

CTX Engine: evidence for testing whether context helps coding agents

Selected implementation and historical evaluation exhibits · curated September 16, 2026

CTX Engine combined repository search, dependency information and structured reading guides for AI coding agents. The engineering question was whether that additional context helped an agent produce a useful, source-grounded answer. Finding relevant files, completing a test run and improving the answer were separate outcomes.

These exhibits make three parts of that work inspectable: the comparison controls, defenses against misleading scores, and the evidence behind an adoption decision. Justin Sublette directed the problem selection, integration, evaluation requirements and interpretation; Codex contributed extensively to implementation and testing. The excerpts are retained implementation, and the results are historical observations, not newly executed tests.

All source excerpts below are verbatim at revision ef8546aaa7a79c006f75a68da027389912b1f2d1. Labels give repository-relative paths and source lines. Each code block is a contiguous excerpt; surrounding code is omitted. This September source snapshot preserves the mechanisms for inspection; the historical comparisons used their own frozen revisions.

1. Make the comparison interpretable

The candidate called “v5” changed how an internal context configuration presented evidence: which implementation owner mattered, how supporting files related to it, and how proposed changes should account for those roles. The intended comparison was against the standard configuration, called “canonical” in the implementation. A better answer should be attributable to the selected experimental configuration rather than an unnoticed change in tasks or source material.

The campaign builder loads its task bank from the chosen Git revision before checking task compatibility:

Source A: systems/ctx_engine/packages/internal_benchmarks/real_utility_freeze.py, lines 734-743.

```
task_bank_rel = _repo_relative_path(root_path, task_bank_path)
frozen_task_bank = _load_json_from_commit(root_path, source_commit, task_bank_rel)
compatibility = build_uictx_task_compatibility_preflight(
    frozen_task_bank,
    root=root_path,
    task_bank_path=task_bank_rel,
    source_commit=source_commit,
    suite=suite,
    task_ids=task_ids,
)
```

Its compatibility report also fingerprints the complete bank, selected tasks and scoring contract:

Source B: the same file, lines 333-335.

```
"task_bank_digest": _freeze_stable_json_hash(task_bank),
"selected_task_digest": _freeze_stable_json_hash(selected_tasks),
"scoring_contract_digest": _freeze_stable_json_hash(task_bank.get("scoring_contract") or {}),
```

These identities provide concrete things to compare when a run's conditions are questioned. They do not make task expectations correct automatically: the surrounding preflight treats missing source anchors as blockers and changed anchors as signals requiring compatibility review.

The v5 profile declares the scope of its intervention:

Source C: `systems/workstation_control/packages/ai_lab_runtime/repo_retrieval/uictx_profiles.py`, lines 258-264.

```
candidate_rows_preserved=True,  
canonical_order_preserved=True,  
default_runtime_changed=False,  
task_bank_changed=False,  
prompt_changed=False,  
scorer_changed=False,  
hidden_gold_used=False,
```

For a given query, the profile preserves retrieval candidates and their order while changing the presentation around them. An agent can still investigate differently after reading that presentation. Consequently, more subsequently opened evidence would not by itself show that the initial retrieval ranking improved. The explicit profile also keeps an experimental candidate separate from the default configuration; entering a comparison is not deployment approval. These declarations identify the intended controls, while the historical result records below establish what the selected comparisons reported.

2. Inspect what the evaluator actually rewards

An external benchmark adapter exposed a small but consequential measurement trap. CTX selected context rather than generating a code patch. Leaving its patch field empty could cause the evaluator to substitute the reference patch, making edit-location measurements misleading. The adapter supplies an explicit nonempty marker:

Source D: `systems/ctx_engine/packages/external_benchmarks/adapters/contextbench.py`, lines 382-385, inside `prediction_from_contexts`.

```
# Non-empty on purpose: ContextBench falls back to the gold patch when  
# this field is empty, which would make edit-location metrics misleading  
# for CTX context-only runs.  
"model_patch": "CTX_CONTEXT_ONLY_NO_PATCH\n",
```

This is an implemented defense against measuring an answer supplied by the test itself. It makes the context-only submission explicit; it does not establish the system's external benchmark performance.

The internal answer scorer addresses a different problem: a response can look complete while missing a decisive fact or making an unsupported claim. It matches specified required and critical facts, forbidden claims, expected verification information and acknowledgments of missing evidence. The following branch limits an otherwise favorable score:

Source E: `systems/ctx_engine/packages/internal_benchmarks/contextbench_mirror_scoring.py`, lines 1141-1154.

```
raw_quality_score = _clamp01(sum(positive_values) / len(positive_values)) if positive_values else
answer_correctness
answer_quality_score = raw_quality_score
caps = _as_dict(scoring.get("caps"))
applied_caps: list[dict[str, Any]] = []
if critical_details and critical_fact_recall is not None and critical_fact_recall < 1.0:
    cap = float(caps.get("missing_critical_fact_max_score", 0.6))
    applied_caps.append({"cap_id": "missing_critical_fact_max_score", "max_score": _clamp01(cap)})
    if answer_quality_score > cap:
        answer_quality_score = _clamp01(cap)
if forbidden_matches:
    cap = float(caps.get("hallucinated_critical_fact_max_score", 0.4))
    applied_caps.append({"cap_id": "hallucinated_critical_fact_max_score", "max_score":
_clamp01(cap)})
    if answer_quality_score > cap:
        answer_quality_score = _clamp01(cap)
```

For illustration, an otherwise 0.9 answer missing a critical fact would be capped at 0.6 under these defaults; matching a specified forbidden claim would cap it at 0.4. Those numbers illustrate the branch, not a recovered historical trial. The rules depend on authored expectations and matching logic. A mention of a verification command is not proof that the command ran, and a deterministic fact match is not general semantic adjudication. Inspecting those differences was part of improving the evaluation itself.

3. Let the held-out result determine adoption

The candidate first earned a stronger test, then failed to establish the improvement needed to replace the standard configuration. The table preserves that sequence without combining different populations into a single success rate.

Source F: `docs/systems/ctx_engine/Operations/CTX_Legacy_Benchmark_Result_Capsules_2026-09-04.md`, capsules 3-5. “Win” means the selected comparison’s answer judgment, not successful completion of an engineering task.

Comparison	Population	Result	Decision meaning
Development comparison: v5 versus standard	36 paired tasks	21 v5 wins, 7 standard wins, 8 ties; v5 used 9.0% more elapsed time and 7.7% more tokens	Promising enough for held-out review; no default change.
Sealed internal holdout: v5 versus standard	24 paired tasks; 48 outputs	0 v5 wins, 3 standard wins, 21 ties/mixed; no material v5 lifts	No promotion candidate.
Separate broader suite: standard versus repository search	46 tasks	9 standard wins, 1 search win, 36 ties/mixed	Bounded favorable evidence for the standard configuration.
Same broader suite: v5 versus standard	The same 46 tasks	4 v5 wins, 7 standard wins, 35 ties/mixed	That broader result also did not favor replacing the standard.

The development comparison retained semantic-review verdicts; its aggregate does not identify who assigned each verdict. The holdout and broader suite used automated rule-based answer scoring. The holdout reserved tasks from the same repository, rather than testing transfer to an unfamiliar repository. Both development comparison arms completed all 36 tasks; a quarantined timeout belonged to the separate search reference arm. The broader suite produced 138 outputs across three configurations, not 138 independent tasks.

The distinction mattered: evidence that standard CTX helped relative to search did not establish that every new CTX configuration improved on it. The completed verdict stayed intact while investigation moved to the gap between acquiring evidence and using it in an answer. The broader analysis also identified coarse score resolution and missing token attribution; it did not establish a general productivity or efficiency gain.

The retained September 4 capsule contains aggregates, verdicts and source fingerprints. Historical task answers and raw run archives were deliberately removed. This exhibit therefore supports inspection of the implementation and recorded decision, but does not offer downloadable historical trials or independent replay. A future trace demonstration would be a separately dated experiment.

Together, these examples show the contribution: engineering an interpretable comparison, identifying ways a test could reward the wrong thing, and allowing the result to constrain adoption while preserving the useful work already demonstrated.