GOVERNED AI ENGINEERING

Why CodeLead

AI can write code. The hard part was never writing code. It's changing a system you half-understand without breaking what already works - and being able to prove you didn't.

MAP	GATE	PROVE
Understand the real change surface	Refuse unsafe or out-of-scope changes	Validate and record evidence

THE MESSAGE

AI coding agents, held by evidence.

CodeLead is a local-first layer that turns AI coding models into a governed engineering process: every change mapped, scoped, executed in small increments, checked before it touches disk, validated, and recorded as evidence.

0 unsafe applies

across 350+ benchmarked runs - every refusal has a written reason

~3 minutes

governed change on a 17-year-old C# codebase, local model, build green

100% local

runs on your machine, on open models you choose - code never leaves

$ codelead /implement "show the page title above the game" → model proposes config rewrites → patch gate refuses the out-of-scope surface → retry constrained to index.html → applied · build green · evidence written

A real run: a 4-billion-parameter local model building a Pong game, one governed increment at a time. Near the end it tried, twenty-four times, to rewrite the project's config files to add a page title. Every attempt was refused; the game kept working. The smaller the model, the more the governance matters.

THE PROBLEM

Coding agents demo on greenfield. You live in brownfield.

Agent demos are new apps built from empty folders. Your reality is a codebase that's old enough to have opinions: files nobody dares touch, tests that don't exist, conventions that live in one veteran's head, and consumers you'll discover the hard way. Point a raw agent at that and you get confident edits to the wrong files, a two-thousand-line diff you're supposed to review, and no way to answer the only question that matters: what did it actually see, change, and check?

The industry's answer is bigger models. But capability was never the bottleneck - trust was. A model with repo access and no governance is a liability whether it has four billion parameters or four trillion.

WHY CODELEAD

Five things it does that your agent doesn't

1 · It knows what your change actually touches - including the file you forgot

Before generating a line, CodeLead computes the change surface from the code itself: a symbol and dependency index, interface closures, and co-change obligations. If your change to a library must also land in a base class three projects away, that obligation is named up front - and a patch that skips it is refused, not merged.

> **EVIDENCE** Asked for one aggregate method on a 17-year-old data-access library, CodeLead flagged a fourth file never mentioned in the request - RepositoryBase, in another project - that a human reviewer had once missed. Retrieval benchmark: 0.97 mean core-recall, measured, re-scored after every change.

2 · Nothing reaches disk unchecked

Model output is treated as untrusted input. Every proposed patch passes deterministic gates - scope, path safety, plan compliance, syntax and convention sanity, repair effectiveness - before it can be applied, and every refusal is recorded with a written reason. The gate does not get tired on file four thousand, and it does not get charmed by confident output.

> **EVIDENCE** 350+ benchmarked runs to date; zero unsafe applies, ever. During the original brownfield experiments a local model hallucinated file contents for fifteen rounds - not one bad patch reached disk.

3 · Small increments, real commits - the way software has always been built

Humans hold a big picture and execute one small, completable thing at a time. CodeLead works the same way: it decomposes work into increments, each one built, validated, and committed on its own. You get a readable git history instead of a mega-diff, a resume point at every step, and failures that name the increment that caused them - with everything before it still green.

> **EVIDENCE** Head-to-head on the same tasks and the same local model: the one-big-build approach produced 0-of-9 working apps; the incremental loop shipped working software as a sequence of verified commits, in roughly half the wall-clock.

Five things it does that your agent doesn't - continued

4 · It runs on your machine, on models you choose

Local-first is not a deployment option; it is the trust anchor. CodeLead runs against local open models by default - anything your hardware serves - and treats remote models as an explicit, policy-gated escalation with a disclosure log. Governance is what makes small local models viable: decomposed, scoped, example-anchored tasks turn a model that is occasionally right into one that is boringly reliable.

> **EVIDENCE** The Pong game was built by a 4B model - small enough to run on a phone - in about forty governed machine-minutes on consumer hardware. Nothing left the machine.

5 · Every change explains itself, forever

Each run closes into a structured evidence record: the request, what was retrieved, the surface, every gate verdict and refusal, validation results, approvals. Six months later you - or an auditor - can walk requirement → change → check → validation → approval without asking anyone. Paired with the Project Knowledge Base, CodeLead keeps learned commands, conventions, invariants, and decisions with a trust level and a citation.

> **EVIDENCE** The evidence chain is the same instrument this document is written from: every number here traces to a session log, a benchmark report, or a git history on the founder's machine.

THE LOOP

How a change lands

01

Map

Index the code; compute the change surface and obligations.

02

Plan

Decompose into small increments; approve the plan first.

03

Pin

Where tests are thin, lock in current behavior first.

04

Generate

The model writes the scoped, example-anchored patch.

05

Gate

Deterministic checks; out-of-scope or unsound means refused, with reasons.

06

Validate

Build, tests, pins and behavior probes at the level you would check yourself.

07

Evidence

Commit plus structured record; knowledge captured for next time.

One increment of many. A failed increment stops itself - bounded retries, honest terminal states - and never takes the finished increments down with it.

FOCUS

Built for brownfield

Legacy code is where AI coding either earns trust or loses it. CodeLead's brownfield discipline is the one good engineers already use, mechanized: understand before editing (system map, contract catalogue, history mining for what actually changes together), pin current behavior with characterization tests where the safety net is thin, make the smallest change that honors the diagnosis, validate at the level the problem was observed, and land with the reasoning attached. Tribal knowledge - the conventions and invariants that live in comments and veterans' heads - gets extracted into the knowledge base as claims, and promoted to trusted only when evidence confirms them.

> EVIDENCE
>
> The design bet, stated plainly: enterprises don't block AI coding because models are weak - models improve monthly. They block it because nothing is auditable. CodeLead is the audit trail, the gate, and the method; the model is a replaceable worker behind it.

ALSO

Greenfield, the same way

New apps get the same machinery, chained: a walking skeleton first, then one capability per increment, each gated, validated (including whether the UI actually renders and responds - not just compiles), and committed. Where other tools hand you one unreviewable generation, CodeLead hands you a git history a human can read - and a resume point if increment seven goes sideways at 2 a.m.

AUDIENCE

Who it's for

Solo founders & indie builders

You're the architect, the reviewer, and the on-call. CodeLead lets cheap local models do the typing while you keep the judgment.

• Ship features as verified increments, not roulette-wheel mega-diffs.
• Run on hardware you own - no per-token bill for iteration, no code exfiltration worry.
• Come back after a week and the evidence trail - not your memory - says exactly where things stand.

Enterprise developers

You maintain the systems the demos avoid. CodeLead is built for your Tuesday.

• Change surfaces that include the co-change three repos away - before review, not after the incident.
• Characterization pins so “did I break anything?” becomes “the pins are green.”
• Refusals with written reasons you can show your tech lead - and an evidence chain your auditor can follow.

For CTOs: the two-paragraph version

Your engineers are already using AI on your code - the only question is whether it happens inside a governed, auditable boundary or in the shadows. CodeLead gives you the boundary: local-first execution, deterministic safety gates that no model can talk its way past, and structured evidence for every change - requirement to approval - exportable as audit packs.

It is model-agnostic by design: as models improve, your governance, knowledge base, and evidence compound instead of resetting.

FOR CTOS

Governance that compounds as models improve

The method is protected - three provisional patents filed across the governed core engine, the enterprise knowledge and control plane, and evidence-derived knowledge - and the roadmap runs from individual developers today to team policy, legacy modernization programs, and an organization-wide knowledge base whose every answer cites its proof. Adoption starts small: one codebase, one developer, read-mostly - and earns its way up on evidence, which is exactly how engineers will judge it too.

EVIDENCE

Every figure in this document - refusal counts, 0.97 recall, 350+ runs, the three-minute chain - is intended to be reproducible from session logs, benchmark reports, and git histories generated by CodeLead's own evidence discipline. We measure before we claim; it is the same rule the product enforces.

CodeLead is in active development by Logos Vectis LLC - former Amazon and Microsoft engineers with 25+ years in industry, including principal engineer and chief architect roles. The prototype and benchmark program are running today; ask for a live demo - it fits on a laptop.