LikeLion@Berkeley X 99P LABS - SPRING 2026

Code Repo Knowledge Layer -> Git Abstraction Tool

A semester-long journey from developer onboarding tooling to a desktop app that makes version control accessible to everyone.

PART 1

Code Repo Knowledge Layer

Our initial project -- Repo Onboarding Buddy

The Problem We Started With

- Explicit knowledge -- code, comments, README. Already in the repo, but hard to navigate at scale.
- Structural knowledge -- 'if you change this, that breaks.' Implicit in code but hard to trace manually.
- Contextual knowledge -- 'why was this written this way?' Lives only in people's heads. Lost completely when team communication is absent.

Root Cause

Teams don't communicate around code -- so new members must read everything manually, and it takes too long.

Gap in existing tools

GitHub Copilot and search tools handle explicit knowledge only. Structural and contextual knowledge still require asking people.

Our goal

Build a knowledge layer covering all three types without leaving the editor.

Five Directions We Explored

1. AI Coaching Agent -- "Repo Onboarding Buddy"

Ask 'how does the data pipeline work?' and get an answer grounded in code structure, dependencies, and git history -- not just docs.

2. Usage Tracking Layer

Git history + AST analysis -> Core Assets, Stable components, Dead Code candidates. Bus factor alerts. Tableau treemap visualization.

3. Auto-Documentation

Living docs that auto-update when code changes. Four layers: one-liner, feature description, architecture context, business impact.

4. Interactive Code Map

Navigable graph of the entire repo. Click to drill down. Data lineage and cross-team dependency discovery.

5. Communication Gap Filler

Weekly AI newsletter of what changed. Cross-team impact alerts. 'Ask the repo' Slack bot.

One-line product definition

"A VS Code Extension that lets you understand every context of a repo without leaving the editor."

What We Actually Built

Repo Onboarding Buddy (Aaron, Sean, Clare)

- VS Code Extension UI with sidebar chat panel
- Context-aware Q&A: agent knows the currently open file
- AST-based code parsing with tree-sitter (no AI for structure)
- Dependency graph via NetworkX; impact analysis on right-click
- Hover metadata: one-liner, caller count, bus factor warning

Usage Tracking Layer (Jonghoon, Gwanho, Yunha)

- CLI tool 'repoknow scan' + FastAPI backend
- PyDriller for git history extraction (churn, ownership, bus factor)
- CodeQL for static import graph -> internal_edges.csv
- Scoring: Core Assets / Stable / Dead Candidates / NonCode
- REST API: /impact, /bus-factor-alerts, /top-imports

PART 2

The Pivot

Why we shifted from Repo Buddy to Git Abstraction Tool

Barriers We Encountered

- No shared DB or server -- external support was limited mainly to AI API access, so we designed around a local-only setup. SQLite became the practical ceiling for Phase 1.
- Cold start problem -- Q&A bot needs seniors to answer first, but they already use Copilot for code explanation.
- Weak value proposition -- 'What does this function do?' is already answered by Claude and Copilot. Not differentiated.
- Tribal knowledge gap -- 'Why was this written this way?' requires git history + PR discussion, hard to build reliably.

Ryan's Feedback -- 3/6 Meeting

"Think of this project as a new approach to version control -- A new generation of GitHub or Git."

Ryan's Feedback -- 4/3 Meeting

"How do we make vibe coding easier? People do not know what they want -- AI can give a high-level plan. Think about the vibe coders using phones or tablets to code."

New Vision

A next-gen git tool for non-technical users -- local-first, no server cost, AI-optional.

What Changed

CODE REPO KNOWLEDGE LAYER

Developer Tool

- Target: engineers in a team codebase
- Goal: faster onboarding & knowledge sharing
- Interface: VS Code Extension sidebar chat
- Requires: existing git & codebase context
- Needs: shared DB + server for team use

->

PIVOT

GIT ABSTRACTION TOOL

Non-Technical User Tool

- Target: vibe coders, designers, writers, students
- Goal: version control without knowing git
- Interface: Electron desktop app, plain language
- Requires: zero git knowledge
- Needs: local-only, no server, no cost

PART 3

Git Abstraction Tool

What we built -- and the 6 core AI features

What It Is

A cross-platform Electron desktop app that wraps git in a plain-language interface. No terminal. No git jargon. No learning curve.

Built local-first: full version control works offline. AI features are optional enhancements layered on top, never requirements.

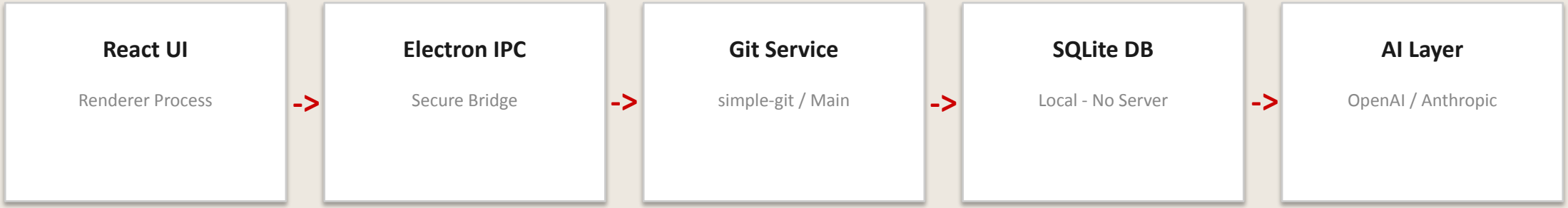
0
git knowledge required

6
AI-powered features

2
languages / EN+KO

- Save Progress -- one-click local snapshot (git commit)
- Full File Visibility -- tracked, staged, untracked in one panel
- Safe Operations -- explicit confirmation for every risky action; no auto force-push
- Pull Preview -- inspect incoming changes before merging
- Bilingual UI -- English & Korean, beginner-friendly terminology
- Bring your own AI key -- OpenAI or Anthropic, stored encrypted locally

How It's Built



Frontend

React + TypeScript, i18n (EN/KO), Tailwind -- renders in browser process

Backend

Electron main process handles all git ops, DB access, secret storage

AI

Modular provider adapters; OS-encrypted key storage; graceful degradation

6 Core AI Components

All optional | All privacy-preserving | All with graceful fallback

AI FEATURE

Auto Commit Message

When a user clicks “AI Suggest” button, the app reads the staged git diff and asks AI to produce one plain-language sentence -- no file paths, no git jargon.

> *"Updated the homepage background color and fixed typos in the contact form."*

The generated message is pre-filled for review. The internal AI summary is stored to power Weekly Report and Natural Language Undo.

- Reads staged diff, file paths, and change types (new/modified/deleted)
- 3-5 second timeout -- saving is never blocked by AI
- Per-project consent required before any diff is sent to AI
- Auto-Save mode: draft -> confirm -> commit in one flow
- Deterministic fallback message when AI is unavailable
- Stored AI summary powers the other three AI features

AI FEATURE

Weekly Report

At the end of any selected week, the app compiles git stats and stored AI summaries from each save into a readable narrative of what was accomplished.

Example Output

"This week you redesigned the portfolio homepage, fixed spacing issues on mobile, and added a new contact form. 14 saves across 3 active days -- 847 lines added, 312 removed."

Useful for status updates, client logs, or personal portfolios -- no git knowledge needed.

- Combines git stats with stored AI commit summaries for the week
- Shows total saves, files changed, lines added/removed, active days
- Cached by commit signature to avoid re-processing the same week
- Stats-only fallback (no AI narrative) when AI is not connected
- Distinguishes AI-summarized commits from message-only entries

AI FEATURE

Natural Language Undo

Instead of scrolling through commit hashes, users describe in plain language when they want to return to.

> *"Go back to before I changed the upload flow."*

> *"Restore to the state two days ago when the button was still red."*

AI searches recent commits enriched with their AI summaries, shows a file-level preview, then waits for explicit confirmation before doing anything.

- Searches up to 120 recent commits enriched with AI summaries
- Returns a primary match + 0-2 alternatives with confidence scores
- Shows which files would be restored and removed -- before applying
- User must explicitly confirm -- nothing applied automatically
- Fallback options when matched commit has no file-level delta

AI FEATURE

AI Feature 4 of 6

Untracked File Review

When untracked files exist and AI is connected, a 'Review untracked' button appears. The app classifies every untracked file: commit it to the repo, or delete it?

Example Recommendations

Commit: src/Button.tsx (94%) -- 'Looks like new source code, likely belongs in version control.'
Delete: node_modules/ (99%) -- 'Generated dependency folder, should never be committed to the repo.'

Stage all commit-recommended files in one click, or select delete-recommended files for instant removal.

- Two-tier decision: instant rule-based pass first, AI only for ambiguous cases -- fast and low token cost
- Rules auto-identify: build artifacts, caches, virtual envs, .env secrets, lock files, and source directories
- AI context: up to 100 files; up to 16 receive content snippets
- Adaptive timeout: 12s base + 220ms per file, capped at 45s
- Conservative fallback on timeout: unresolved files default to 'commit'
- Live UI updates as deletions complete; failed paths reported

AI FEATURE

AI Feature 5 of 6

File Insight

Instead of requiring users to understand every source file directly, File Insights gives a quick plain-language explanation of a selected file's purpose, behavior, and relationship to the rest of the project.

> *"What does this file do in the project?"*

> *"Why is this file related to the upload flow?"*

AI reads the selected file, recent commit history, and related file candidates, then explains the file's purpose without changing anything.

- Summarizes the selected file in plain language
- Explains the file's role and main functionality
- Uses recent commit history to add project context
- Suggests related files the user may want to inspect next
- Read-only by design -- no files are modified automatically

AI FEATURE

AI Feature 6 of 6

Smart Conflict Resolver

When a pull or merge creates git conflicts, the app automatically surfaces a guided dialog -- no raw conflict markers exposed. Optional AI explains which version to keep and why.

AI Hint Example

"Keep your version -- it has the new layout changes. The incoming version only adds a minor color fix that's already in yours."

- Dialog auto-opens when conflicts are detected after pull/merge
- "Keep Mine" / "Keep Theirs" per file -- no diff syntax required
- AI explains the tradeoff for each conflicted file on request
- Once resolved, AI can suggest the merge commit message
- Visual progress: resolved files show a checkmark badge
- Abort button always visible to cancel the entire merge safely

6 Core AI Components at a Glance

FEATURE 1

Auto Commit Message

Turns staged diffs into a plain-language save description. Never blocks saving. Stored summaries power the other features.

FEATURE 2

Weekly Report

Generates a narrative of the week's work -- what was built, how many saves, across how many days. Stats-only fallback without AI.

FEATURE 3

Natural Language Undo

Describe when to return to in plain English. AI finds the commit, previews the file changes, and waits for your confirmation.

FEATURE 4

Untracked File Review

Classifies untracked files as commit or delete. Rule-based pass first, AI handles ambiguous cases. Stage or delete in one click.

FEATURE 5

File Insight

Click any file for a plain-language explanation of what it does, how it works, and which related files connect to it.

FEATURE 6

Smart Conflict Resolver

Guided dialog for merge conflicts -- file by file, plain language. AI explains which version to keep and drafts the merge message.

SUMMARY

Bringing version control to everyone -- not just developers.

Code Repo Knowledge Layer

Developer onboarding | VS Code extension

->

Git Abstraction Tool

Version control for everyone | Electron desktop app