

From Prompt-Level Goals to Prefix-Conditioned Execution: A Position Paper on Incremental Task Execution and Cumulative Audit

RM369

Abstract

Language-model agents commonly receive a complete instruction, plan or generate a result, and then optionally critique it. This design can leave a gap between understanding a request and reliably satisfying every constraint, particularly when requirements are numerous, late-arriving, or mutually dependent. We propose Prefix-Conditioned Execution and Cumulative Audit (PECA), a control-loop design in which an agent incrementally interprets a prompt, updates a traceable ledger of active requirements, executes reversible microtasks when sufficient semantic context exists, and audits the working artifact against the entire interpreted prefix after each meaningful step. Later text can revise earlier interpretations, and externally checkable tests should be preferred over self-approval. The proposal is related to, but distinct from, iterative self-refinement, tool-interactive critique, reason-and-act agents, and process verification: its defining unit is the evolving instruction prefix coupled to changes in a persistent artifact. We make no empirical performance claim. Instead, we specify falsifiable hypotheses, baselines, ablations, outcome measures, and safeguards for a controlled evaluation. The central question is whether prefix-conditioned execution and cumulative audit improve requirement satisfaction enough to justify their added inference and tool costs, and under what conditions they fail.

1. Motivation

A prompt is not merely a goal label. It is a sequence of requirements whose meaning becomes clearer as clauses, exceptions, examples, and constraints arrive. A system that reads the whole prompt and produces one polished answer may satisfy its most salient request while omitting a less salient restriction. A final self-critique can catch some omissions, but it occurs after the artifact has already been shaped around an initial interpretation.

Consider: “Draft a launch announcement. Make it enthusiastic. Do not claim the product is available yet. Say early access opens next month. Keep it under 60 words.” An appealing draft can still fail by implying current availability, missing the early-access date, or exceeding the word limit. The issue is not only whether the model can understand the final prompt; it is how that understanding controls work while the artifact is being built.

We propose a simple design principle: make incremental interpretation and cumulative verification part of the execution controller, rather than relying exclusively on a single end-goal representation and a final review. Each new meaningful instruction unit updates a live requirement model. The agent takes a small, auditable action, tests the current artifact against all requirements understood so far, and repairs it before proceeding. This can expose omissions earlier and maintain traceability from instruction to implementation.

This is a systems hypothesis, not a claim that decomposition alone eliminates reward hacking. A

verifier can be wrong, the executor and reviewer can share a blind spot, and excessive intermediate work can waste resources or create regressions. The purpose of this paper is to define a testable architecture and identify evidence needed to assess it.

2. Relation to prior work

PECA should be viewed in the context of established work on acting, reflection, critique, and step-level evaluation—not as a replacement for, or proof of superiority over, those methods.

Interleaved reasoning and action. ReAct interleaves model-generated reasoning and actions, using environment observations to update subsequent behavior [1]. PECA shares the execution-feedback loop but makes progress through the instruction prefix and a requirement ledger explicit control inputs. ReAct does not, by itself, require an audit of the current artifact against every active instruction after each step.

Iterative self-refinement and agent reflection. Self-Refine generates a candidate, obtains self-feedback, and revises it iteratively [2]. Reflexion uses verbal feedback and episodic memory to improve later attempts [3]. These approaches motivate deliberate feedback loops. PECA instead specifies when the loop is triggered: after a new meaningful portion of the instruction becomes actionable, while maintaining a cumulative, source-linked set of requirements and a persistent work product.

Tool-based critique and process verification. CRITIC uses external tools to critique and progressively amend model outputs [4]. Process-reward work evaluates intermediate steps rather than relying only on final outcomes [5], while Socratic Self-Refine uses verifiable sub-question/sub-answer pairs and targeted re-solving [6]. Several 2026 preprints make the overlap more direct. Certificate-Gated Prefix Acceptance uses a verifier to gate proposed prefixes in hard-constrained sequential decision problems [7]; BRA-Audit allocates audits using cumulative exposure in multi-agent systems [8]; and recent agent-evaluation work inspects execution trajectories and the earliest consequential failure [9]. These are adjacent to PECA’s use of prefix state and intermediate auditing. The distinction proposed here is narrower: the controller’s progress variable is the evolving natural-language instruction prefix, reconciled into a provenance-bearing requirement ledger that governs changes to a persistent task artifact. Whether that distinction yields useful behavior beyond prior controller designs is an open empirical question, not a novelty claim.

3. Proposed architecture

We call the design **Prefix-Conditioned Execution and Cumulative Audit (PECA)**. The name denotes a controller pattern, not a trained model or a claim of a new base-model capability.

Let a prompt be segmented into meaningful units $u_1, \dots, u_n$. These units may be clauses, sentences, list items, code requirements, or other semantic chunks. They are not necessarily individual characters or tokens. At step i , the system maintains (1) an interpretation of the observed prefix $p_i = (u_1, \dots, u_i)$, (2) an active requirement ledger R_i , (3) a persistent artifact or environment state S_i , and (4) evidence from audits and tools.

The loop has five stages:

1. **Update interpretation.** Incorporate the next unit, including its effects on earlier text. Identify ambiguity, negation, exceptions, dependencies, and conflicts.
2. **Reconcile requirements.** Update a ledger whose entries include a requirement, its source span, status (active, superseded, conflicting, or unresolved), dependencies, and a proposed verification method.
3. **Choose a meaningful microtask.** Select a reversible action that advances the artifact only when enough context exists. If the unit is incomplete or changes meaning but is not yet actionable, update state without forcing an implementation step.
4. **Execute and audit.** Perform the action in a sandbox or other appropriate environment. Check the resulting artifact against all active

requirements in R_i , not only the newest one. Prefer executable tests, schema checks, retrieved evidence, or external tools when they are available.

5. **Repair and checkpoint.** Correct detected failures, rerun affected tests, record evidence, and checkpoint the accepted state. If a new unit changes earlier interpretation, revisit dependent work and roll back unsafe or inconsistent changes.

A compact formulation is:

$$R_i = \text{Reconcile}(R_{i-1}, u_i),$$

$$\tilde{S}_i = \text{Execute}(S_{i-1}, a_i), \quad S_i = \text{Repair}(\tilde{S}_i, \text{Audit}(\tilde{S}_i, R_i)).$$

Here a_i may be a no-op when the new text is not yet actionable. *Reconcile* is not a simple append operation: later clauses can qualify, override, or contradict earlier interpretations. The active ledger must retain provenance so a reviewer can identify which part of the prompt motivated each obligation and which evidence supports its status.

3.1. Why not execute every character?

Literal character-by-character execution is a deliberately poor default. Meaning is compositional: “make it red” and “do not make it red” share most characters but express different constraints. Acting too early can encode a provisional interpretation into the artifact and cause costly rework. PECA therefore separates *incremental input consumption* from *execution granularity*. The controller may read characters or tokens, but should execute at semantic boundaries and defer irreversible actions until the instruction is sufficiently resolved.

3.2. Artifact state and side effects

The loop assumes persistent work state. For text, this may be a draft and requirement ledger; for code, a working tree and tests; for a browser agent, a proposed action queue and observed page state. Intermediate changes should be reversible where possible. External side effects—sending a message, purchasing an item, deleting data, deploying code—require a separate commit gate and any authorization demanded by the environment. A failed audit should lead to repair or explicit unresolved status, not a claim of success.

4. Hypotheses and evaluation plan

The central hypothesis is that, under a fixed resource budget, PECA increases satisfaction of explicit requirements and reduces severe constraint violations on tasks where instructions accumulate or interact. This has not yet been demonstrated.

The alternative is that additional audits merely increase cost, induce over-editing, or give little benefit over a strong final-pass reviewer.

4.1. Baselines and ablations

A controlled study should compare the same base model, tools, sampling settings, and task instances across:

- **B0: Direct execution.** Interpret the full prompt and produce the artifact, without a dedicated audit.
- **B1: Final-pass review.** Produce the artifact from the full prompt, then run one explicit requirement audit and repair pass.
- **B2: PECA.** Incremental semantic-prefix updates, persistent requirement ledger, execution, cumulative audit, and repair.
- **B3: PECA without cumulative audit.** Retain prefix processing and execution but remove the after-step audit to isolate the audit loop’s contribution.
- **B4: Naive fine-grained execution.** Trigger work at very small text units without semantic deferral, to test whether the boundary policy matters.

Budgets should be reported in two ways: equal-budget comparisons (tokens, tool calls, or wall-clock limits matched as closely as possible) and unconstrained quality comparisons. Otherwise, PECA could appear better simply because it spent more inference-time compute. A proposed first study is 200 prompts stratified across writing, scheduling, structured data, code modification, multi-step tool use, and constraint-heavy planning, with three independent runs per condition. This is a design recommendation, not a completed experiment.

4.2. Outcomes and controls

The primary outcome should be requirement satisfaction: the proportion of prompt-derived requirements passed by blind evaluators or deterministic checks. Also report (i) critical-violation rate, (ii) omission and contradiction rates, (iii) detected failures repaired successfully, (iv) regressions introduced by repairs, (v) unresolved ambiguity rate, (vi) tokens, tool calls, latency, and cost. For coding tasks, unit/integration tests and diffs can supply objective signals. For writing tasks, evaluators should use a predefined checklist with source-span traceability; model-as-judge scores alone are insufficient.

Randomize task order, preserve identical task inputs, use blinded evaluation, publish prompts and scoring rubrics, report all failures, and hold out task templates from development. Human evaluation should be sampled and reliability reported. Include adversarial cases with negation, late constraints, superseding instructions, conflicting requirements,

and instructions that appear easy to satisfy superficially. Compare the full system to ablations to attribute any gains to specific components.

5. Failure modes and safeguards

Correlated errors. An executor may also approve its own mistaken interpretation. Where possible, use independent verifiers, deterministic tests, typed schemas, or external evidence. Record uncertainty rather than silently passing a requirement.

Reward hacking at microtask scale. Local checks may reward cosmetic progress while overall task quality declines. The ledger and final audit must cover both individual requirements and interactions between them. Use held-out evaluators and test for gaming.

Premature commitment. Early action before a clause is complete can harden a wrong interpretation. Execute only at meaningful boundaries, maintain speculative changes separately, and make rollback cheap.

Constraint accumulation and contradictions. Not every instruction can be satisfied simultaneously. The system must identify conflict, preserve provenance, and ask for clarification or report the conflict rather than inventing a resolution. Superseded requirements should remain in the trace for auditability.

Cost and latency. An audit after every tiny unit can be expensive. A practical controller should batch units until a semantic boundary, use lightweight deterministic checks for simple requirements, and invoke stronger review where risk or ambiguity is high. The threshold policy itself needs evaluation.

Safety and authorization. Incremental execution increases the number of action opportunities, which can also increase risk. Irreversible actions should be staged behind permission checks and explicit commit gates. This paper does not propose weakening policy constraints or bypassing human oversight.

6. Limitations and scope

PECA is a testable systems proposal rather than a finished algorithm. We provide no new benchmark results, no measured performance improvement, and no evidence that it prevents reward hacking. The architecture may overlap with existing agent-controller implementations not identified in this brief literature review. Its claimed distinction is the explicit coupling of (a) instruction-prefix evolution, (b) a provenance-bearing active requirement ledger, (c) incremental artifact-changing actions, and (d) cumulative audits after each meaningful action. The brief literature review already identifies

adjacent 2026 work on verifier-gated prefixes and budgeted audit placement; a broader prior-art review and implementation comparison are necessary before claiming novelty.

The method may be most useful for work products with enumerable constraints and observable intermediate state. It may be less useful for open-ended creative tasks, problems requiring a global plan before any safe action can be taken, or tasks where verification is subjective and unreliable. Evaluations should report both benefits and regressions rather than assuming that more steps imply better alignment.

7. Conclusion

We propose PECA: interpret the prompt progressively, reconcile a traceable requirement ledger, execute only meaningful and preferably reversible microtasks, and audit the accumulated artifact against the full understood prefix after each step. The proposal shifts verification from an optional final phase into the agent’s execution controller. Its potential value lies in earlier omission detection and evidence-linked correction, not in the claim that microtasks inherently eliminate reward hacking. The decisive next step is a budget-controlled, blind, ablation-based evaluation against direct generation and strong final-pass review.

Drafting and reproducibility note

This manuscript is a position-paper draft prepared with language-model assistance. It contains a proposed architecture and evaluation protocol, not completed empirical research. Before submission, the author should verify every citation, conduct a broader prior-art review, revise the argument in their own voice, and disclose assistance according to the chosen venue’s policies. No benchmark scores are reported as experimental results.

References

- [1] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629. <https://arxiv.org/abs/2210.03629>
- [2] Aman Madaan et al. 2023. Self-Refine: Iterative Refinement with Self-Feedback. arXiv:2303.17651. <https://arxiv.org/abs/2303.17651>
- [3] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. In *Advances in Neural Information Processing Systems 36*. <https://arxiv.org/abs/2303.11366>
- [4] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. 2024. CRITIC: Large Language Models Can Self-Correct with Tool-Interactive Critiquing. In *International Conference on Learning Representations (ICLR)*. <https://arxiv.org/abs/2305.11738>
- [5] Amrith Setlur et al. 2025. Rewarding Progress: Scaling Automated Process Verifiers for LLM Reasoning. In *International Conference on Learning Representations (ICLR)*. <https://arxiv.org/abs/2410.08146>
- [6] Haizhou Shi et al. 2025. SSR: Socratic Self-Refine for Large Language Model Reasoning. arXiv:2511.10621. <https://arxiv.org/abs/2511.10621>
- [7] Chenyu Zhou, Qiliang Jiang, Shuning Wu, and Xu Zhou. 2026. Certified Speculative Execution for Untrusted AI Agents. arXiv:2606.31023. <https://arxiv.org/abs/2606.31023>
- [8] Kaixiang Wang, Yidan Lin, Jiong Lou, and Jie Li. 2026. BRA-Audit: Budgeted Runtime Auditing for LLM Multi-Agent Systems via Cumulative-Exposure Audit-Point Placement. arXiv:2608.14668. <https://arxiv.org/abs/2608.14668>
- [9] Chengguang Gan, Zimeng He, Yoshihiro Tsujii, Ken-ichiro Kobayashi, Hiroki Itoh, and Kotaro Funakoshi. 2026. Auditing Web Agent Evaluation on WebArena-Lite: Human Review of Outcomes and Trajectories. arXiv:2610.01491. <https://arxiv.org/abs/2610.01491>