

CTX Engine: building context for coding agents and testing whether it helps

Creator and technical lead · 2025-2026

A coding agent working in a large repository needs to discover which code matters, how the pieces connect, what constraints apply, and what evidence would justify its answer. I built CTX Engine to help with that work. The project combined search, dependency information, source-aware indexing and structured reading guides. It also became a substantial investigation into how to test the value of those mechanisms.

The central challenge was establishing a connection between better context and better work. Relevant files appearing in a result list were encouraging, but an agent still had to inspect them, identify the important facts, understand their relationships and produce a useful answer. Improving one step could leave the final outcome unchanged.

I led the architecture, integration, experimental direction and adoption decisions, working extensively with Codex on implementation and testing. This case follows how we built the context infrastructure, made its usefulness testable, and used the results to refine both the system and the evaluation methods. The strongest contribution was the combination: developing something ambitious while building increasingly discriminating ways to judge it.

Giving an agent a useful starting point

The work grew out of attempts to make an evolving software repository easier for AI assistants to understand. Repository maps and structured context representations preceded CTX. As the system developed, the practical interface became a command-line request: an assistant supplied a task question and received a reading guide.

The guide combined keyword search, semantic search and dependency information. It could point to relevant source and documentation, related components, likely follow-up reading, missing evidence and verification steps. This gave an agent a proposed route into the work. It still needed to inspect the underlying sources and use judgment about what they established.

That distinction shaped the interface. A task could require further investigation, a report of conflicting evidence, or a qualified answer. The system returned structured recommendations for those situations. Later versions made evidence obligations more explicit, including facts an answer should address and claims that the available material did not support. Those recommendations were guidance to the consuming agent; they did not intercept all of its subsequent actions.

Keeping the reading guide connected to the actual codebase required separate engineering. Source revisions and local edits could change what a search result meant. We worked on version-aware indexing, synchronization of changed files and explicit freshness states. If a rebuild unexpectedly produced an empty index, recovery could retain the previous searchable data and identify it as stale. That preserved a useful recovery option while exposing the state the agent was actually reading.

These mechanisms mattered to evaluation as well as operation. A retrieval experiment could otherwise compare different source states, or pick up incidental editing-session signals that made a task easier. Authored fixture runs therefore suppressed unrelated live dirty-path influence. Source state, retrieval behavior and test expectations had to be distinguishable before a score could support a meaningful conclusion.

Making a promising retrieval result answer a harder question

The early tests developed alongside the system. They moved from structural properties of repository maps toward whether an agent's first reading suggestions contained useful files. Later fixtures included recent workstream tasks and measured whether expected helpful files appeared near the beginning of the reading guide. This improved the question the tests asked, even where the retained record did not establish a passing result for the newer fixture sets.

A May 2026 internal benchmark gave a concrete favorable result. Across 30 code-navigation tasks from my own repository, the combined reading-guide approach found approximately 69% of the reference set of relevant files, compared with 34% for keyword search; 85% of its selected files were relevant. This supported the value of integrating retrieval with repository structure and task guidance.

The internal benchmark sharpened the question of whether better file finding helped an agent produce a better answer. It measured retrieval in an environment the system had been built around. Portability to unfamiliar repositories was a separate line of investigation that had begun earlier.

The external work made portability concrete. We built an isolated benchmark path that checked out the exact revision associated with each task, kept foreign repositories and indexes separate from daily CTX state, and converted selected files and source spans into the external evaluator's input format. A small adapter detail was especially revealing: leaving the model-patch field empty could make the evaluator fall back to the reference patch. Our context-only adapter supplied an explicit nonempty marker to prevent that fallback from producing misleading edit-location measurements.

The early external smoke results were weak. In two 14-task runs, keyword and vector-only modes each found four of 64 reference files. Those runs examined different configurations and tasks from the favorable internal benchmark, so they could not form a single before-and-after trend. They did identify portability problems: broad documentation and nearby tests could be found while the task's implementation, configuration or schema files were missed.

The work now had several concrete questions. Could the machinery run correctly? Could it locate useful evidence? Could an agent use that evidence? Did the combined method improve the result enough to justify its cost? Each question needed an observable outcome and a suitable comparison.

Building experiments around the agent's actual answer

The first explicit task-utility protocol was modest. It introduced ordinary-agent and disciplined repository-search controls alongside CTX methods. Tasks covered behavior tracing, preparation for implementation, work crossing component boundaries, operations and decision support. The protocol recorded inspected and changed paths, verification, task outcomes, preserved boundaries, time to confidence and unsupported assertions.

At that stage, outcome judgments were manual, and the protocol did not yet automate agents or conceal scoring keys. Its contribution was to define a comparison that could challenge the claim that CTX helped. Later harness work added automated execution, protected reference material and more explicit scoring. Those capabilities were developed over time, with different controls appropriate to each experiment.

I treated successful execution of the harness as its own result. A run could produce valid packets, dispatch agents and collect outputs while the answers remained poor. In one 50-task packet sweep, all 50 projections passed the check for exposure of hidden reference material; individual packets could still report that more evidence was needed. Separate small agent diagnostics showed weak answer quality despite useful evidence localization. That made the gap between preparing context and using it impossible to ignore.

The answer-scoring implementation examined required facts, critical facts, forbidden claims, verification information, risk topics and acknowledgment of missing evidence. Missing critical facts or specified unsupported claims could cap an answer's score. This gave the evaluator ways to reject an answer that looked complete while omitting something important or asserting something the evidence did not establish.

Scoring itself remained an engineering subject. A deterministic fact check depends on the task's expected facts, matching rules and rubric. Compliance with an expected answer structure could be measured separately from whether the answer contained the necessary source-backed content. A well-formed response was one observable property; factual usefulness required additional checks.

The comparison conditions also became more explicit. For the later sealed holdout, we fixed the source, model, prompt and scoring conditions. Reference answers were protected from trial agents, and the competing configurations used the same frozen source material. The experimental profile preserved the retrieval candidates and their order for the same query, while adding guidance about ownership, evidence and proposed changes. That guidance could influence which files an agent subsequently chose to inspect. This made the result interpretable as a comparison of those configurations under that protocol.

Cost and execution quality stayed visible alongside answer scores. Time, token use, missing measurements, failed dispatches and invalid outputs could change the practical meaning of a result. The harness needed to preserve those distinctions so that a favorable answer comparison could be assessed together with what it took to obtain it.

Using failures to change the context presentation

The internal work developed a standard configuration and a series of experimental alternatives for presenting evidence to an agent. We called this unified internal context, or UICTX. The alternatives remained explicitly selected experiments while the standard configuration stayed in ordinary use.

The experiments targeted specific problems. An early trial could steer tasks about other systems toward CTX's own internals. That exposed a problem with the experiment's task direction and required a repaired comparison. Another revision improved some tasks while underemphasizing a component that performed an external action. These were concrete reasons to examine which parts of a task the context presentation made salient.

A later revision showed that relevant metadata could exist and even be partly read without adequately shaping the answer. The next version made the primary implementation owner and supporting files more explicit, then asked the agent to account for proposed changed paths against those roles. The idea was to help it connect evidence to the actual place where work belonged.

These presentation experiments built on an existing answer contract that identified evidence obligations, missing information and possible response postures. Verification suggestions were distinguished from verification that had actually been performed. When evidence was insufficient, the agent could be directed to expand its investigation or state the limitation.

The development comparisons eventually produced a promising candidate. In one 36-task paired comparison, the experimental presentation recorded 21 wins, seven losses and eight ties or mixed results against the standard configuration. These were recorded semantic-review judgments; the retained summary does not identify who assigned the individual verdicts. The compared configurations both produced outputs for all 36 tasks; a separate repository-search reference run retained one quarantined timeout.

The favorable result came with a measurable cost: the candidate used about 9% more elapsed time and 7.7% more tokens on that comparison. It earned consideration for the next test. The standard configuration remained unchanged while the candidate faced a protected holdout.

Letting the holdout decide what to adopt

The holdout contained 24 tasks from the same repository, reserved from the candidate's iterative development. Its purpose was to test whether the promising presentation improved answers on material it had not been repeatedly revised around. Both the standard and experimental configurations would run, producing 48 trial outputs for comparison. This holdout and the separate 46-task suite described below used automated, rule-based answer scores to assign wins, losses and ties, checking expected facts, specified unsupported claims and other task requirements under a fixed tie band.

The first two attempts had execution, environment, retrieval or output-contract problems. We excluded them from the result used to judge adoption, repaired the run conditions and completed a clean rerun. The clean comparison collected and scored all 48 outputs, with no missing outputs, runner errors or quarantined compared trials.

The candidate recorded no wins. The standard configuration won three comparisons, and 21 were ties or mixed. This did not demonstrate the improvement required to replace the standard configuration. I retained the standard and redirected the work toward understanding the result.

A separate, broader internal suite compared the standard configuration, the experimental candidate and repository search across 46 tasks. The standard configuration recorded nine wins and one loss against search, with 36 ties or mixed results under coarse answer-quality scoring. The candidate also had favorable comparisons against search, but lost more comparisons against the standard configuration than it won.

That distinction was central to the adoption decision. Evidence that some CTX behavior helped relative to search did not automatically favor every new CTX design. Replacing the existing configuration required evidence about the replacement itself.

The results also had a clear scope. These were internal tasks and repository-derived expectations. They supported decisions within that setting; external transfer and broad software-engineering productivity remained open. The protected holdout strengthened the test of the candidate without turning it into an external benchmark.

Investigating why more evidence did not produce better answers

The broader comparison exposed a particularly useful discrepancy. On the 36 tasks with file-level evidence scoring, trials using the experimental presentation recorded opening more of the reference evidence, including more documentation, than trials using the standard configuration. These measurements came from paths recorded in trial outputs as opened or used as evidence. They concerned the agents' subsequent investigation, rather than an improvement to the initial retrieval candidate set. Across the full 46-task suite, that additional evidence coverage did not translate into better answer quality.

Several explanations remained plausible. An agent might receive useful evidence and fail to use it. Extra material might be relevant to the repository yet unhelpful for the actual question. The standard configuration might already supply enough information with less cognitive overhead. Some tasks might depend more on verification or honest treatment of missing evidence than on finding another file. The scorer itself might also be too coarse to distinguish meaningful differences.

The scoring review gave that last possibility substance. Every top-level trial outcome in the broader suite was marked partial with a score of 0.5. Pairwise judgments depended on a separate answer-quality measure, but that measure also placed many answers into identical buckets. Thirty-one of the 35 ties or mixed results between the candidate and standard configuration had exactly equal answer-quality scores.

This was enough evidence to question the measurement's resolution. It did not justify changing the completed verdict or declaring that the candidate would have won under a different evaluator. I kept the adoption result intact while making the next investigation more specific.

We added diagnostics that distinguished the evidence a method exposed, the files an agent opened, and the sequence in which useful evidence was reached. The implementation could examine irrelevant openings before the first relevant file, progress toward a sufficient evidence set and differences in category coverage. These measures made it possible to ask where the process was losing value.

The existing run could not supply every new measurement. It lacked per-file timestamps, so true time to the first useful file or sufficient evidence could not be reconstructed. It also lacked some records of the order in which the method presented its evidence, and token attribution was unavailable for the broader suite. Those gaps stayed explicit. Adding instrumentation improved the available diagnostic method without manufacturing observations for an earlier run.

An experimental evidence-ledger format followed, exposing source provenance, bounded exact reads, missing information and answer obligations more directly. A later interactive evaluation design also targeted the newly exposed gap. The ledger implementation and the design represented further work; their comparative advantage had not been established. They remained separate from the standard configuration and from the already completed results.

What this work demonstrates

CTX brought together context infrastructure, comparative agent testing and experimental judgment. The implementation included search and dependency integration, source-aware indexing, reading guides, recovery behavior, benchmark adapters, answer scoring and diagnostic instrumentation. The evaluations then exposed distinctions that individual component tests could not settle.

My contribution was to keep connecting the mechanism to the outcome it was meant to improve: choose a useful question, make the comparison interpretable, investigate failures in both the system and the test, and decide what the evidence justified adopting. Codex contributed extensively to implementation and testing throughout that work.

CTX did not become the broadly validated context platform I had aimed to build. It is now retained as a legacy system, with reusable parts requiring their own decisions. The demonstrated contribution remains substantial: a working body of context and evaluation infrastructure, bounded favorable results, and a sequence of engineering decisions shaped by increasingly demanding tests.

For future agent systems, this experience gives me concrete ways to investigate usefulness. I can separate finding evidence from using it, inspect whether a test rewards the intended behavior, and look for the point where a local improvement stops producing a better result. That is the connection between building with AI agents and engineering the conditions under which their work becomes useful and trustworthy.