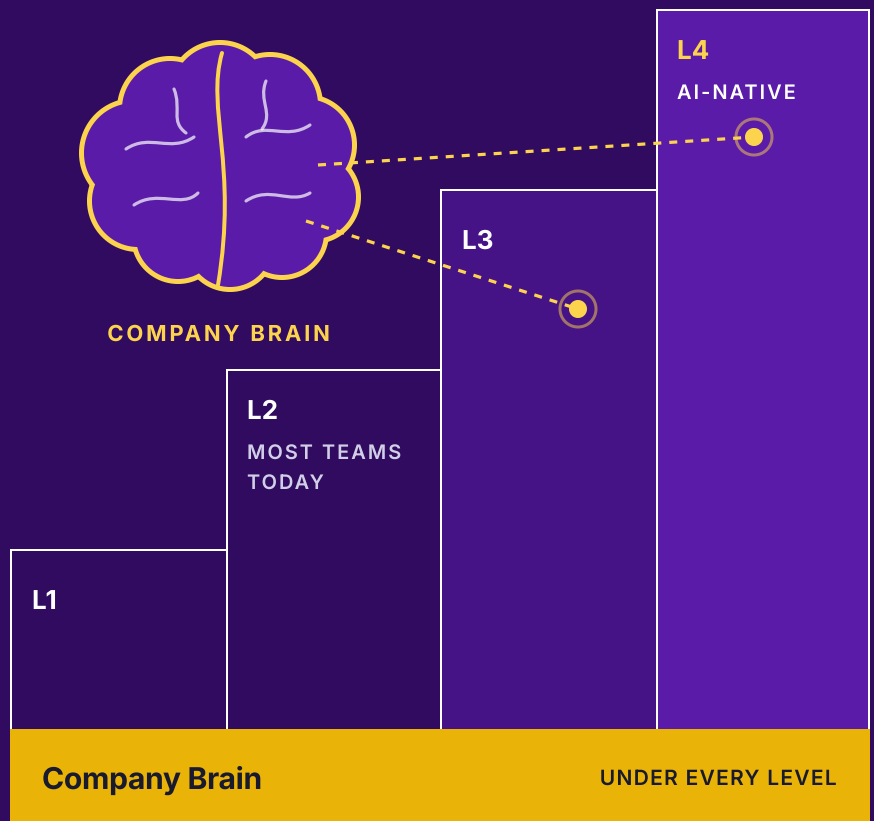


FOR PRODUCT AND ENGINEERING LEADERS

Why product and engineering teams need a company brain to become AI-Native

Agents are only as good as the context they work from. How a shared, permission-aware knowledge layer moves teams up the four levels.

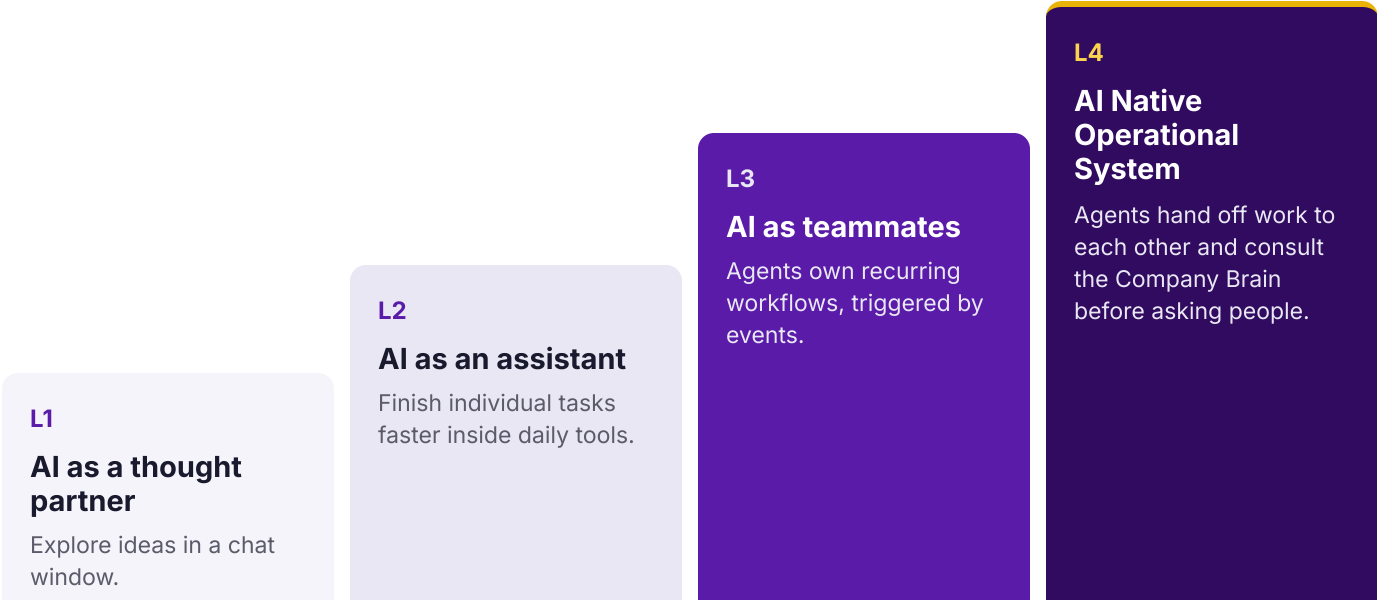


INSIDE

- The four levels
- What a Company Brain is
- How agents use the Brain
- Use cases
- Case studies: SRE, PR Review, Data Analyst
- The AST code graph
- The Analytics Harness

Every level above the first depends on context.

AI adoption in product and engineering moves through four levels, from a chat window an engineer opens on their own to a system where agents run critical workflows end to end. What separates the levels is less the model than what the model knows about your company, and who is allowed to start the work.



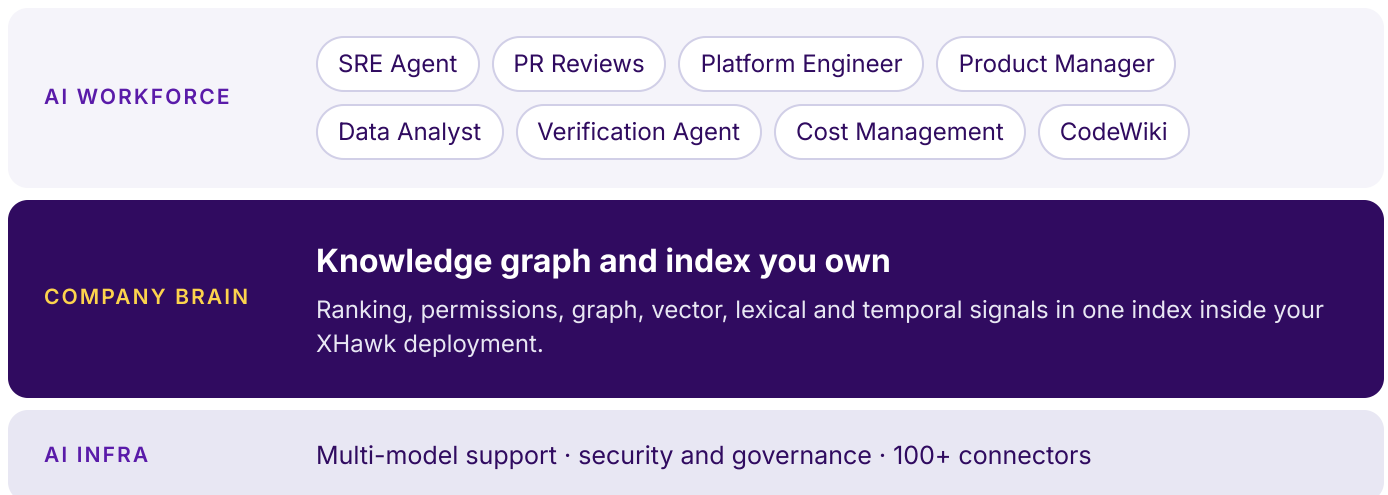
LEVEL	WHAT THE AI KNOWS	WHO STARTS THE WORK
L1	Pretraining, plus whatever someone pastes in	The individual
L2	One tool's slice: the repo in the IDE, the doc in the editor	The individual, inside a tool
L3	Shared company context across code, tickets, docs, and chat	An event: a PR, an alert, a ticket
L4	Shared context plus memory of past decisions, incidents, and outcomes	The system, within policies leaders set

WHY TEAMS STALL

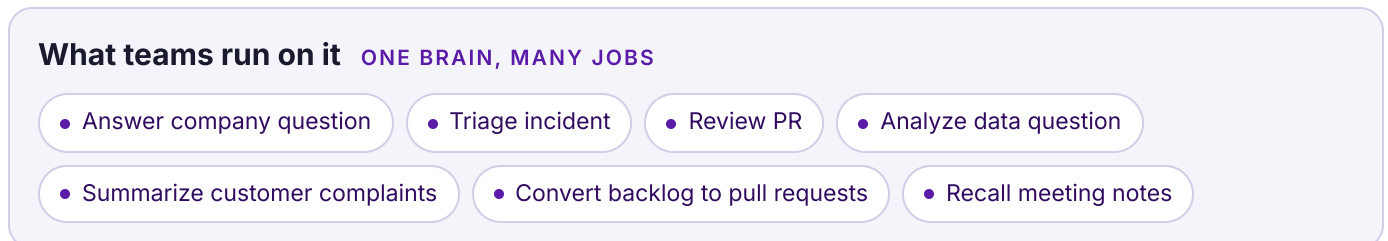
Most teams stall at Level 2 because each tool sees only its own slice. An agent cannot own a workflow it does not understand, and understanding a workflow means seeing across all the systems it touches.

Retrieval alone is not knowledge.

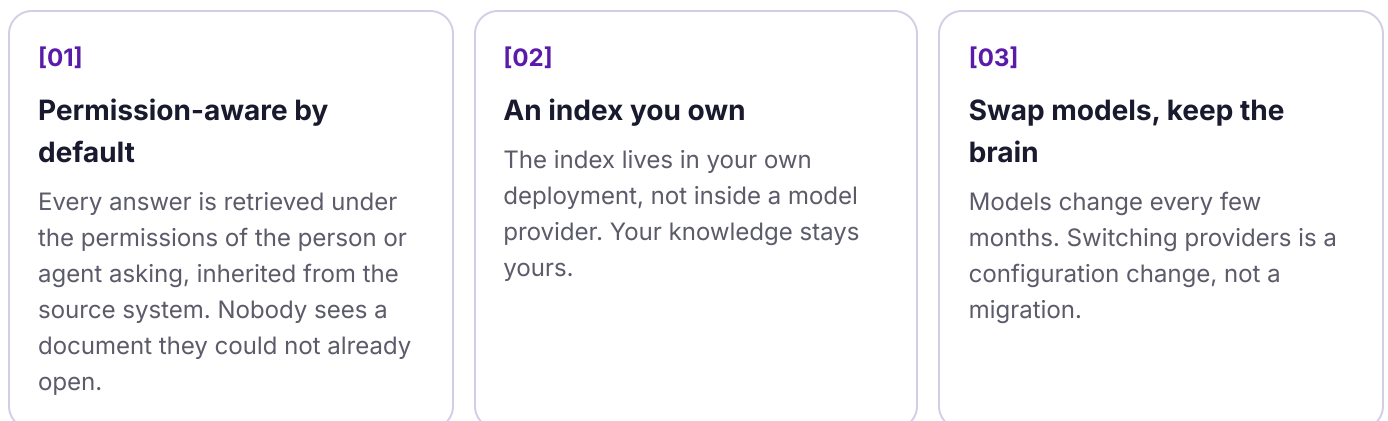
A Company Brain is the shared memory every agent works from. It indexes Slack, GitHub, Jira, Zendesk, Drive and 100+ more sources into one knowledge layer that is permission-aware and portable across every model you run. Product, analytics, engineering, and the exec team all read from the same context instead of keeping private versions.



The XHawk stack. Agents sit on top of the Brain; models and connectors sit underneath.

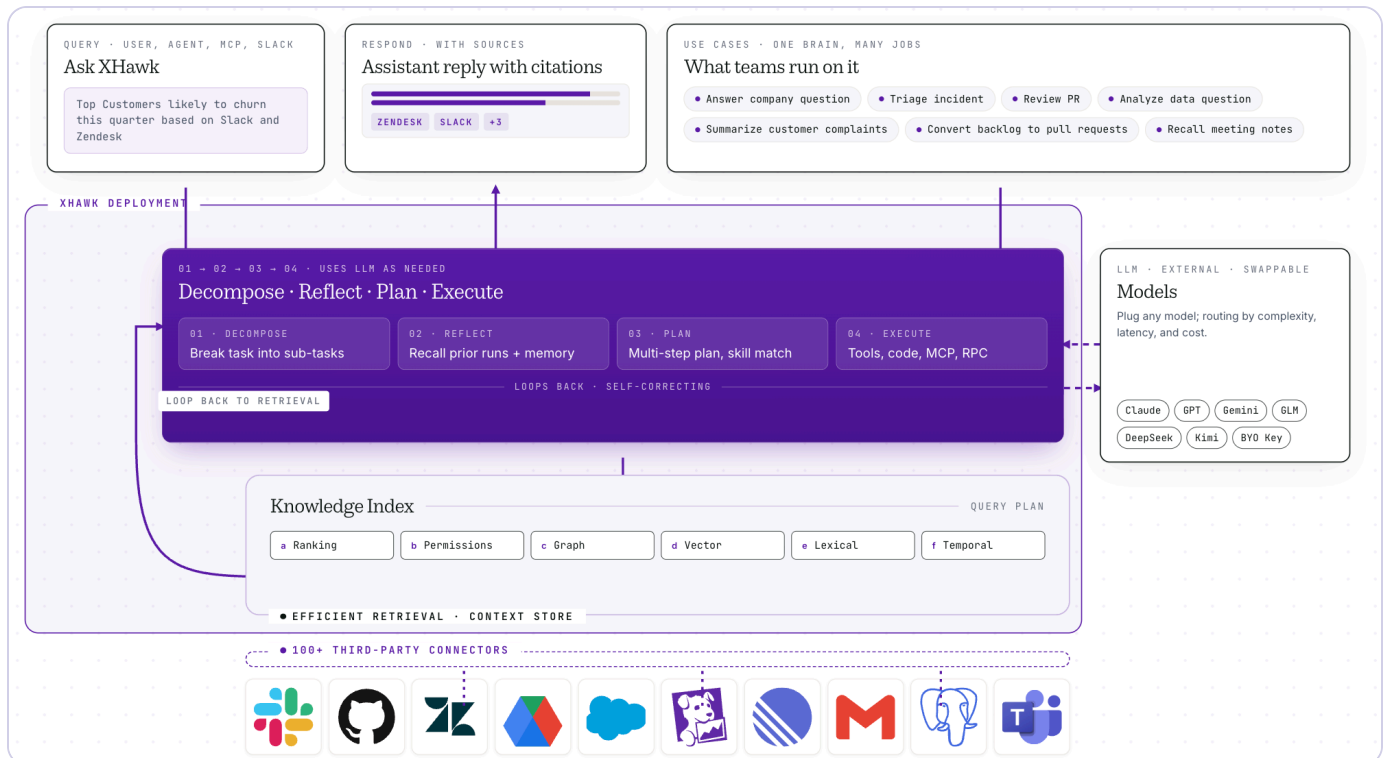


Three properties that make it last



From question to cited answer.

A query can come from a person, an agent, an MCP client, or Slack. Inside your XHawk deployment, the Brain decomposes the task, recalls prior runs, plans, and executes, calling an external model only when a step needs one. It loops back to the Knowledge Index until the answer holds, then responds with its sources.



The Company Brain inside an XHawk deployment. Models are external and swappable; connectors feed the index; all retrieval is permission-aware.

What the Knowledge Index adds

Graph links a service to its owner, its callers, its tickets, and its past incidents.

Vector and lexical find meaning and exact identifiers like error codes and ticket numbers.

Temporal prefers the latest decision over an older one that was reversed.

Permissions filter every result to what the asker can already open.

MODEL INDEPENDENCE

Each request routes by complexity, latency, and cost: Claude, GPT, Gemini, GLM, DeepSeek, Kimi, or your own key.

One brain, many jobs.

Every agent works from the same company context. That is what lets a product question, an incident, and a pull request all get answers grounded in the same decisions, owners, and history.

CTO asks the Company Brain

LEADERSHIP

"Why did we decide not to build this feature?"

Institutional memory a CTO can query directly, so leadership stops reopening old debates.

Product Manager

PRODUCT

"Draft a spec for SSO, using our last three auth decisions"

Specs, feedback synthesis, and launch readiness grounded in your docs and tickets.

Platform Engineer

ENGINEERING

"Take ENG-412 from the backlog and open a pull request"

Ticket to PR, migrations, and doc generation with repo conventions already loaded.

New Engineer asks CodeWiki

ENGINEERING

"How does the billing service talk to Stripe?"

Answers from the current code, not a stale README.

Data Analyst

CASE STUDY

"How many teams that signed up in July are still active?"

Plain English to query, with metric definitions codified once.

Verification Agent

ENG · DATA

"Check every figure in this graph against the warehouse"

Checks the work against the source before anyone acts on it.

SRE Agent

CASE STUDY

"What changed before the 02:14 alert fired?"

First responder that has already read the runbook.

PR Reviews

CASE STUDY

"Do a deep PR Review for the new onboarding feature"

Holds standards on every pull request, not only the ones a senior engineer had time for.

THE PATTERN

Different agents, one context. The case studies that follow show what that context layer does in incident triage, PR review, and analytics.

Triage needs more than the logs.

When an alert fires, the logs show what broke. They do not show which customers are complaining, which deploy went out ten minutes earlier, or that the same page fired three weeks ago and was fixed by a config rollback. An on-call engineer pieces that together from five tools. The SRE Agent starts with it already assembled.

CONTEXT LAYER

Logs and metrics	Datadog, Grafana
Alerts and pages	PagerDuty
Recent deploys	GitHub, CI
Customer complaints	Zendesk, Slack
Previous incidents	Postmortems
Runbooks and owners	Confluence



SRE AGENT

- 1 **Detect.** Watches alerts. No prompt needed.
- 2 **Correlate.** Ties the alert to the deploy and the change that preceded it.
- 3 **Size the impact.** Matches customer tickets to the failing service.
- 4 **Recall.** Finds similar past incidents and what fixed them.
- 5 **Act.** Runs the runbook, updates the Slack thread, escalates to the owner.
- 6 **Remember.** Drafts the postmortem and writes it back to the Brain.

With and without the context layer

	LOGS ONLY	WITH THE COMPANY BRAIN
Starting point	A stack trace	A hypothesis tied to a specific change
Priority	Set by alert severity	Set by real customer impact
Repeat incidents	Investigated from scratch	Matched to the earlier fix
Handover	A Slack scrollback	A timeline, an owner, and a ticket with evidence

A PR review that knows what the change is for.

A generic reviewer sees only the diff. Guided Review starts from the same context every agent sees, sizes the risk, runs specialists in parallel, and posts only findings backed by evidence.

Pull request → Context

what every agent sees

PR diff

Full body of every modified file

Product requirements

PRDs and specs: what the change is meant to do

Feedback loop

Learnings from human review comments

Unified Context Layer

Service graph, cross-repo consumers, API contracts, topics, past incidents



Diff Blast Radius

how risky is this change?

Risk

Change type, criticality, incident hotspots, public surface, callers reached

+ Complexity

Diff size, cognitive complexity, concurrency, files and symbols touched, traced through the AST code graph

LOW

MEDIUM

HIGH

↓ Tier sets reasoning effort

Specialists

recall-first · parallel

Bugs

Security

Performance

Team standards

Run in parallel, each with its own checks. Only findings backed by evidence are posted.

↓ Risk tier, evidence, and comments

Gate

act, don't just advise

Auto-approve

LOW, and every comment resolved

CI check red

While a BLOCKER is open

Approval withdrawn

On a new commit or a reopened comment

Merge →

Evals

Production bugs as input. PRs whose bugs shipped, paired with the PR that fixed them. Every prompt and threshold change ships through evals.



Replay every config. Models and prompts on the bug-introducing commit.

QUESTIONS FOR THE CTO

01 **How is your review system compounding every day?**

02 Is it reviewing like a senior engineer who knows the product, or like a linter that only sees the diff?

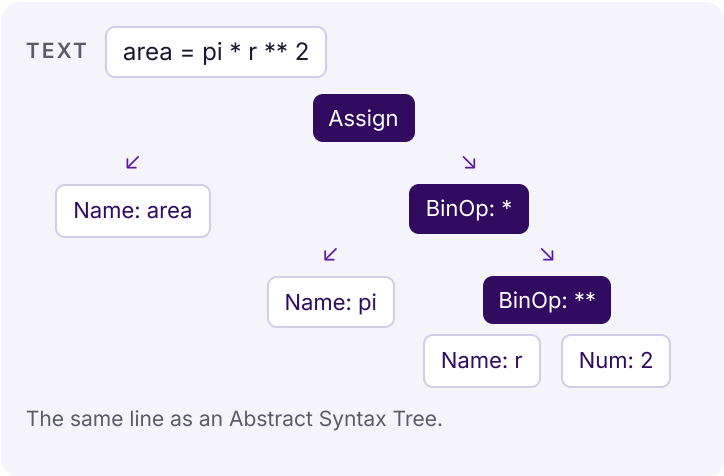
03 Does every human review comment make the next review better?

Blast radius needs structure, not text.

Line-based reviews and flat text parsing with regular expressions miss logical scope, data flow, and intent. The result is either a flood of false positives or shallow style checks. Guided Review parses every change into an Abstract Syntax Tree (AST) and enriches it into a code graph, so it can follow a variable through its real scope, see every caller a change reaches, and size the blast radius precisely.

HOW CODE BECOMES A GRAPH

- 1 **Parse.** Tree-sitter, Clang, or native parsers such as Python's ast module turn source into structured nodes.
- 2 **Normalize.** Punctuation, comments, and formatting are dropped, leaving pure structural logic.
- 3 **Graph.** The tree becomes a control-flow and abstract syntax graph that tracks how data moves between functions and variables.



CAPABILITY	TEXT-BASED (REGEX, BASIC LLM)	AST CODE GRAPH
Blast radius	Guesses from file names and text matches	Walks every caller, consumer, and data path the change reaches
Scope and variables	Confused by the same name in different scopes	Follows each variable through its real lexical scope
Token use	Pastes whole files into the prompt	Extracts only the changed functions and their dependencies
Automatic fixes	Higher risk of broken syntax	Codemods guarantee valid syntax
Security	Misses cross-file issues	Traces data paths for taint-analysis vulnerabilities

AST LAYER · THE GUARDRAIL

Deterministic rules catch structural errors, missing parameters, unused imports, and pattern violations.

LLM LAYER · THE EXPLAINER

Receives a compact serialization of the AST and writes contextual critique of design and intent without wasted tokens.

Dead code

Flags imports and functions no node ever references.

Semantic clones

AST fingerprints catch copied code with renamed variables.

Coding agents

Incremental AST updates keep coding agents grounded in real fields.

Every teammate gets an analyst, in plain English.

DATA & ANALYTICS PRODUCT EXECUTIVES

A query is only as good as its definitions. The Data Analyst reads the same Company Brain as every other agent, so it knows which table holds signups, what "active team" means at your company, and which dashboard the CEO already trusts.

CONTEXT LAYER

Warehouses	Snowflake, Databricks, Google BigQuery
Metric definitions	Codified once
Databases	PostgreSQL, MongoDB
Emails	Customer and team threads



DATA ANALYST
Writes the query
Runs it against the warehouse
Explains what the answer means
Delivers to Docs, Slack, or Teams

Why context has to compound

Every confirmed definition and answered question is written back to the Brain, so the next answer starts further ahead.

Ambiguous metric definitions
The same metric can mean different things to different people and table or column names confused by AI.

Aggregating and validating context manually is expensive
Significant time and effort go into gathering and validating it manually.

Context is perishable and goes stale quickly
As data and logic evolve, previously defined context falls out of sync.

Coverage is incomplete and hard to maintain
Critical business knowledge is often missing or never fully captured.

01 Self-service analytics
"How many teams that signed up in July are still active?"
Ask a data question in plain English. The agent writes the query, runs it, and explains what the answer means.

Snowflake Databricks PostgreSQL

GROUNDED IN YOUR WAREHOUSE

02 Metrics standardization
"Define active team once, and show me which dashboards disagree"
Codify definitions once so the whole company speaks one number, instead of three dashboards disagreeing about revenue.

Looker Snowflake

GROUNDED IN YOUR WAREHOUSE

03 Report drafting
"Turn last week's retention query into a note I can send the CEO"
Turn raw queries into a narrative someone can read and act on, rather than a table they still have to interpret.

Google Docs Slack

GROUNDED IN YOUR WAREHOUSE

04 Morning insights
"Every morning, tell me what moved more than 10% and why"
Get fresh findings delivered on a schedule, so a shift in the numbers reaches you before someone asks about it.

Slack Microsoft Teams Looker

GROUNDED IN YOUR WAREHOUSE

An LLM connected through MCP is not enough for analytics.

Connecting a general model to your warehouse through MCP gives it access, not reliability. In coding, failures announce themselves. In analytics, a query can run perfectly and still be wrong.

CODING

Failures are loud

Compilers reject bad syntax, tests fail, programs crash. The harness routes those signals back so the agent can retry.

ANALYTICS

Failures are silent

A polished number can rest on the wrong logic. Ask again tomorrow and the agent may take a different path to a different number, with no way to know which to trust.

THE FIX

A specialized Analytics Harness: a deterministic execution layer that applies the right context at the right moment, so the same question gets the same answer today and next year.

Six jobs the harness owns

01 Interpret the question

Resolves entity, grain, and time window before any SQL is written. Reuses approved SQL when the question was answered before.

02 Select context

Feeds only the schemas, metric definitions, and past queries that govern the question. Too little invites hallucination; too much confuses the model and bloats cost.

03 Enforce access

Keeps the model inside permitted tables and rows, and masks PII or other regulated columns.

04 Optimize the path

Picks the most efficient sources, queries, and execution order, so tokens go to answering instead of hunting for data.

05 Plan and execute

Maps the question across every system that holds relevant data, flags missing or ambiguous context, and carries definitions through each step.

06 Verify and repair

Checks the result against what the context says should be true. When a check fails, traces the cause and routes the fix for review.

WHY IT COMPOUNDS

Every job above draws on the Company Brain. Each approved query and confirmed definition is written back, so answers get faster, cheaper, and more consistent over time. Company context is a must-have, not an add-on.

Build the brain once. Every agent gets smarter.

The SRE Agent, Guided Review, and the Data Analyst work for the same reason: each starts from shared company context instead of a single tool's slice. Each run writes back what it learned, so the next agent starts further ahead.

01

Connect

Link GitHub, Slack, Jira, and your observability stack. Permissions carry over from each source.

02

Start with one loop

Turn on PR Reviews or the SRE Agent. Both show value on the first day of real traffic.

03

Compound

Add agents across product, engineering, and analytics. They all read from and write to the same brain.

Explore it yourself

XHawk is usage based. Connect a repository, turn on your first agents, and pay only for the work they do.

Talk to our team

Learn how teams like yours use a Company Brain to accelerate team velocity.

**Every company building software
will need a context layer.**
Build your
AI-Native Operational System.

xhawk.ai/brain →