown unknowns

- A dependency you didn't know existed breaks in prod
- Your SD gets sent back: "did you check with Team X?"
- Halfway through implementation, you find out someone already solved this
- You deploy and accidentally break another team's running experiment

The real cost

- Rewriting the entire SD after discovering a missed flow
- Duplicate work across teams who didn't know about each other
- Incident postmortems that say "we didn't know X was connected to Y"
- Knowledge locked in people's heads — gone when they leave

FIRST ATTEMPT

I tried getting AI to search Confluence

Claude Code has an Atlassian integration. So I asked it: *"What do I need to know before working on the onboarding redesign?"*

What actually happened

1. Search Confluence for "onboarding" pages
2. Read each page, pull out keywords
3. Search again with those keywords
4. Read those pages, find more references...
5. **Hit the Atlassian rate limit. Start over.**

And the worst part?

After 15-20 API calls and 90 seconds of waiting, I got 163K characters of raw JSON. No relationships between tickets. No people mapping. Lessons from incidents? They're in Slack, not Confluence.

And next sprint, when I touch a similar area? **The whole loop starts again.**

THE BREAKTHROUGH

What if AI had its own memory?

Instead of searching Confluence every single time, what if I built a persistent memory that knows everything — projects, people, relationships, past decisions, lessons learned — and can answer questions instantly?



I called it **Dory** — because the whole point is to remember things. Yes, the fish from Finding Nemo.

A graph, not flat files

Instead of markdown docs, Dory stores knowledge as **nodes and relationships**. "Alice engineers on Project X which depends on Project Y" — that's a first-class concept, not text buried in a doc.

Syncs from your tools

It pulls from Jira, Confluence, and Slack automatically. Projects, solution designs, who's working on what, incidents, lessons — all connected in one graph.

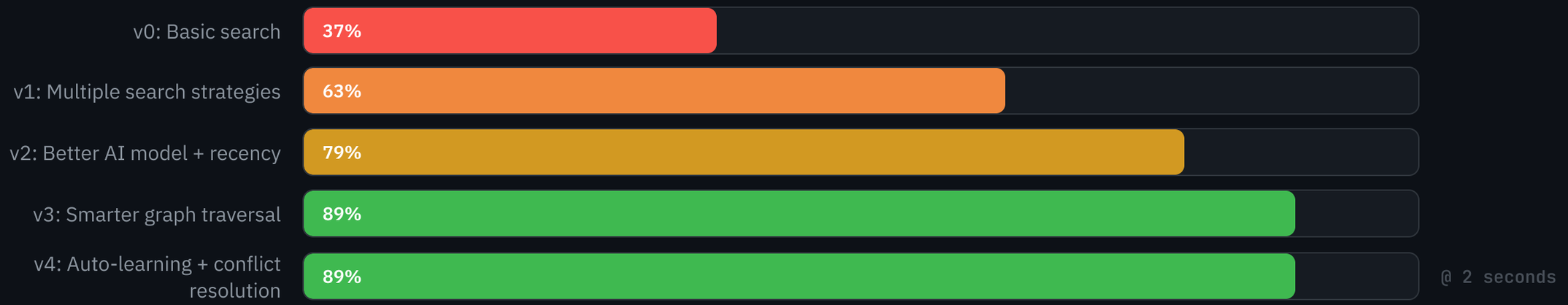
Just ask it questions

Natural language in, structured answers out. Not a JSON dump. Ranked results with relationships, people, and blind spots — in under 2 seconds.

THE JOURNEY

I iterated 5 times in a day

Didn't plan everything upfront. Built the simplest thing, ran a benchmark, found the gaps, fixed them. Repeat.



The biggest single win? Breaking each question into sub-questions. Instead of one search for "onboarding redesign lean solution design", it also searches "onboarding", "redesign", and "lean" separately. That alone jumped accuracy from 37% to 63%. Sometimes the simplest idea is the best one.

THE DIFFERENCE

Same question. Two completely different experiences.

"I'm starting the onboarding redesign. What projects, people, and risks should I know about?"

With Dory — 1 question, 2 seconds

Related projects	10 found, with tech details
People to talk to	8 people, with expertise
Dependencies	Shown as graph links
Blind spots	10+ surfaced
Past lessons	From incidents + retros
Active experiments	Flagged automatically

VS

Confluence — 15+ API calls, 90 seconds

Related projects	Titles only, no details
People to talk to	Just an "assignee" field
Dependencies	Not captured at all
Blind spots	None surfaced
Past lessons	Locked in Slack threads
Active experiments	Have to check manually

What this actually changes

45x

Faster context gathering

2 seconds vs 90 seconds

10+

Blind spots caught

That I'd have missed completely

$0

Running cost

Everything runs on my laptop

Before: 15-30 minutes searching Jira, reading Confluence, checking Slack. Manually piecing together who worked on what, what depends on what, what went wrong last time.

After: One question. Two seconds. Full context including relationships and blind spots I wouldn't have found on my own.

IT GETS SMARTER

Dory learns as you work

It's not a static snapshot. Dory syncs from your tools and learns from conversations. When something comes up in a standup or a PR review, you feed it in and it remembers.

Auto-extraction

Paste raw text — meeting notes, Slack threads, standup updates. A local AI model pulls out the entities and relationships, matches them to what it already knows, and saves them.

```
"Mike said PROJ-821 is moving to Elaboration"
→ Updates status. Links Mike as engineer.
```

Conflict resolution

When new info contradicts what's stored — a status change, a role change — Dory catches it and replaces the outdated fact. The graph stays current without manual cleanup.

```
Old: "PROJ-821 is in Backlog"
→ Replaced: "moved to Elaboration"
```

BIGGER PICTURE

This isn't just an engineering thing

The "unknown unknowns" problem shows up everywhere in an org. If a graph memory helps engineers write better SDs, it can help others too.

Product Managers

"What experiments are running in checkout right now?" "Which initiatives touch partner onboarding?" Instant context before planning sprints or writing PRDs.

New Joiners

"Who owns the executor flow?" "What happened with the last redesign?" Access to institutional knowledge from day one — not locked in someone's head.

Leadership

"What's everything touching the onboarding funnel right now?" Cross-team visibility without scheduling alignment meetings.

The idea is simple: **a shared, structured, searchable memory of everything your org knows**. Not buried in Confluence pages nobody reads — connected as a graph that AI can navigate.

SIMPLE STACK

Runs entirely on a laptop

No cloud services, no API keys, no subscription. A graph database, a Python server, and a local AI model.

500+	800+	89%	2s
Knowledge Nodes	Relationships	Retrieval Accuracy	Average Response

I benchmarked this against the **official Neo4j GraphRAG library** — same database, same AI model, same query. Dory hit **89% vs their 58%**. Turns out the retrieval pipeline matters way more than which library you use.

Built with Neo4j, Python, Ollama, and Claude Code. The whole thing took a day.

TAKEAWAYS

What I took away from this

The best AI tool is the one built for your context

Even well-built generic libraries can't compete with something purpose-built for your team's data, naming conventions, and way of working. Domain knowledge is the moat.

Iterate with benchmarks, not vibes

Every change got benchmarked against the same query. Some "obvious" improvements barely moved the needle. The biggest win was the simplest: asking the question multiple ways.

Memory is the missing piece in AI tools

LLMs are great at reasoning. Terrible at remembering. A structured knowledge graph fills that gap — persistent, queryable, relationship-aware memory that grows over time.

You can build this in a day

This isn't a 6-month platform play. It's a Python server, a graph DB, and a local AI model. Built, iterated, and benchmarked before dinner. The hard part isn't the tech — it's knowing what to put in the graph.

TV

Tung Vo
Software Engineer • Sydney