CSE 5911 — Team 8

REBOOT

Retrieval Enhancement Based On Observed Trends

User Demo 3

Spring 2026 · 99P Labs / Honda Research Institute USA

Team Members

Pratham Kancherla

Harry Manzler

Jack Lualdi

Jacob Uy

Sponsor

99P Labs / Honda Research Institute USA

Agenda

1. **Project Recap**
   Problem, solution, and what we built
2. **Demo 2 → Demo 3 Recap**
   Iteration changes and new components
3. **Phase 3: Tuning & Stretch**
   Confidence decay, eval harness, and adaptation
4. **Evaluation Results**
   LLM-as-judge framework and retrieval metrics
5. **Live Demo**
   Ingestion, graph, search, and feedback loop
6. **RTM & Regression Suite**
   Use cases, owners, and test status
7. **Sprint Schedule & Release Plan**
   Updated 14-week plan and final backlog
8. **Hand-off & Future Work**
   Deliverables, docs, open issues, next steps

The Problem

Why existing retrieval systems fall short

Static Retrieval Pipelines

Retrieval systems such as RAG are typically static. As codebases evolve, the retrieval layer may surface stale or misaligned context — degrading LLM response quality over time without any visible warning.

No Feedback Integration

Default systems discard real usage signals. Feedback is never used to improve retrieval layer.

Real Developer Cost

Engineers spend significant time re-prompting or correcting AI-generated code because retrieved context was incomplete or irrelevant, eroding trust in AI tooling.

Sponsor: 99P Labs / Honda Research Institute USA · Use case: Code & Engineering Search in IDE

Harry Manzler

Our Solution: REBOOT

REBOOT — adaptive retrieval middleware

REBOOT is adaptive retrieval middleware that learns from observed usage patterns, query reformulations, and feedback signals — continuously improving context quality without retraining the LLM or requiring manual labeling.

Adaptive

Retrieval strategy evolves via confidence-weighted memory nodes that reinforce on positive signals and decay exponentially over time.

Self-Improving

No manual labeling or model retraining required. The system improves autonomously from real developer interactions.

Transparent

Every retrieval decision is explainable via reboot_explain. Scores, weights, and confidence multipliers are stored and queryable.

Agent-Agnostic

Delivered as an MCP server. Claude Code, Cursor, Windsurf — any compatible agent connects without a custom fork.

Implementation Approach

Tech stack, tools, and team organization

Tech Stack

Frontend / Agent	OpenCode (any LLM agent with config)
Middleware	Python + FastAPI
Backend	Graphiti + Neo4j (Docker)
Ingestion	tree-sitter (function/class boundaries)
Persistence	SQLite (feedback event log)
Version Control	GitHub — feature branch workflow

Development Process

Trello

Backlog and user story tracking

2-Week Agile Sprints

Weekly sponsor and team meetings

AI Tools (Claude / Claude Code)

Used as research and code assistant

Demo 2 → Demo 3 Recap

What we built in the final phase

Evaluation Harness

Manifest-driven eval pipeline (eval/) with isolated Neo4j, LLM-as-judge scoring, and per-run artifact output. Enables repeatable, quantitative retrieval quality measurement.

Confidence Decay & Unit Tests

Configurable exponential decay in FeedbackLogger with decay anchors and a global ingest timestamp for untracked nodes. Unit-tested with pytest (test_feedback_decay.py).

Async Ingest + Incremental Mode

Ingest jobs run as background tasks with job IDs, status polling, and cancel support. Incremental mode skips unchanged files — reducing MCP timeouts on large repos.

Retrieval Metrics in Feedback Loop

Precision@K and MRR computed and logged at feedback time via retrieval_metrics.py, closing the loop between user signals and measurable ranking quality.

Visualizer & Graph Tuning

Integrated graphiti_visualizer at /visualizer. Tuned BFS depth and reranker selection per query type, aligning hybrid search weights to REBOOT SearchConfig recipes.

Expanded Query Classification

Added a 4th query type — debugging — with its own SearchConfig weight recipe, improving retrieval specificity for error diagnosis queries.

Phase 3: Tuning & Stretch

What we planned and what we delivered

Evaluation Harness

LLM-as-judge eval pipeline with manifest-driven runs, isolated Neo4j, and per-run artifacts. Produces judge scores + P@K/MRR per feedback event.

Delivered

Confidence Decay & Reinforcement

Exponential decay (configurable λ) with access anchors, unit-tested. Positive feedback: ×1.1 (cap 2.0). Negative: ×0.9 (floor 0.1).

Delivered

Expanded Parser & Async Ingest

Incremental ingest by file mtime, background job management with status/cancel endpoints, bulk path for empty-graph first runs.

Delivered

Visualizer & Graph Tuning

Knowledge graph visualization at /visualizer. BFS depth and reranker selection tuned. Hybrid search weights aligned to per-type SearchConfig recipes.

Delivered

Temporal Edges (Git History)

Tree-sitter episodes use Graphiti's native temporal model. Explicit git-log ingestion for historical commit edges was descoped in favour of eval harness work.

Deferred

Live Demo

End-to-end adaptive retrieval demonstration

1

Ingest

Call reboot_ingest against a target open-source repo. Observe episodes_added count confirming function/class-level nodes have been indexed into Graphiti/Neo4j.

2

Visualize

Navigate to /visualizer. Explore the live knowledge graph — nodes for functions, classes, and modules; edges showing relationships extracted by tree-sitter.

3

Search

Issue a query via reboot_search. Observe: query_type classification, SearchConfig weights selected, confidence-ranked results returned with query_id.

4

Explain

Call reboot_explain. Surface the classification decision, SearchConfig weights applied, raw vs. post-ranked scores, and confidence multipliers per node.

5

Feedback & Adapt

Submit reboot_feedback with a positive signal. Confidence updates in SQLite (×1.1). Re-run reboot_search and observe ranking shift on top result.

Evaluation Harness & Results

LLM judges first-query retrieval on real repo issues

Evaluation Approach

Manifest-driven: JSON config defines repo, issue set, and resolutions. Generated given SWE-Bench repo subset.

Judge and Query agents: Generate query from issue, then evaluate retrieved context relevance given repo snapshot and gold patch.

Isolated env: Separate Neo4j container per repo and eval run prevents cross-contamination.

Artifact output: Per-run summary.json with objects for each test case containing:
- query generation with justification
- our search results
- judge scores and rationale
- lists of key hits and missing context

Preliminary Results (at 20% partial ingest)

Avg Judge Score Judge's relevance rating	~10%
Post-hit Precision@10 No. of relevant results in successful queries	30%
Post-hit MRR Mean reciprocal rank in successful queries	0.67

5% → 20% ingestion coverage not associated with significant score increase - two explanations

Evaluation Analysis

Eval results surface flaws in underlying approach

Pitfalls and potential alternatives

Conceptual vs Code Graphs:
- Graphiti's conceptual graph may ignore code structures
- Custom entity types attempt to account for this
  - Hints, not constraints

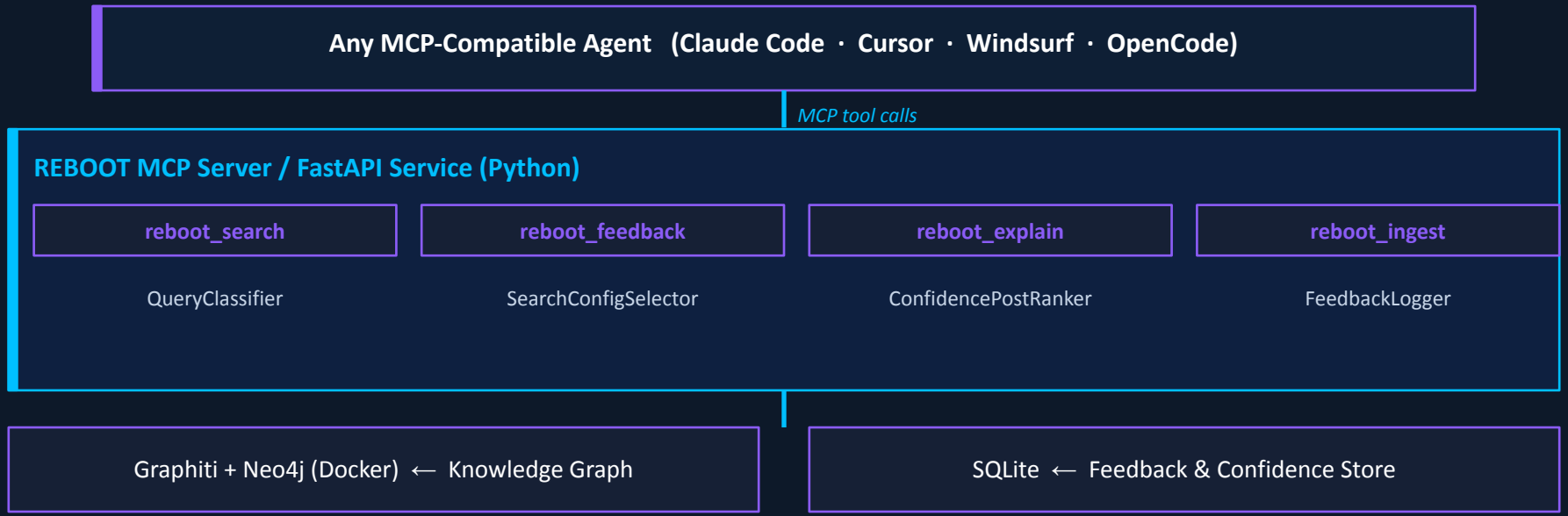
Search results mix nodes and edges:
- Queries like "where is X implemented" surface relationship-fact triples
- Relationships are numerous and thus crowd out actual code nodes

Search not tuned for code retrieval:
- Uses reciprocal rank fusion over semantic + keyword + graph traversal
- Simple grep over keywords performs better on classic search use case

Reliance on pre-loading knowledge and cost/speed constraints:
- Ingestion step adds friction regardless
- Scales poorly with large codebases
  - Thousands of nodes = hours of inference

System Architecture

End-to-end data flow through REBOOT



Ingestion Pipeline: tree-sitter (Python / JS / TS) · Async background jobs · Incremental mode (file mtime)

Requirements Traceability Matrix

Use cases, stakeholders, owners, and test status

Use Case	Requirement	Owner	Result
Agent can call all REBOOT endpoints via MCP	Agent MCP Access	Pratham	Pass
Submit feedback to update node confidence scores	Feedback	Jack	Pass
Explain last retrieval decision with scores	Explain	Jacob	Pass
Visualize knowledge graph at /visualizer	Graph Visual	Harry	Pass
Confidence decay and reinforcement over time	Confidence Updates	Jack	Pass
Retrieve confidence-ranked results via reboot_search	Search & Ranking	Pratham	Pass
Ingest repository with incremental support	Ingestion Pipeline	Harry	Pass

Sprint Schedule

14-week plan from kickoff to final demo

Phase	Dates	Focus	Key Deliverables
Phase 0	1/26–2/23	Setup & Research	Neo4j/Docker env · Graphiti research · node schema design · stub pipeline
Phase 1a	2/16–3/9	Core Build — Middleware	4 REBOOT components · FastAPI service · MCP server setup
Phase 1b	2/16–3/9	Core Build — Ingestion	tree-sitter ingestion · index target repo in Graphiti/Neo4j
Phase 2	3/9–3/23	Feedback Loop ← Demo 2	✓ Confidence decay/reinforcement · testing dataset · measurable adaptation
Phase 3	3/23–4/13	Tuning & Stretch	✓ Eval harness · async ingest · P@K/MRR metrics · reboot_explain
Phase 4	4/6–4/22	Demo & Docs ← Final Demo	✓ Sponsor demo prep · blog post · final documentation

Release Plan / Backlog

28

Total Features

Full backlog across all project phases

26

Completed

Features delivered and regression-tested

2

Deferred

Git temporal edges · richer feedback signals

7 tasks / sprint

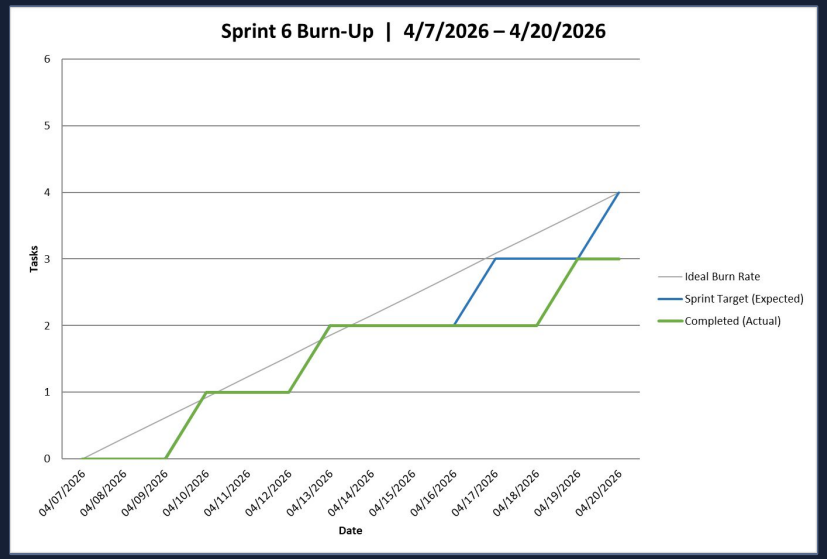
Avg. Velocity

Final measured team velocity

Burn-Up Charts

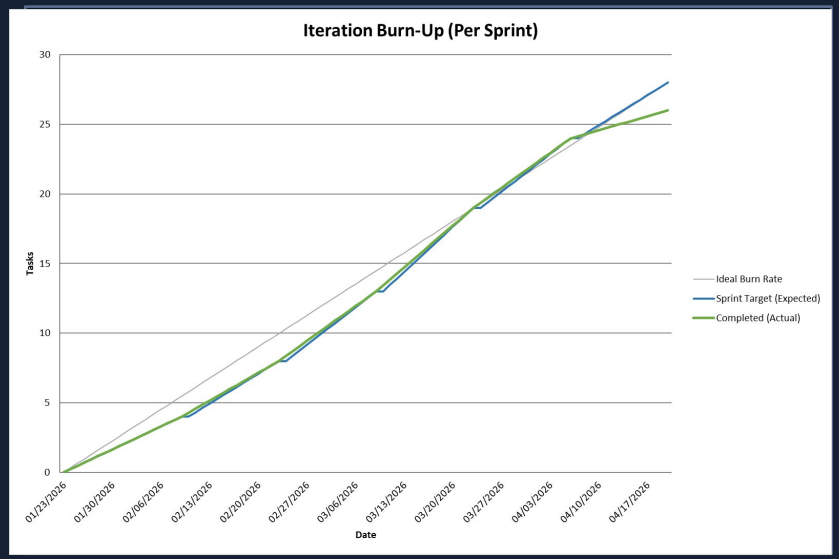
Sprint velocity and projected completion date

Iteration Burn-Up (Per Sprint)



Avg Velocity: 3 Tasks/Week

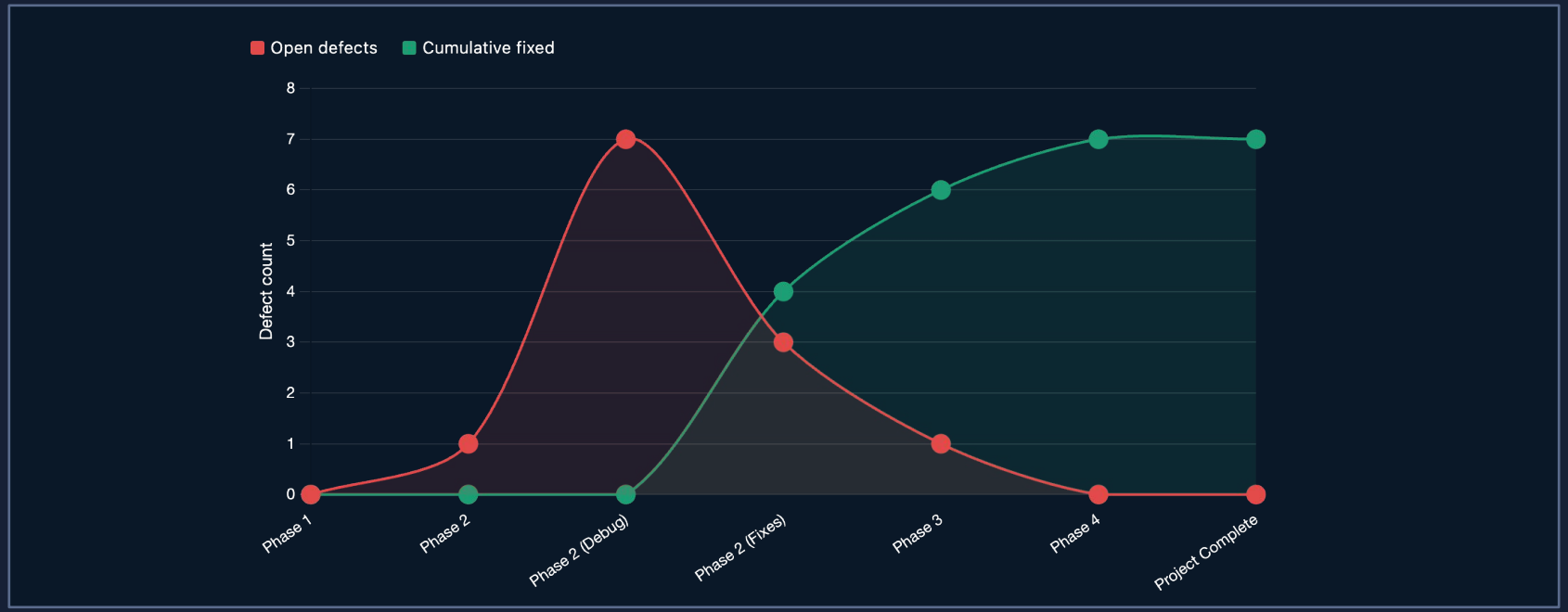
Release Burn-Up (Project-Wide)



Avg Velocity: 7 Tasks/Sprint

Defects Trend Chart

Net defects tracked over the life of the project



Project Hand-off

Handing off what we accomplished to our sponsor

✓ Working MVP

Fully functional REBOOT system: 4 MCP tools, confidence-weighted search, LLM-as-judge eval harness, and graph visualization. Connects to any MCP-compatible agent in minutes.

✓ GitHub Repository

Full codebase handed off to 99P Labs. Includes middleware, ingestion pipeline, eval harness, and all configuration. Repository link sent to sponsor.

✓ Documentation

ONBOARDING.md (clone + setup), PROJECT.md (architecture + endpoints), PLAN.md (module map), eval/README.md (running evaluation), .env.example + Docker Compose.

– Medium Blog Post (IP)

Technical post covering the REBOOT design, LLM-as-judge framework, and lessons learned. Published via 99P Labs on Medium.

Future Work

High-value extensions for the next team or iteration

Repo / Tenant Scoping

Search is currently global across all ingested repos. First-class multi-repo scoping would eliminate cross-contamination and make REBOOT production-ready for enterprise environments.

Richer Feedback Signals

Expand beyond positive/negative to reformulation, context_used, and context_ignored. Finer-grained signals enable more targeted confidence updates per node.

Git Temporal Edges

Mine git-log for commit history and surface it as temporal edges in Graphiti, making recently-changed code rank higher for factual queries about current state.

Human-Labeled Eval Set

Augment the LLM judge with a small human-labeled set (50–200 cases) for high-trust P@K/MRR baselines, making evaluation more credible for stakeholder review.

Agent Integration Hardening

Package REBOOT-optimized system prompt snippets, Cursor rules, and an automated smoke test suite that drives all 4 MCP tools in a scripted session end-to-end.

Thank You

Questions & Discussion

Team 8 — REBOOT · CSE 5911 · Spring 2026

Contact

Pratham Kancherla

pratham.kancherla@gmail.com

Harry Manzler

manzlerh@gmail.com

Jack Lualdi

jacklualdi@gmail.com

Jacob Uy

juyyyy.418@gmail.com