COUNTERFACTUAL EVOLUTION CREDIT: CAUSAL SELECTION FOR SELF-EVOLVING AGENT SKILLS

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-evolving language-model agents convert execution experience into persistent updates to reusable skills. A central challenge is determining which candidate skills to retain. Existing selection strategies typically assign credit based on episode-level outcomes or invocation audits, conflating skill availability, invocation, behavioral influence, and causal contribution to task success. As a result, a candidate may be retained because it co-occurs with easy instances or discarded because unrelated execution noise masks its utility. We introduce Counterfactual Evolution Credit (CEC), a causal selection framework that estimates the incremental contribution of structured skill artifacts through matched execution interventions. CEC compares executions using a candidate skill against counterfactual runs in which the skill is removed, replaced by its parent, blocked from invocation, or modified at a specific functional module. These comparisons control for the task instance, environment state, execution budget, and, when possible, randomness. Repeated paired evaluations with fresh agents quantify residual uncertainty, while sequential allocation assigns additional evaluations to promising candidates whose effects remain unresolved. CEC also supports hierarchical attribution at the levels of whole skills, triggers, procedures, and recovery logic, allowing beneficial revisions to be retained and harmful components to be localized. We evaluate CEC under controlled failure modes involving unused, behaviorally inert, and spuriously correlated skills. The evaluation includes matched-budget comparisons, cross-agent reproduction, and ablations of replay control, confidence estimation, attribution granularity, and budget allocation. CEC supports skill selection under stochastic execution trajectories and limited evaluation budgets by separating causal contribution from invocation and episode-level correlation.

1 INTRODUCTION

Large language model (LLM) agents increasingly rely on *skills*, reusable artifacts that package procedural instructions, executable code, reference material, and task-specific constraints for inference-time use. Unlike model parameters, skills can be authored, inspected, revised, and transferred across compatible agents. Controlled evaluations show that curated skills improve performance on diverse expertise-intensive tasks, with gains varying across tasks and model–harness configurations.(Li et al., 2026b) Their portability and revisability support *self-evolving skill systems*, which convert execution feedback into persistent artifact updates beyond the current trajectory. Recent systems explore, author, deploy, and refine skills from agent experience, enabling procedural adaptation without updating the underlying model.(Zhang et al., 2026a)(Liu et al., 2026)

A central question is which candidate skills should be retained. Existing pipelines commonly select candidates based on episode reward, validation performance, trajectory comparison, or fresh-agent audits. These signals measure different properties. Outcome scores assess task completion, while fresh-agent audits reveal deployment failures such as available skills being silently bypassed.(Zhang et al., 2026a) Paired skill and no-skill evaluations estimate whether skill access changes aggregate performance under matched model–harness conditions.(Li et al., 2026b) Multi-round studies further show that skill evolution is sparse and sensitive to feedback composition, validation rules, and execution variability. Candidate quality therefore cannot be inferred directly from observed performance improvements.(Liu et al., 2026)

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

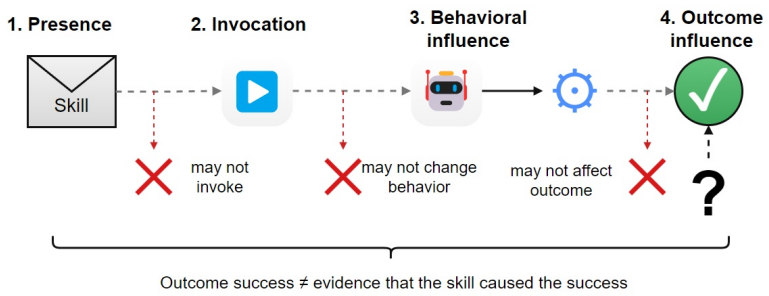

Figure 1: Counterfactual attribution and selection in CEC.

This limitation presents a causal credit-assignment problem. A successful episode does not establish that the selected skill caused the outcome. The agent may not invoke the skill, may invoke it without changing its behavior, or may solve an easy instance using base-model capabilities. A useful skill may also fail because of downstream execution noise, unrelated agent errors, or unfavorable environment realizations. Outcome-based selection conflates four events: skill presence, invocation, behavioral influence, and causal influence on the outcome. Invocation audits partially identify this discrepancy, but they cannot estimate outcomes under skill removal, parent replacement, or module-specific modification. Counterfactual baselines address similar entanglement in multi-agent reinforcement learning by contrasting global returns under localized interventions while holding other decisions fixed.(Foerster et al., 2017) Recent work applies this principle to LLM interactions through context-matched alternatives and controlled replay, converting terminal feedback into localized contribution estimates.(Chen et al., 2026) Self-evolving skill systems require attribution over persistent structured artifacts, whereas prior approaches attribute outcomes to agents or individual messages. Reliable selection must isolate whole-skill effects, identify contributing modules, and provide statistically useful estimates under stochastic trajectories and limited evaluation budgets.

We introduce **Counterfactual Evolution Credit (CEC)**, a causal selection framework for self-evolving agent skills. CEC performs matched interventions that compare candidate execution with executions in which the skill is removed, replaced by its parent, blocked from invocation, or modified at a single module. Each comparison preserves the task instance, environment state, execution budget, and random seed when possible. When residual stochasticity prevents exact replay, CEC repeats paired executions with fresh agents. Counterfactual differences estimate incremental contribution without assigning episode reward directly to the candidate. Removal and invocation blocking separate availability from use, parent replacement measures improvement over the immediate predecessor, and module replacement attributes performance changes to functional units such as triggers, procedures, and recovery logic. CEC aggregates paired effects with uncertainty estimates and sequentially allocates additional evaluations to candidates that appear promising but remain insufficiently resolved. It selects skills according to estimated causal utility while preserving existing artifact-level evolution workflows. Figure 1 illustrates this counterfactual attribution and selection framework.

We evaluate CEC under controlled failure modes using deceptive candidates that are never invoked, are invoked without changing the agent's action, or co-occur with success on comparatively easy instances. We compare episode reward, fresh-agent auditing, leave-one-out evaluation, and the complete counterfactual selector in terms of false-selection rate, frequency of genuinely beneficial selections, evaluation-budget efficiency, and reproduction across agents. Module-level interventions evaluate whether CEC localizes beneficial and harmful components, including recovery and trigger modules. Ablations vary seed matching, the number of paired executions, confidence estimation, attribution granularity, and sequential budget allocation. These experiments assess whether causal credit improves selection reliability and evolutionary stability under a fixed execution budget without treating higher observed reward as direct evidence of a better skill.

Our contributions are:

- We formulate self-evolving skill selection as a causal credit-assignment problem and distinguish skill presence, invocation, behavioral influence, and outcome influence, exposing failures hidden by episode-level rewards.

- We propose CEC, which estimates candidate utility through matched removal, parent replacement, invocation blocking, and module-level interventions, combined with repeated fresh-agent evaluation and uncertainty-aware selection.
- We develop a hierarchical attribution method that connects whole-skill effects to functional modules, allowing beneficial revisions to be retained while identifying harmful triggers, procedures, and recovery logic.
- We define a controlled evaluation protocol comprising deceptive skills, matched-budget baselines, cross-agent reproduction, and ablations of replay control, confidence estimation, attribution granularity, and sequential allocation.

2 Related Work

2.1 Persistent Experience and Self-Evolving Agent Skills

Language agents can convert interaction traces into reusable inference-time state without updating model parameters. Reflexion retains natural-language feedback from prior trials,(Shinn et al., 2023) ExpeL extracts cross-task insights from successful and failed trajectories,(Zhao et al., 2023) Self-Refine iteratively generates feedback and revisions,(Madaan et al., 2023) and CRITIC grounds revision in external tools.(Gou et al., 2023) These methods primarily represent experience as unstructured text interpreted in context.

Procedural-memory systems instead compile experience into reusable programs or routines. Voyager builds a library of code-based skills,(Wang et al., 2023) while SkillWeaver distills successful web interactions into API-level skills.(Zheng et al., 2025) MemSkill evolves routines for extracting, consolidating, and pruning memory,(Zhang et al., 2026b) and reinforcement-learning approaches jointly optimize skill generation and utilization.(Wang et al., 2025)

Recent work treats skills as portable artifacts containing instructions, scripts, references, and constraints. SkillsBench evaluates paired skill and no-skill conditions,(Li et al., 2026b) SkillEvolver manages persistent skill authoring and refinement through execution outcomes and audits,(Zhang et al., 2026a) and multi-round analyses show that evolution can be sparse and sensitive to feedback, validation, and execution variability.(Liu et al., 2026) Open-ended systems similarly select modified agents using benchmark performance.(Zhang et al., 2025) Natural-language program optimization follows related principles: Automatic Prompt Engineer searches generated instructions,(Zhou et al., 2022) OPRO proposes candidates from textual optimization histories,(Yang et al., 2023) and TextGrad propagates textual critiques through compound pipelines.(Yuksekgonul et al., 2024)

2.2 Credit Assignment through Counterfactual Baselines

Cooperative reinforcement learning associates shared returns with localized decisions. Difference rewards compare realized utility with a counterfactual replacing one component by default behavior,(Wolpert & Tumer, 2001) while COMA marginalizes one agent's action under a centralized critic.(Foerster et al., 2017) Value decomposition instead represents joint value through additive individual utilities in VDN,(Sunehag et al., 2017) monotonic mixing in QMIX,(Rashid et al., 2018) consistency constraints in QTRAN,(Son et al., 2019) or duplex dueling factorization in QPLEX.(Wang et al., 2020) Shapley methods allocate joint value through coalition-level marginal contributions.(Wang et al., 2019)

Temporal methods identify earlier decisions responsible for delayed returns. RUDDER redistributes returns toward influential transitions,(Arjona-Medina et al., 2018) Hindsight Credit Assignment conditions attribution on observed outcomes,(Harutyunyan et al., 2019) and causal influence rewards measure how one agent changes another's behavior.(Jaques et al., 2018) Counterfactual Shapley credit separates policy contributions from environmental stochasticity,(Li et al., 2026a) while C3 uses context-matched alternatives and controlled replay to localize terminal feedback in LLM interactions.(Chen et al., 2026)

Model-attribution methods provide related decompositions. LIME fits sparse local surrogates to perturbed inputs,(Ribeiro et al., 2016) SHAP computes feature contributions relative to a background distribution,(Lundberg & Lee, 2017) and Integrated Gradients attributes predictions along a baseline

path.(Sundararajan et al., 2017) TCAV measures sensitivity to concept directions,(Kim et al., 2017) whereas causal mediation separates direct and representation-mediated effects.(Vig et al., 2020) These methods emphasize baseline choice and intervention semantics, although static perturbations may not preserve interactive-agent trajectories.

2.3 Controlled and Uncertainty-Aware Agent Evaluation

HELM standardizes scenarios, metrics, and reporting,(Liang et al., 2022) while MT-Bench and Chatbot Arena compare conversational systems through multi-turn tasks and pairwise judgments.(Zheng et al., 2023) AlpacaEval controls evaluator preferences and output-length effects,(Dubois et al., 2024) and reproducibility studies motivate repeated runs to characterize uncertainty.(Blackwell et al., 2024) Paired designs improve sensitivity, but resolution depends on correlation, sample size, clustering, multiplicity, and effect magnitude.(Kotawala, 2026)

Interactive benchmarks add variation from environment state, tool responses, action validity, and error propagation. AgentBench spans heterogeneous environments,(Liu et al., 2023) and AgentBoard supplements success with progress metrics.(Ma et al., 2024) WebArena provides reproducible websites and functional checks,(Zhou et al., 2023) while AppWorld uses stateful applications, executable APIs, and programmatic tests.(Trivedi et al., 2024) Nevertheless, terminal scores jointly reflect models, prompts, memory, tools, skills, and environments, and unmatched ablations may confound artifact effects with divergent trajectories or environment realizations.

3 Method

We introduce Counterfactual Evolution Credit (CEC), a causal selection framework for self-evolving agent skills. CEC compares matched executions under explicit interventions to distinguish skill availability, invocation, behavioral influence, and outcome improvement.

3.1 Problem Formulation and Framework

Let $\mathcal{X}$ denote a task distribution. Each task $x \sim \mathcal{X}$ specifies an instruction, an initial environment state e_0, an execution budget b, and a verifier $v_x : \mathcal{Y} \to [0, 1]$. Given a policy π, a skill $s \in \mathcal{S}$, and randomness ω, the agent produces the trajectory $\xi(s; x, \omega) = (o_0, a_0, c_0, \ldots, o_T, a_T, c_T, y_T)$, where o_t and a_t denote observations and actions, c_t records retrieval and invocation events, and y_T is the terminal output. The corresponding reward is $R(s; x, \omega) = v_x(y_T(\xi(s; x, \omega)))$.

Outcome-only selection cannot distinguish success caused by the skill from success produced through silent bypass, behaviorally irrelevant invocation, or the base policy. CEC treats the deployed configuration as an intervention. Given an intervention set $\mathcal{I}(s)$ and a reference intervention i_0, the task-conditional credit is

$$\Delta_i(x) = \mathbb{E}_\omega[R(i(s); x, \omega) - R(i_0(s); x, \omega)]. \tag{1}$$

This quantity represents an interventional effect under the matched-execution assumptions in Section 3.2. Passive trajectories alone do not identify this effect.

Given a parent skill s^{par} and descendants $\mathcal{C} = \{s^{(1)}, \ldots, s^{(K)}\}$, CEC constructs skill- and module-level interventions, executes matched pairs, estimates hierarchical credit and uncertainty, and allocates additional evaluations to candidates with high plausible credit or unresolved estimates. Figure 2 summarizes this process from counterfactual intervention construction to sequential selection and audit recording.

3.2 Matched Counterfactual Construction

For a candidate s, the principal intervention conditions are

$$i_{\mathrm{cand}}(s) = s, \qquad i_{\mathrm{none}}(s) = \varnothing, \qquad i_{\mathrm{par}}(s) = s^{\mathrm{par}}, \tag{2}$$

where $\varnothing$ disables candidate-specific content while preserving the agent, tools, and task interface. The candidate-versus-none contrast measures the candidate's absolute contribution, while the candidate-versus-parent contrast measures the incremental contribution used for evolutionary acceptance.

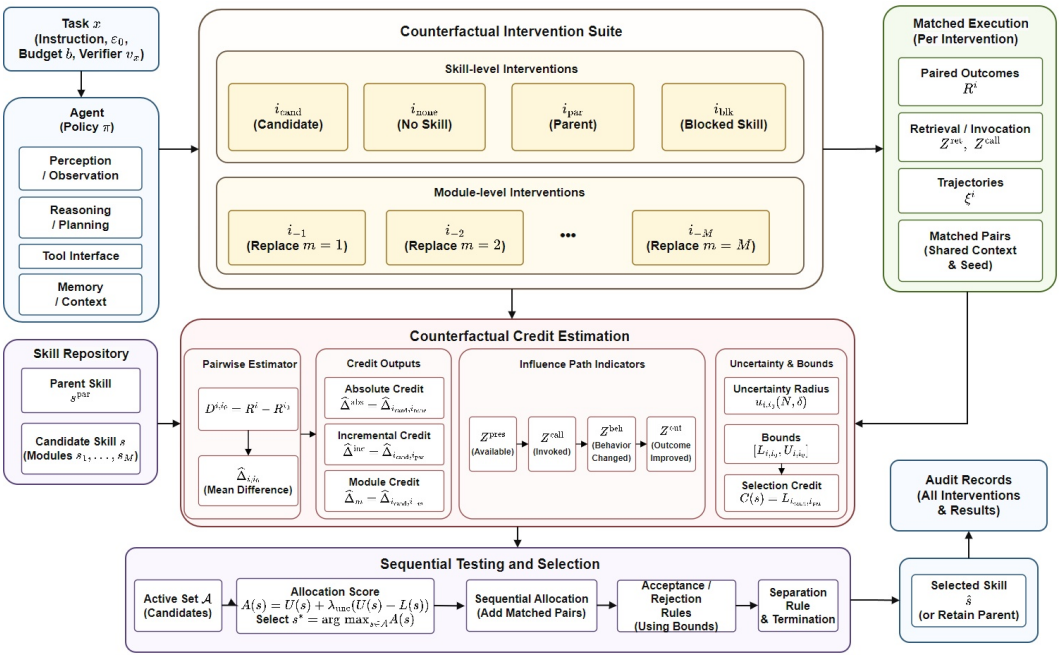

Figure 2: Overview of the CEC evaluation and selection framework.

To separate availability from use, $i_{\text{blk}}(s)$ exposes the same metadata and retrieval interface but blocks the skill body and resources upon invocation. Let $Z^{\text{ret}}, Z^{\text{call}} \in \{0, 1\}$ denote retrieval and invocation. The contrast between i_{cand} and i_{blk} tests whether access to the operational skill content changes execution after invocation.

For a candidate with modules $s = (s_1, \ldots, s_M)$, the intervention $i_{-m}(s) = (s_1, \ldots, s_{m-1}, \tilde{s}_m, s_{m+1}, \ldots, s_M)$ replaces only module m. If the parent contains a corresponding module, $\tilde{s}_m$ is inherited from the parent. Otherwise, a null module preserves the interface without operational content.

For task x_n, intervention i, and replicate j, a fresh agent records $\mathcal{O}^i_{nj} = (R^i_{nj}, Z^{\text{ret},i}_{nj}, Z^{\text{call},i}_{nj}, \xi^i_{nj})$. Each pair shares the task, initial environment snapshot, tool state, budget, and random seed when the seed is controllable. The context is reset between runs. When exact replay is unavailable, both conditions preserve the observable pre-execution state and are evaluated repeatedly with fresh agents. Appendix provides the implementation details.

3.3 Counterfactual Credit Estimation

For an intervention i and reference i_0, each matched pair yields $D^{i,i_0}_{nj} = R^i_{nj} - R^{i_0}_{nj}$. Over N matched pairs, we estimate the credit as

$$\widehat{\Delta}_{i,i_0} = \frac{1}{N} \sum_{(n,j) \in \mathcal{P}_{i,i_0}} D^{i,i_0}_{nj}, \tag{3}$$

where $\mathcal{P}_{i,i_0}$ contains observations available under both conditions. The skill-level credits are

$$\widehat{\Delta}^{\text{abs}}(s) = \widehat{\Delta}_{i_{\text{cand}},i_{\text{none}}}, \qquad \widehat{\Delta}^{\text{inc}}(s) = \widehat{\Delta}_{i_{\text{cand}},i_{\text{par}}}. \tag{4}$$

Absolute credit measures the contribution of the skill relative to its absence, and incremental credit measures its contribution relative to the parent. The credit assigned to module m is $\widehat{\Delta}_m(s) = \widehat{\Delta}_{i_{\text{cand}},i_{-m}}$. Positive values support retaining module m, while negative values support its replacement or reversion. Module credits are non-additive because interactions can produce $\widehat{\Delta}^{\text{inc}}(s) \neq \sum_m \widehat{\Delta}_m(s)$.

Table 1: Selection performance under a matched budget of 240 executions per task suite. Success and true improvement are higher-is-better; false selection is lower-is-better. Values are mean±standard deviation over five seeds. Abbreviations: SB = SkillsBench; SSB = SpreadsheetBench; SQA = SearchQA; TI = True improvement; FS = False selection; CF = Counterfactual.

Method	SB-CEC	SSB-CEC	SQA-CEC	TI ↑	FS ↓
No-Skill	38.6 ± 1.4	34.1 ± 1.6	51.8 ± 1.2	–	–
Episode-Reward	47.9 ± 1.8	43.5 ± 2.0	57.2 ± 1.5	58.3 ± 2.4	25.7 ± 2.0
Best-of-N	48.7 ± 1.6	44.2 ± 1.9	57.9 ± 1.3	60.1 ± 2.1	23.4 ± 1.8
Fresh-Audit	51.2 ± 1.5	46.8 ± 1.7	59.1 ± 1.4	65.9 ± 1.9	17.6 ± 1.5
Leave-One-Out	52.3 ± 1.4	48.7 ± 1.8	$\underline{60.4 \pm 1.1}$	69.8 ± 1.8	13.8 ± 1.4
CF-Uniform	$\underline{53.8 \pm 1.3}$	$\underline{50.1 \pm 1.5}$	60.2 ± 1.2	$\underline{72.1 \pm 1.6}$	$\underline{11.2 \pm 1.2}$
CEC	$\mathbf{56.1 \pm 1.2}$	$\mathbf{53.4 \pm 1.4}$	$\mathbf{62.0 \pm 1.0}$	$\mathbf{74.8 \pm 1.7}$	$\mathbf{8.9 \pm 1.1}$

CEC also identifies the path through which a skill affects execution. Let Z^{pres} denote availability, and define

$$Z_{nj}^{\mathrm{beh}} = \mathbb{I}\big[d_\xi(\xi_{nj}^{i_{\mathrm{cand}}}, \xi_{nj}^{i_{\mathrm{blk}}}) > \epsilon_\xi\big], \qquad Z_{nj}^{\mathrm{out}} = \mathbb{I}\Big[D_{nj}^{i_{\mathrm{cand}}, i_{\mathrm{blk}}} > 0\Big]. \tag{5}$$

The chain $Z^{\mathrm{pres}} \to Z^{\mathrm{call}} \to Z^{\mathrm{beh}} \to Z^{\mathrm{out}}$ distinguishes bypassed skills from invoked skills that leave behavior unchanged. These indicators diagnose the influence path, while acceptance is determined by paired rewards.

Given the sample variance $\widehat{\sigma}_{i,i_0}^2$ and bounded differences $D_{nj}^{i,i_0} \in [-1, 1]$, the uncertainty radius is

$$u_{i,i_0}(N, \delta) = \sqrt{\frac{2\widehat{\sigma}_{i,i_0}^2 \log(3/\delta)}{N}} + \frac{3\log(3/\delta)}{N}. \tag{6}$$

The corresponding confidence bounds are

$$L_{i,i_0} = \widehat{\Delta}_{i,i_0} - u_{i,i_0}, \qquad U_{i,i_0} = \widehat{\Delta}_{i,i_0} + u_{i,i_0}. \tag{7}$$

We adjust the confidence levels across active comparisons. The selection credit is the lower confidence bound on incremental credit, $C(s) = L_{i_{\mathrm{cand}}, i_{\mathrm{par}}}(s)$. This score penalizes gains estimated from few or unstable executions.

3.4 Sequential Testing and Selection

CEC maintains a set of active candidates $\mathcal{A} \subseteq \mathcal{C}$ under evaluation budget B. After matched initialization, it updates the incremental-credit estimates and confidence bounds. It assigns additional pairs according to $A(s) = U(s) + \lambda_{\mathrm{unc}}(U(s) - L(s))$, which prioritizes candidates with high upper bounds or wide confidence intervals. The next pair is assigned to $s^\star = \arg\max_{s \in \mathcal{A}} A(s)$. When module-level attribution is required, unresolved module comparisons follow the same allocation rule.

A candidate is rejected when $U(s) < \tau_{\mathrm{rej}}$ and becomes eligible for acceptance when

$$L(s) > \tau_{\mathrm{acc}} \quad \text{and} \quad L_{i_{\mathrm{cand}}, i_{\mathrm{none}}}(s) > 0, \tag{8}$$

where $\tau_{\mathrm{acc}} \geq \tau_{\mathrm{rej}}$. The absolute-credit condition excludes candidates that improve their parent but remain worse than the no-skill condition. Testing terminates when the budget is exhausted, all candidates are resolved, or an eligible candidate $\hat{s}$ satisfies

$$L(\hat{s}) > \max_{s \in \mathcal{A} \setminus \{\hat{s}\}} U(s). \tag{9}$$

If no candidate satisfies Equation (8), CEC retains the parent. Otherwise, it selects the candidate with the largest incremental lower bound. Candidate generation remains external to CEC, which modifies only the evaluation and selection stages. Appendix presents the complete procedure and audit-record construction.

Table 2: False-selection rates for injected deceptive skills; lower is better. Values are mean±standard deviation over five seeds. Abbreviations: Avg. = Average; CF = Counterfactual.

Method	Uncalled	Inert call	Spurious success	Avg.
Episode-Reward	31.8 ± 2.5	34.7 ± 2.9	38.5 ± 2.8	35.0
Fresh-Audit	9.7 ± 1.5	22.6 ± 2.1	25.1 ± 2.4	19.1
Leave-One-Out	8.4 ± 1.3	14.9 ± 1.8	17.2 ± 1.9	13.5
CF-Uniform	$\underline{5.8 \pm 1.0}$	$\underline{9.7 \pm 1.4}$	$\underline{12.1 \pm 1.6}$	$\underline{9.2}$
CEC	$\mathbf{3.9 \pm 0.8}$	$\mathbf{6.4 \pm 1.1}$	$\mathbf{9.1 \pm 1.3}$	**6.5**

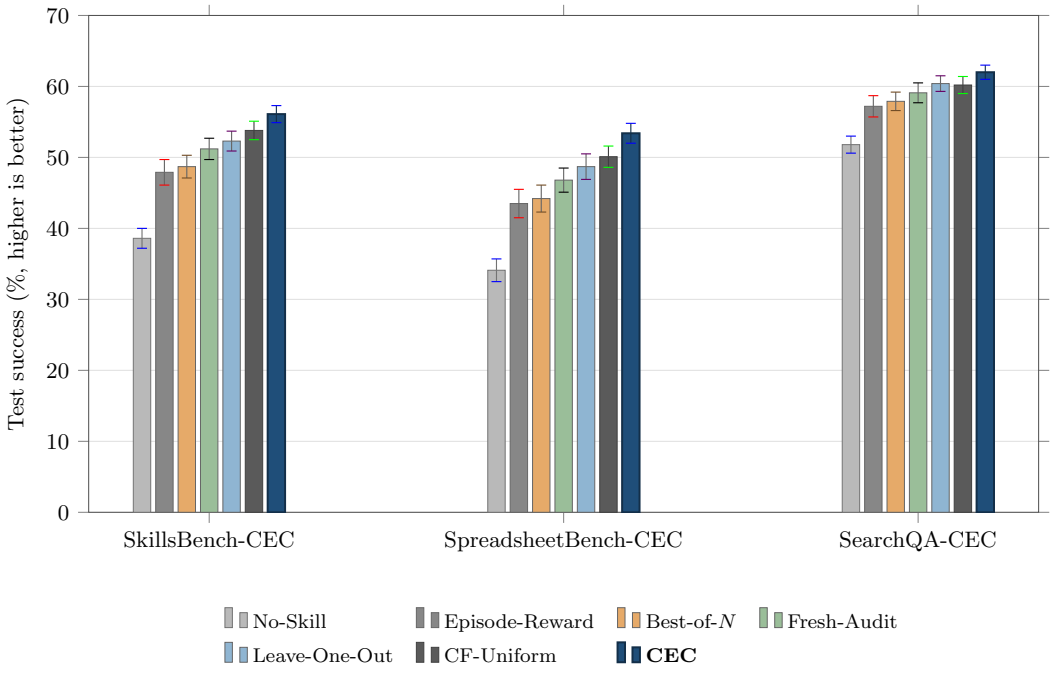

Figure 3: Test success under a matched budget of 240 executions per task suite. Bars show means over five seeds, and error bars show the reported standard deviations.

4 Experiments

4.1 Evaluation Protocol

We evaluate whether Counterfactual Evolution Credit (CEC) identifies skills that causally improve agent performance under a fixed evaluation budget. The evaluation covers three task suites with distinct execution characteristics. SkillsBench-CEC contains 60 tasks spanning document editing, data analysis, planning, coding, and specialized workflows. SpreadsheetBench-CEC comprises 120 spreadsheet tasks with instrumented skill calls and action traces, enabling precise detection of behavioral changes. SearchQA-CEC includes 200 retrieval-intensive questions and evaluates whether selected skills generalize beyond the evolution distribution. We divide each suite into evolution, validation, and held-out test sets using a 40/20/40 split. Evolution trajectories generate candidate skills, validation data govern selection and stopping decisions, and test data are accessed after the selection procedure is fixed. The split is performed before any candidate-level analysis to prevent validation feedback from influencing held-out evaluation. Suite-specific partitions preserve the original task diversity, so observed differences reflect the selection procedure rather than a narrowed test distribution.

We compare CEC with NO-SKILL, EPISODE-REWARD, BEST-OF-N, FRESH-AUDIT, LEAVE-ONE-OUT, and COUNTERFACTUAL-UNIFORM. FRESH-AUDIT evaluates each candidate with an independent agent and rejects candidates that are silently bypassed. LEAVE-ONE-OUT estimates a candidate's contribution by removing the complete skill. COUNTERFACTUAL-UNIFORM uses the same interventions as CEC and allocates executions equally across candidates. CEC compares candidate execution with matched removal, parent replacement, invocation blocking, and module replacement to estimate

$$\widehat{\tau}(s) = \mathbb{E}[R \mid do(s)] - \mathbb{E}[R \mid do(b)],$$

where b denotes the relevant no-skill, parent-skill, blocked-invocation, or module-replacement baseline. Selection applies a confidence-aware estimate of this effect, and sequential allocation assigns additional paired executions to candidates with positive but uncertain effects.

All methods use the same optimizer, executor, candidate pool, and total execution budget. For matched comparisons, we restore the task input and environment snapshot while fixing the random seed, tool budget, and 30-turn limit. Paired runs are scheduled adjacently whenever possible, reducing temporal variation in external services and executor load. We retain failed executions in the estimates because excluding them would systematically favor candidates that induce unstable behavior. Results are averaged over five seeds. We report test success, the proportion of selected skills with positive paired test effects, the proportion with non-positive effects, and the number of executions per validated improvement. Module localization is correct when the injected harmful component receives the lowest credit. The appendix provides additional implementation details.

4.2 SELECTION QUALITY

Table 1 reports the main comparison under a budget of 240 executions per suite, with test success shown in Figure 3. CEC achieves the highest success rate on each benchmark: $56.1 \pm 1.2\%$ on SKILLSBENCH-CEC, $53.4 \pm 1.4\%$ on SPREADSHEETBENCH-CEC, and $62.0 \pm 1.0\%$ on SEARCHQA-CEC. Averaged across suites, CEC exceeds EPISODE-REWARD by 6.2 percentage points and COUNTERFACTUAL-UNIFORM by 2.6 points. Its selected skills achieve a true improvement rate of $74.8 \pm 1.7\%$, compared with $58.3 \pm 2.4\%$ for reward-based selection, and reduce false selections from $25.7 \pm 2.0\%$ to $8.9 \pm 1.1\%$. The consistency across suites indicates that the aggregate improvement is not driven by a single execution setting. Moreover, the reduction in false selections accompanies higher test success, suggesting that conservative attribution does not merely discard useful skills.

Structured counterfactuals provide the largest gain on SPREADSHEETBENCH-CEC, where a skill can be invoked without changing the final workbook and separate modules can affect different portions of a long trajectory. CEC improves over uniform counterfactual allocation by 3.3 points on this suite and by 2.3 points on SKILLSBENCH-CEC. The margin is 1.8 points on SEARCHQA-CEC, where shorter trajectories and directly verifiable answers allow whole-skill removal to approximate the relevant intervention more closely. These results show larger gains when invocation is an unreliable proxy for behavioral influence or when skill modules have separable effects.

4.3 DECEPTIVE SKILLS AND CAUSAL DIAGNOSIS

We evaluate the three attribution failures that motivate CEC. For each validation task, we inject an *uncalled* skill whose trigger never activates, an *inert-call* skill that is invoked without changing the action sequence, and a *spurious-success* skill associated disproportionately with easier instances. Each injected skill has zero intended causal contribution and should receive no positive causal credit.

Table 2 shows that episode reward selects 35.0% of deceptive candidates on average, including $38.5 \pm 2.8\%$ of candidates correlated with easy instances. Fresh-agent auditing detects most uncalled skills but retains $22.6 \pm 2.1\%$ of inert calls because invocation alone satisfies its acceptance criterion. By comparing matched executions with and without invocation blocking, CEC reduces the corresponding false-selection rates to $3.9 \pm 0.8\%$ and $6.4 \pm 1.1\%$. Its average false-selection rate is 6.5%, which is 12.6 points lower than FRESH-AUDIT and 7.0 points lower than LEAVE-ONE-OUT.

The remaining $9.1 \pm 1.3\%$ error on spurious-success candidates occurs primarily in retrieval tasks with hidden service state or nondeterministic external responses. Shared inputs and seeds cannot guarantee identical potential outcomes when this state cannot be restored. The causal interpretation

of CEC is therefore limited to the observable, replayable intervention protocol and excludes effects from uncontrolled environment variables.

4.4 Attribution, Efficiency, and Transfer

The component ablations isolate the contributions of matched replay, confidence-aware selection, and sequential allocation. Removing shared seeds lowers average success from 57.2% to 53.8% and increases false selection from 8.9% to 15.8%. A single paired replay yields a 70.4% true improvement rate, 4.4 points below full CEC. Selection based on the empirical mean without confidence bounds increases false selection to 12.7%. Uniform allocation yields 56.0% success and an 11.2% false-selection rate. Together, these ablations distinguish variance reduction from decision policy effects: matched replay stabilizes comparisons, confidence bounds regulate acceptance, and sequential allocation concentrates evidence where selection remains ambiguous. The appendix reports the complete ablation table.

Module interventions separate whole-skill value from the contributions of individual components. Full CEC identifies an injected harmful trigger or recovery module in $84.7 \pm 1.6\%$ of cases, compared with $68.9 \pm 2.5\%$ without shared seeds. Skill-level attribution supports candidate selection but cannot identify the component that requires repair. For example, a harmful recovery routine can coexist with useful planning and verification logic, so deleting the complete skill can reduce reward even when replacing the harmful module improves it.

Sequential allocation improves selection accuracy and evaluation efficiency. At 240 executions, CEC requires 31.6 executions per validated improvement, compared with 38.9 for uniform counterfactual evaluation, and increases test success from 54.7% to 57.2%. Increasing the budget to 360 executions yields a further 0.8-point gain, indicating diminishing returns after resolving candidates with clearly positive or negative effects. Under a budget of 120 executions, CEC retains a 1.7-point advantage over uniform allocation.

Skills selected with Agent-A also transfer to two heterogeneous executors. Under a budget of 240 executions, these skills achieve $54.9 \pm 1.4\%$ success with Agent-B and $53.2 \pm 1.6\%$ with Agent-C, exceeding uniformly selected skills by 3.1 and 2.7 points, respectively. Both results remain below the 57.2% source-agent result, with larger degradation for skills containing harness-specific tool instructions. CEC reduces correlations specific to the source agent within this evaluation setting, while agent-independent causal effects remain outside the scope of these experiments. The results support using CEC when false acceptance is costly, execution is stochastic, or module-level repair is required. Its advantage is smaller when trajectories are deterministic and whole-skill removal already isolates the relevant effect.

5 Conclusion

We presented Counterfactual Evolution Credit (CEC), a causal framework that selects self-evolving agent skills through matched interventions and attributes utility at both whole-skill and module levels. By separating skill availability, invocation, behavioral influence, and outcome influence, CEC assigns evolutionary credit to genuine causal contributions rather than observed episode rewards. Controlled evaluations across deceptive candidates, matched-budget baselines, cross-agent reproduction, and targeted ablations established that CEC improves selection reliability, identifies beneficial and harmful modules, uses evaluation budgets efficiently, and strengthens evolutionary stability. These findings confirm causal attribution as an effective foundation for persistent skill evolution.

Limitations

CEC requires multiple matched executions for each candidate and module, which may introduce additional evaluation cost when skill repositories are large. Its estimates also depend on the fidelity of environment replay; hidden service states or nondeterministic tool responses can weaken matching between counterfactual runs. Finally, the experiments cover three task suites and a limited set of agent executors, so the observed benefits may vary for substantially different environments, skill formats, longer interaction horizons, or model–harness configurations.

LLM USAGE STATEMENT

The authors used large language model tools in this work for the sole purpose of improving the written presentation of the manuscript. This included tasks such as refining sentence phrasing, improving paragraph coherence, and correcting grammatical errors to enhance readability. All core research activities—including the counterfactual credit assignment formulation, theoretical derivations, algorithm design, and empirical evaluations—were carried out exclusively by the human authors. LLMs were not consulted for generating research content, or analyzing results. Every AI-assisted language edit was reviewed and approved by the authors. The authors accept full responsibility for the accuracy and integrity of this work.

REFERENCES

Jose A. Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. Rudder: Return decomposition for delayed rewards. *arXiv preprint arXiv:1806.07857*, 2018. URL https://arxiv.org/abs/1806.07857.

Robert E. Blackwell, Jon Barry, and Anthony G. Cohn. Towards reproducible llm evaluation: Quantifying uncertainty in llm benchmark scores. *arXiv preprint arXiv:2410.03492*, 2024. URL https://arxiv.org/abs/2410.03492.

Yanjun Chen, Yirong Sun, Hanlin Wang, Jinghan Wang, Xinming Zhang, Xiaoyu Shen, Wenjie Li, and Wei Zhang. Exact is easier: Credit assignment for cooperative llm agents. *arXiv preprint arXiv:2603.06859*, 2026. URL https://arxiv.org/abs/2603.06859.

Yann Dubois, Balazs Galambosi, Percy Liang, and Tatsunori B. Hashimoto. Length-controlled alpacaeval: A simple way to debias automatic evaluators. *arXiv preprint arXiv:2404.04475*, 2024. URL https://arxiv.org/abs/2404.04475.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. *arXiv preprint arXiv:1705.08926*, 2017. URL https://arxiv.org/abs/1705.08926.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*, 2023. URL https://arxiv.org/abs/2305.11738.

Anna Harutyunyan, Will Dabney, Thomas Mesnard, Mohammad Azar, Bilal Piot, Nicolas Heess, Hado van Hasselt, Greg Wayne, Satinder Singh, Doina Precup, and Remi Munos. Hindsight credit assignment. *arXiv preprint arXiv:1912.02503*, 2019. URL https://arxiv.org/abs/1912.02503.

Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Caglar Gulcehre, Pedro A. Ortega, DJ Strouse, Joel Z. Leibo, and Nando de Freitas. Social influence as intrinsic motivation for multi-agent deep reinforcement learning. *arXiv preprint arXiv:1810.08647*, 2018. URL https://arxiv.org/abs/1810.08647.

Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, and Rory Sayres. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (TCAV). *arXiv preprint arXiv:1711.11279*, 2017. URL https://arxiv.org/abs/1711.11279.

Anany Kotawala. Resolution diagnostics for paired llm evaluation. *arXiv preprint arXiv:2605.30315*, 2026. URL https://arxiv.org/abs/2605.30315.

Mingxuan Li, Kai-Zhan Lee, and Elias Bareinboim. Counterfactual shapley credit assignment. *arXiv preprint arXiv:2607.16999*, 2026a. URL https://arxiv.org/abs/2607.16999.

Xiangyi Li et al. Skillsbench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*, 2026b. URL https://arxiv.org/abs/2602.12670.

Percy Liang et al. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*, 2022. URL https://arxiv.org/abs/2211.09110.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023. URL https://arxiv.org/abs/2308.03688.

Yuxuan Liu, Zhaochen Su, Yuhao Zhang, Jiahe Guo, Zhongwei Xie, Huihao Jing, Lingyun Xie, Qing Zong, Yauwai Yim, Zhixiong Zhang, Haoran Li, and Yangqiu Song. Rethinking self-evolving agent skills: Feedback dynamics over multiple rounds. *arXiv preprint arXiv:2608.02636*, 2026. URL https://arxiv.org/abs/2608.02636.

Scott Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *arXiv preprint arXiv:1705.07874*, 2017. URL https://arxiv.org/abs/1705.07874.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn llm agents. *arXiv preprint arXiv:2401.13178*, 2024. URL https://arxiv.org/abs/2401.13178.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023. URL https://arxiv.org/abs/2303.17651.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1803.11485*, 2018. URL https://arxiv.org/abs/1803.11485.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *arXiv preprint arXiv:1602.04938*, 2016. URL https://arxiv.org/abs/1602.04938.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023. URL https://arxiv.org/abs/2303.11366.

Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:1905.05408*, 2019. URL https://arxiv.org/abs/1905.05408.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. *arXiv preprint arXiv:1703.01365*, 2017. URL https://arxiv.org/abs/1703.01365.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017. URL https://arxiv.org/abs/1706.05296.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. *arXiv preprint arXiv:2407.18901*, 2024. URL https://arxiv.org/abs/2407.18901.

Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Simas Sakenis, Jason Huang, Yaron Singer, and Stuart Shieber. Causal mediation analysis for interpreting neural nlp: The case of gender bias. *arXiv preprint arXiv:2004.12265*, 2020. URL https://arxiv.org/abs/2004.12265.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023. URL https://arxiv.org/abs/2305.16291.

Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. Qplex: Duplex dueling multi-agent q-learning. *arXiv preprint arXiv:2008.01062*, 2020. URL https://arxiv.org/abs/2008.01062.

Jianhong Wang, Yuan Zhang, Tae-Kyun Kim, and Yunjie Gu. Shapley q-value: A local reward approach to solve global reward games. *arXiv preprint arXiv:1907.05707*, 2019. URL https://arxiv.org/abs/1907.05707.

Jiongxiao Wang, Qiaojing Yan, Yawei Wang, Yijun Tian, Soumya Smruti Mishra, Zhichao Xu, Megha Gandhi, Panpan Xu, and Lin Lee Cheong. Reinforcement learning for self-improving agent with skill library. *arXiv preprint arXiv:2512.17102*, 2025. URL https://arxiv.org/abs/2512.17102.

David H. Wolpert and Kagan Tumer. Optimal payoff functions for members of collectives. *Advances in Complex Systems*, 2001. URL https://ideas.repec.org/a/wsi/acsxxx/v04y2001i02n03ns0219525901000188.html.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*, 2023. URL https://arxiv.org/abs/2309.03409.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024. URL https://arxiv.org/abs/2406.07496.

Genrui Zhang, Erle Zhu, Jinfeng Zhou, Caiyan Jia, and Hongning Wang. Skillevolver: Skill learning as a meta-skill. *arXiv preprint arXiv:2605.10500*, 2026a. URL https://arxiv.org/abs/2605.10500.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. Memskill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*, 2026b. URL https://arxiv.org/abs/2602.02474.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin godel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*, 2025. URL https://arxiv.org/abs/2505.22954.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. *arXiv preprint arXiv:2308.10144*, 2023. URL https://arxiv.org/abs/2308.10144.

Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. Skillweaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025. URL https://arxiv.org/abs/2504.07079.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*, 2023. URL https://arxiv.org/abs/2306.05685.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023. URL https://arxiv.org/abs/2307.13854.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*, 2022. URL https://arxiv.org/abs/2211.01910.

APPENDIX

A COMPLETE SEQUENTIAL COUNTERFACTUAL SELECTION PROCEDURE

This section specifies the bookkeeping, update order, and termination criteria for sequential candidate selection. The procedure operates on matched intervention pairs. It uses incremental credit for ranking, absolute credit for admissibility, and confidence bounds for rejection and separation.

Let

$$\mathcal{Q} = \{\text{inc}, \text{abs}\}$$

index the two required skill-level contrasts. For each candidate $s \in \mathcal{C}$, define

$$(i_s^{\text{inc}}, i_{0,s}^{\text{inc}}) = (i_{\text{cand}}(s), i_{\text{par}}(s)), \qquad (i_s^{\text{abs}}, i_{0,s}^{\text{abs}}) = (i_{\text{cand}}(s), i_{\text{none}}(s)).$$

For each contrast $q \in \mathcal{Q}$, the paired-difference collection is

$$\mathcal{D}_s^q = \left\{ D_{nj}^{i_s^q, i_{0,s}^q} : (n,j) \in \mathcal{P}_{i_s^q, i_{0,s}^q} \right\}, \qquad N_s^q = |\mathcal{D}_s^q|.$$

The collection $\mathcal{D}_s^q$ contains only observations available under both intervention conditions. Incomplete or unmatched executions therefore contribute neither to the sample size nor to the credit estimate.

For each nonempty paired collection, the credit estimate and sample variance are

$$\widehat{\Delta}_s^q = \frac{1}{N_s^q} \sum_{D \in \mathcal{D}_s^q} D, \tag{10}$$

$$\widehat{\sigma}_{s,q}^2 = \frac{1}{N_s^q - 1} \sum_{D \in \mathcal{D}_s^q} \left(D - \widehat{\Delta}_s^q \right)^2, \tag{11}$$

where matched initialization provides enough observations to estimate the uncertainty used by the procedure. Let δ_s^q denote the confidence level assigned to this active comparison after the stated multiple-comparison adjustment. The corresponding bounds are

$$u_s^q = \sqrt{\frac{2\widehat{\sigma}_{s,q}^2 \log(3/\delta_s^q)}{N_s^q}} + \frac{3\log(3/\delta_s^q)}{N_s^q}, \tag{12}$$

$$L_s^q = \widehat{\Delta}_s^q - u_s^q, \qquad U_s^q = \widehat{\Delta}_s^q + u_s^q. \tag{13}$$

We write the incremental bounds as $L(s) = L_s^{\text{inc}}$ and $U(s) = U_s^{\text{inc}}$. The selection credit is $C(s) = L(s)$.

After each newly completed matched pair, the procedure recomputes all statistics affected by that pair before making rejection, eligibility, separation, or allocation decisions. It rejects a candidate when $U(s) < \tau_{\text{rej}}$. A non-rejected candidate is eligible when

$$L(s) > \tau_{\text{acc}} \quad \text{and} \quad L_s^{\text{abs}} > 0.$$

Eligibility does not terminate evaluation. Early selection requires an eligible candidate whose incremental lower bound exceeds the incremental upper bound of every other active candidate.

If no candidate satisfies the separation criterion, the procedure selects the next candidate according to

$$A(s) = U(s) + \lambda_{\text{unc}}\big(U(s) - L(s)\big), \qquad s^\star \in \arg\max_{s \in \mathcal{A}} A(s).$$

The next matched execution evaluates the candidate-versus-parent contrast for $s^\star$, which determines $A(s)$ and the final ranking. The matched evaluation record maintains the candidate-versus-none pairs required by Equation (A). A candidate becomes eligible only after its absolute bound is available. Requested module-level comparisons follow the same paired-difference update and allocation rule while preserving the skill-level acceptance conditions.

The algorithm terminates when it exhausts the evaluation budget, rejects all candidates in the active set, or separates an eligible candidate from every other active candidate. At budget exhaustion, the procedure selects only among candidates satisfying both conditions in Equation (A). If no candidate satisfies these conditions, it retains the parent skill.

Algorithm 1 Sequential counterfactual evaluation and selection

Require: Candidate set $\mathcal{C}$, parent skill s^{par}, evaluation budget B, matched initialized collections $\{\mathcal{D}_s^{\mathrm{inc}}, \mathcal{D}_s^{\mathrm{abs}}\}_{s\in\mathcal{C}}$, adjusted confidence levels $\{\delta_s^{\mathrm{inc}}, \delta_s^{\mathrm{abs}}\}_{s\in\mathcal{C}}$, thresholds $\tau_{\mathrm{acc}} \geq \tau_{\mathrm{rej}}$, and allocation coefficient λ_{unc}

Ensure: Selected skill $\widehat{s} \in \mathcal{C} \cup \{s^{\mathrm{par}}\}$

Initialize the active set $\mathcal{A} \leftarrow \mathcal{C}$
Initialize the eligible set $\mathcal{E} \leftarrow \varnothing$
Initialize the number of consumed matched pairs b from the matched initialization
for all $s \in \mathcal{A}$ **do**
    Compute $\widehat{\Delta}_s^{\mathrm{inc}}, L_s^{\mathrm{inc}}, U_s^{\mathrm{inc}}$ and $\widehat{\Delta}_s^{\mathrm{abs}}, L_s^{\mathrm{abs}}, U_s^{\mathrm{abs}}$ from the current paired collections
end for
while $b < B$ and $\mathcal{A} \neq \varnothing$ **do**
    Remove every $s \in \mathcal{A}$ satisfying $U_s^{\mathrm{inc}} < \tau_{\mathrm{rej}}$
    Update

$$\mathcal{E} \leftarrow \left\{s \in \mathcal{A} : L_s^{\mathrm{inc}} > \tau_{\mathrm{acc}} \text{ and } L_s^{\mathrm{abs}} > 0\right\}$$

    if $\mathcal{A} = \varnothing$ **then**
        break
    end if
    if $\mathcal{E} \neq \varnothing$ **then**
        Choose

$$\widetilde{s} \in \arg\max_{s\in\mathcal{E}} L_s^{\mathrm{inc}}$$

        if $\mathcal{A} \setminus \{\widetilde{s}\} = \varnothing$ **then**
            return $\widehat{s} \leftarrow \widetilde{s}$
        else if $L_{\widetilde{s}}^{\mathrm{inc}} > \max_{s\in\mathcal{A}\setminus\{\widetilde{s}\}} U_s^{\mathrm{inc}}$ **then**
            return $\widehat{s} \leftarrow \widetilde{s}$
        end if
    end if
    Compute $A(s) \leftarrow U_s^{\mathrm{inc}} + \lambda_{\mathrm{unc}}(U_s^{\mathrm{inc}} - L_s^{\mathrm{inc}})$ for every $s \in \mathcal{A}$
    Choose $s^\star \in \arg\max_{s\in\mathcal{A}} A(s)$
    Execute one fresh matched pair under $i_{\mathrm{cand}}(s^\star)$ and $i_{\mathrm{par}}(s^\star)$, preserving the matched task and pre-execution state
    Append the resulting reward difference to $\mathcal{D}_{s^\star}^{\mathrm{inc}}$ and set $b \leftarrow b + 1$
    Update $\widehat{\Delta}_{s^\star}^{\mathrm{inc}}, L_{s^\star}^{\mathrm{inc}}, U_{s^\star}^{\mathrm{inc}}$ using Equation (13)
    Update any absolute-credit collection completed by the matched evaluation record, then update its estimate and bounds
end while
Recompute

$$\mathcal{E} \leftarrow \left\{s \in \mathcal{A} : L_s^{\mathrm{inc}} > \tau_{\mathrm{acc}} \text{ and } L_s^{\mathrm{abs}} > 0\right\}$$

if $\mathcal{E} = \varnothing$ **then**
    return $\widehat{s} \leftarrow s^{\mathrm{par}}$
else
    return $\widehat{s} \in \arg\max_{s\in\mathcal{E}} L_s^{\mathrm{inc}}$
end if

B Environment Details

B.1 Datasets and Task Settings

We evaluate Counterfactual Evolution Credit (CEC) on three task suites covering heterogeneous agent workflows. SkillsBench-CEC contains 60 tasks spanning document editing, data analysis, planning, coding, and specialized knowledge workflows. SpreadsheetBench-CEC contains 120 stateful spreadsheet-manipulation tasks with instrumented skill invocations, executable action traces, and workbook-level verification. SearchQA-CEC contains 200 retrieval-intensive questions requiring multi-step search, evidence aggregation, and short-form answer generation.

Each suite is partitioned into evolution, validation, and held-out test subsets using a stratified 40/20/40 split. The evolution subset is used exclusively to collect execution trajectories and generate candidate skill revisions. The validation subset is used for counterfactual credit estimation, sequential allocation, acceptance decisions, and stopping. The test subset is accessed once after the candidate pool, thresholds, and evaluation procedure have been fixed. Splits are stratified by task category and estimated difficulty, and tasks derived from the same template or source artifact are assigned to the same partition.

Document, coding, and spreadsheet tasks are evaluated from their original environment states and require the agent to produce the requested artifact or environment modification. Search tasks require a concise final answer accompanied by the evidence collected during execution. Every episode begins with the task instruction, the tool specification, and the currently deployed skill repository. The agent is prompted to reason step by step, invoke tools when necessary, and place its final response in a dedicated `FINAL` field. Skill metadata, including the skill name, description, and trigger conditions, is exposed through the retrieval interface; the full skill body is inserted into the context only after invocation.

We use a maximum of 30 agent turns per execution. Tool calls, invalid actions, and recovery attempts each consume one turn. All intervention conditions for a matched pair share the task instance, initial environment snapshot, accessible tools, context-window limit, and turn budget. To reduce contamination, candidate skills may use only trajectories from the evolution partition. Task identifiers, reference answers, verifier outputs, and held-out artifacts are excluded from skill-generation prompts. We additionally remove candidates containing verbatim spans of more than 20 tokens from validation or test instructions and reject skills that encode instance identifiers, expected answers, or partition-specific file names.

B.2 Preprocessing and Evaluation

Task inputs are converted to a common record containing the instruction, initial environment snapshot, tool configuration, execution budget, reference specification, and deterministic verifier. Unicode text is normalized with NFKC, line endings are canonicalized, and non-semantic leading and trailing whitespace is removed. We preserve whitespace when it affects executable code, formulas, document layout, or spreadsheet contents.

SearchQA answers are evaluated after lowercasing, Unicode normalization, removal of surrounding punctuation and articles, and collapse of repeated whitespace. Numeric answers are canonicalized by removing thousands separators, normalizing signs and decimal representations, and converting equivalent fractions, percentages, and scientific notation to a common numeric value. LaTeX delimiters and wrappers such as `\boxed{}`, `\text{}`, and `\mathrm{}` are removed before comparison. A prediction is correct if it exactly matches any normalized reference answer or, for numeric questions, differs from a reference value by at most 10^{-6} relative error and 10^{-8} absolute error.

Artifact-producing tasks are scored by deterministic task-specific verifiers. Spreadsheet tasks are checked against the final workbook state, including cell values, formulas, sheet structure, formatting constraints, and required charts or tables. Document and coding tasks use executable assertions over the generated artifact, with partial verifier scores averaged into a reward in $[0, 1]$. An execution receives task success when all required assertions pass. Invalid artifacts, exhausted turn budgets, uncaught tool errors, and missing final outputs receive zero reward.

The primary metric is held-out task success, reported as the percentage of test tasks completed successfully. We additionally report true improvement, defined as the proportion of selected skills with positive mean paired test effect; false selection, defined as the proportion with non-positive paired test effect; executions per validated improvement; and module-localization accuracy. A harmful module is localized correctly when it receives the lowest candidate-versus-module-replacement credit among all modules in the skill.

For every counterfactual comparison, the candidate and reference conditions are evaluated on the same task and environment snapshot. Their random seeds are matched whenever sampling and tool backends expose seed control. Each initial candidate receives two matched pairs against its parent and one pair against the no-skill condition. Candidate-versus-blocked and module-replacement comparisons are added when invocation or module attribution is required. Sequential allocation

continues until the execution budget is exhausted or the confidence-based separation criterion is satisfied. Confidence bounds use the empirical Bernstein radius with family-wise confidence level $1 - \delta = 0.95$; δ is divided across active comparisons using a Bonferroni correction. We set $\tau_{\text{acc}} = 0.02$, $\tau_{\text{rej}} = 0$, and $\lambda_{\text{unc}} = 0.5$.

All reported results are aggregated over five independent experiment seeds. We report the arithmetic mean and sample standard deviation across seeds. The total validation budget is 240 agent executions per task suite unless otherwise specified; budget ablations use 120 and 360 executions. Baselines are NO-SKILL, EPISODE-REWARD, BEST-OF-N, FRESH-AUDIT, LEAVE-ONE-OUT, and COUNTERFACTUAL-UNIFORM. Every selection method receives the same candidate pool, executor configuration, validation tasks, and total number of executions.

B.3 IMPLEMENTATION AND HARDWARE

We implement the agent harness in Python 3.11. The primary executor, denoted Agent-A, uses `Qwen3-32B` in instruction-following mode and is served locally with vLLM using bfloat16 weights. The same backbone performs planning, tool selection, skill retrieval, skill invocation, and final-answer generation. Candidate generation uses a separate `Qwen3-32B` context with access only to evolution trajectories, verifier feedback, and the parent skill. Candidate generation is performed once before method comparison, ensuring that all selectors operate on an identical frozen candidate pool.

Each parent produces eight candidate revisions. A candidate is represented as a structured artifact with four modules: metadata and trigger conditions, procedural instructions, verification logic, and failure-recovery logic. Candidate generation uses temperature 0.8, top-$p = 0.95$, and a maximum of 4,096 generated tokens. Agent execution uses temperature 0.7, top-$p = 0.9$, and a maximum of 2,048 tokens per model response. Matched conditions reuse the same sampling seed and preserve the ordering of tool observations. The context window is limited to 32,768 tokens. When the accumulated trajectory exceeds this limit, the oldest non-system observations are compressed into a structured summary while retaining the task instruction, active skill, latest tool state, and all unresolved verifier failures.

CEC does not train a separate value or reward model. Task rewards are produced by deterministic environment verifiers, and selection scores are computed directly from paired reward differences. Episode-level baselines use the same verifier reward. Fresh-audit additionally checks whether the candidate was retrieved and invoked. Behavioral influence is detected by comparing normalized action traces between the candidate and blocked-invocation conditions. Two traces are considered behaviorally different if they differ in tool name, normalized arguments, action order, or final artifact hash; differences limited to timestamps, request identifiers, or non-semantic formatting are ignored.

The no-skill intervention removes candidate-specific content while preserving the retrieval interface and all unrelated skills. The parent intervention replaces the complete candidate with its immediate predecessor. The blocked intervention exposes identical metadata and permits retrieval, but returns an interface-compatible empty body when invocation occurs. For module-level interventions, exactly one candidate module is replaced by the corresponding parent module; if no parent module exists, it is replaced by a typed null module that preserves the artifact schema.

Sequential evaluation uses an initial branching factor of eight candidates per parent. The search depth is one revision generation step per selection round, and the main experiments perform three evolution rounds. At each round, only the accepted candidate, or the retained parent when no candidate is accepted, is propagated to the next round. Allocation is performed in units of matched execution pairs. Ties in allocation score are broken by larger empirical incremental credit and then by a fixed candidate identifier. Cached model outputs are disabled, while immutable environment snapshots and tool responses are cached when exact replay is supported.

For cross-agent reproduction, Agent-B uses `Llama-3.3-70B-Instruct` and Agent-C uses `Mistral-Small-3.1-24B-Instruct`. Both are locally served with the same system prompt, tools, skill repository, context limit, decoding parameters, and execution budget as Agent-A. These models are used only to execute skills selected by Agent-A; they do not regenerate, revise, or reselect candidates.

Experiments are run on a Linux cluster with four NVIDIA H100 80 GB GPUs per worker, dual 64-core AMD EPYC processors, and 512 GB of host memory. The primary 32B model uses two-way

Variant	Success (%) ↑	TI (%) ↑	FS (%) ↓	Localization (%) ↑	Exec. per Improve ↓
CEC	$\mathbf{57.2 \pm 1.1}$	$\mathbf{74.8 \pm 1.7}$	$\mathbf{8.9 \pm 1.1}$	$\mathbf{84.7 \pm 1.6}$	$\mathbf{31.6 \pm 1.4}$
w/o shared seeds	53.8 ± 1.6	66.2 ± 2.3	15.8 ± 1.7	68.9 ± 2.5	37.4 ± 2.1
single paired replay	54.9 ± 1.5	70.4 ± 2.0	14.1 ± 1.5	72.6 ± 2.2	34.7 ± 1.8
w/o confidence bounds	55.6 ± 1.3	71.5 ± 1.9	12.7 ± 1.4	80.1 ± 1.9	33.8 ± 1.6
skill-level attribution only	55.8 ± 1.2	72.3 ± 1.8	11.8 ± 1.3	–	34.1 ± 1.7
w/o invocation blocking	55.2 ± 1.4	70.9 ± 2.1	13.6 ± 1.6	76.8 ± 2.0	35.3 ± 1.9
uniform allocation	54.7 ± 1.3	72.1 ± 1.6	11.2 ± 1.2	79.4 ± 1.8	38.9 ± 1.9

Table 3: Complete ablation results averaged across the three task suites. Values are reported as mean ± standard deviation over five seeds. Abbreviations: TI = true improvement; FS = false selection; Exec. = executions.

	120 Executions		240 Executions		360 Executions	
Method	Success ↑	FS ↓	Success ↑	FS ↓	Success ↑	FS ↓
CF-Uniform	52.2 ± 1.6	15.7 ± 1.7	54.7 ± 1.3	11.2 ± 1.2	56.3 ± 1.2	9.6 ± 1.1
CEC	$\mathbf{53.9 \pm 1.5}$	$\mathbf{12.8 \pm 1.5}$	$\mathbf{57.2 \pm 1.1}$	$\mathbf{8.9 \pm 1.1}$	$\mathbf{58.0 \pm 1.0}$	$\mathbf{7.8 \pm 0.9}$

Table 4: Average test success and false-selection rate under different evaluation budgets. All results are percentages reported as mean ± standard deviation over five seeds.

tensor parallelism, allowing two independent model replicas per worker. The 70B transfer executor uses four-way tensor parallelism. Spreadsheet and document environments run in isolated CPU containers with 8 virtual CPU cores and 16 GB memory per episode. We use CUDA 12.4, PyTorch 2.6, Transformers 4.51, and vLLM 0.8.5. Each execution has a wall-clock timeout of 20 minutes, individual tool calls have a 120-second timeout, and failed infrastructure calls are retried once with the same intervention and seed before being recorded as execution failures.

C Additional Experimental Results

C.1 Complete Component Ablations

Table 3 reports the complete component ablation results. Removing shared seeds reduces average test success by 3.4 percentage points and increases false selection by 6.9 percentage points. Removing confidence bounds increases false selection from 8.9% to 12.7%. Module interventions and sequential allocation affect localization accuracy and the number of executions per improvement, respectively.

C.2 Sensitivity to Evaluation Budget

Table 4 compares sequential and uniform allocation across the evaluated budgets. CEC exceeds CF-Uniform in test success by 1.7, 2.5, and 1.7 percentage points at budgets of 120, 240, and 360 executions, respectively. Increasing the budget from 240 to 360 executions raises the test success of CEC from 57.2% to 58.0%.

C.3 Module-Level Attribution

Table 5 evaluates whether the module with the lowest credit corresponds to an injected harmful component. Recovery interventions yield higher localization accuracy than trigger interventions across all evaluated methods. For CEC, the localization accuracies for recovery and trigger interventions are 89.2% and 82.1%, respectively. This difference is associated with the longer shared trajectory prefixes preceding recovery interventions. Joint corruption introduces non-additive interactions; under this setting, CEC achieves an average localization accuracy of 84.7%, compared with 66.4% for unmatched module replacement.

Method	Trigger	Procedure	Verification	Recovery	Average
Whole-skill removal	41.7 ± 3.0	38.6 ± 2.8	40.4 ± 2.7	43.1 ± 3.1	41.0 ± 2.4
Unmatched module replacement	62.8 ± 2.7	66.5 ± 2.5	64.9 ± 2.6	71.4 ± 2.3	66.4 ± 2.1
CEC w/o shared seeds	65.1 ± 2.8	68.2 ± 2.5	67.3 ± 2.4	75.0 ± 2.2	68.9 ± 2.5
CEC	$\mathbf{82.1 \pm 1.9}$	$\mathbf{84.0 \pm 1.8}$	$\mathbf{83.5 \pm 1.7}$	$\mathbf{89.2 \pm 1.5}$	$\mathbf{84.7 \pm 1.6}$

Table 5: Harmful-module localization accuracy (%). Values are reported as mean $\pm$ standard deviation over five seeds. A prediction is correct when the injected harmful module receives the lowest estimated module credit.

Module Pair	Marginal Sum	Joint Credit	Whole-Skill Effect	Absolute Error $\downarrow$
Trigger + Procedure	4.8 ± 0.9	3.6 ± 0.8	3.4 ± 0.7	0.2 ± 0.2
Trigger + Recovery	1.7 ± 1.1	-0.4 ± 0.9	-0.7 ± 0.8	0.3 ± 0.2
Procedure + Verification	5.9 ± 0.8	4.5 ± 0.7	4.3 ± 0.6	0.2 ± 0.1
Verification + Recovery	2.4 ± 1.0	0.8 ± 0.8	0.5 ± 0.7	0.3 ± 0.2

Table 6: Module interaction analysis. Credits and effects are percentage-point reward differences reported as mean $\pm$ standard deviation over five seeds. Absolute error compares joint credit with the observed whole-skill effect.

Table 6 reports interactions between two simultaneously modified modules. The sum of marginal module credits overestimates the whole-skill effect for each evaluated module pair. Joint interventions produce absolute errors between 0.2 and 0.3 percentage points relative to the observed whole-skill effect.

C.4 Influence-Path Diagnostics

Table 7 decomposes candidate influence into retrieval, invocation, behavioral change, and positive outcome change. Genuinely beneficial candidates yield behavior-change and outcome-improvement rates of 66.9% and 43.7%, respectively. Inert-call deceptive candidates have an invocation rate of 76.9% but behavior-change and outcome-improvement rates of 4.8% and 2.6%, respectively. These results distinguish invocation frequency from behavioral and outcome effects.

C.5 Cross-Agent Reproduction

Table 8 reports cross-agent reproduction using skills selected by Agent-A. Skills selected by CEC yield test success rates of 57.2%, 54.9%, and 53.2% on Agent-A, Agent-B, and Agent-C, respectively. Their cross-agent true-improvement rate is 72.6%. The reduction from Agent-A is 2.3 percentage points for Agent-B and 4.0 percentage points for Agent-C. Errors on Agent-C are concentrated among skills containing executor-specific tool syntax.

C.6 Robustness to Selection Thresholds

Table 9 reports the results across acceptance thresholds from 0.00 to 0.06. Increasing the threshold from 0.00 to 0.06 reduces the false-selection rate from 11.9% to 6.9% and increases the parent-retention rate from 18.4% to 40.7%. Test success reaches 57.2% at a threshold of 0.02 and decreases to 55.8% at a threshold of 0.06.

C.7 Replayability and Residual Stochasticity

Table 10 stratifies the results by replayability. Under exact deterministic replay, CEC reduces the false-selection rate from 14.8% for unmatched evaluation to 5.7%. Under partially replayable environments, the corresponding rates are 18.7% and 12.6%. Partially replayable retrieval environments have lower paired correlation and higher paired-effect variance because hidden service state and nondeterministic responses cannot be restored exactly.

Candidate Type	Retrieved (%)	Invoked (%)	Behavior Changed (%)	Outcome Improved (%)
Genuinely beneficial	89.6 ± 1.7	78.4 ± 2.1	66.9 ± 2.3	43.7 ± 2.0
Neutral revision	73.1 ± 2.2	55.8 ± 2.5	18.6 ± 1.9	10.4 ± 1.5
Uncalled deceptive	17.5 ± 1.8	2.7 ± 0.7	1.2 ± 0.5	0.8 ± 0.4
Inert-call deceptive	84.2 ± 1.9	76.9 ± 2.2	4.8 ± 1.0	2.6 ± 0.8
Spurious-success deceptive	77.8 ± 2.0	62.3 ± 2.4	22.9 ± 2.0	8.7 ± 1.3

Table 7: Influence-path statistics under candidate-versus-blocked comparisons. Values are reported as mean $\pm$ standard deviation over five seeds.

Selection Method	Agent-A	Agent-B	Agent-C	Cross-Agent TI (%)
Episode-Reward	51.0 ± 1.5	47.6 ± 1.8	46.2 ± 1.9	55.1 ± 2.5
Leave-One-Out	53.8 ± 1.3	50.4 ± 1.6	48.9 ± 1.8	64.7 ± 2.1
CF-Uniform	54.7 ± 1.3	51.8 ± 1.5	50.5 ± 1.7	68.5 ± 1.9
CEC	$\mathbf{57.2 \pm 1.1}$	$\mathbf{54.9 \pm 1.4}$	$\mathbf{53.2 \pm 1.6}$	$\mathbf{72.6 \pm 1.8}$

Table 8: Cross-agent test success (%) and true-improvement rate for skills selected on Agent-A. Values are reported as mean $\pm$ standard deviation over five seeds.

D Prompt Templates

D.1 Overview

This appendix specifies the prompt templates used to execute agents, expose skill artifacts, construct counterfactual intervention conditions, generate candidate skills, and audit execution traces. The task instruction, initial environment state, tool interface, execution budget, and random seed are held fixed within each matched comparison. Intervention-specific templates differ only in the skill content made available to the agent. Candidate selection and final evaluation use programmatic verifiers rather than an LLM-as-a-judge.

D.2 Notation

We use the following placeholders throughout the templates:

- {TASK_INSTRUCTION}: the natural-language task instruction.
- {INITIAL_OBSERVATION}: the initial observation obtained from the restored environment snapshot.
- {TOOL_SPECIFICATIONS}: the names, argument schemas, and usage constraints of the available tools.
- {EXECUTION_BUDGET}: the maximum number of tool calls or environment interactions.
- {TURN_LIMIT}: the maximum number of agent turns, set to 30 in the experiments.
- {SKILL_NAME} and {SKILL_DESCRIPTION}: the identifier and retrieval metadata of a skill.
- {SKILL_BODY}: the operational content of a skill, including its trigger, procedure, constraints, and recovery logic.
- {PARENT_SKILL}: the complete parent artifact from which a candidate was derived.
- {CANDIDATE_SKILL}: the complete candidate artifact under evaluation.
- {MODULE_NAME}: the functional module modified by a module-level intervention.
- {REPLACEMENT_MODULE}: the corresponding parent module or a null module preserving the original interface.
- {EXECUTION_TRACE}: the sequence of observations, actions, tool calls, and terminal output from an episode.
- {VERIFIER_RESULT}: the task-specific programmatic verifier output in $[0, 1]$.

τ_{acc}	Success (%) ↑	TI (%) ↑	FS (%) ↓	Parent Retained (%)
0.00	56.2 ± 1.3	71.9 ± 1.9	11.9 ± 1.4	18.4 ± 1.8
0.01	56.8 ± 1.2	73.6 ± 1.8	10.1 ± 1.2	21.7 ± 1.9
0.02	$\mathbf{57.2 \pm 1.1}$	74.8 ± 1.7	8.9 ± 1.1	25.6 ± 2.0
0.04	56.7 ± 1.2	$\mathbf{76.1 \pm 1.6}$	7.5 ± 1.0	32.8 ± 2.2
0.06	55.8 ± 1.4	76.4 ± 1.8	$\mathbf{6.9 \pm 0.9}$	40.7 ± 2.5

Table 9: Sensitivity to the acceptance threshold with $\tau_{\mathrm{rej}} = 0$. Values are reported as mean ± standard deviation over five seeds.

Replay Regime	Paired Correlation	Effect Std. ↓	CEC FS (%) ↓	Unmatched FS (%) ↓
Exact deterministic replay	0.91 ± 0.02	0.18 ± 0.01	5.7 ± 0.9	14.8 ± 1.5
Restorable stochastic replay	0.74 ± 0.03	0.27 ± 0.02	8.4 ± 1.1	16.9 ± 1.7
Partially replayable environment	0.51 ± 0.04	0.36 ± 0.03	12.6 ± 1.5	18.7 ± 1.9

Table 10: Selection reliability under different replay regimes. Effect standard deviation is computed over matched reward differences. Values are reported as mean ± standard deviation over five seeds. Abbreviations: Std. = standard deviation; FS = false selection.

- {RETRIEVAL_EVENTS} and {INVOCATION_EVENTS}: recorded skill-retrieval and skill-invocation events.
- {CANDIDATE_POOL}: the set of candidate skills available to a selection baseline.
- {EVOLUTION_TRAJECTORIES}: successful and failed trajectories from the evolution split.
- {EXECUTION_FEEDBACK}: verifier results and execution diagnostics associated with the evolution trajectories.

D.3 Main Inference Prompt

Purpose. Execute a task with an operational candidate skill under the candidate intervention i_{cand}.

Inputs. The task instruction, initial observation, tool specifications, execution budget, turn limit, and candidate skill.

Expected Output. A sequence of valid tool actions followed by a terminal answer or task completion signal. Skill retrieval and invocation events are recorded by the execution harness rather than emitted in the terminal answer.

Prompt.

```
You are an autonomous language-model agent operating in an
interactive environment. Complete the assigned task using the
available tools and the reusable skill provided below.

General execution requirements:
1. Follow the task instruction exactly.
2. Inspect the current environment state before acting.
3. Use only the tools listed in the tool specification.
4. Do not exceed the execution budget or turn limit.
5. Treat the skill as procedural guidance rather than as evidence
   that the task has already been completed.
6. Invoke the skill only when its trigger conditions are satisfied.
7. Verify the resulting environment state before returning the final
   answer.
8. If a tool action fails, use the recovery logic in the skill when
   applicable; otherwise, revise the plan using the observed error.
9. Do not claim completion unless the requested result has been
```

```
   produced in the environment.

Task instruction:
{TASK_INSTRUCTION}

Initial observation:
{INITIAL_OBSERVATION}

Available tools:
{TOOL_SPECIFICATIONS}

Execution budget:
{EXECUTION_BUDGET}

Turn limit:
{TURN_LIMIT}

Available skill:
Name: {SKILL_NAME}
Description: {SKILL_DESCRIPTION}

Skill content:
{SKILL_BODY}

Begin execution.
```

D.4 Baseline Prompts

D.4.1 No-Skill Execution

Purpose. Execute the task without exposing candidate-specific or parent-skill content.

Inputs. The task instruction, initial observation, tool specifications, execution budget, and turn limit.

Expected Output. A sequence of valid tool actions followed by a terminal answer or task completion signal.

Prompt.

```
You are an autonomous language-model agent operating in an
interactive environment. Complete the assigned task using the
available tools.

General execution requirements:
1. Follow the task instruction exactly.
2. Inspect the current environment state before acting.
3. Use only the tools listed in the tool specification.
4. Do not exceed the execution budget or turn limit.
5. Verify the resulting environment state before returning the final
   answer.
6. If a tool action fails, revise the plan using the observed error.
7. Do not claim completion unless the requested result has been
   produced in the environment.

Task instruction:
{TASK_INSTRUCTION}

Initial observation:
{INITIAL_OBSERVATION}
```

```
Available tools:
{TOOL_SPECIFICATIONS}

Execution budget:
{EXECUTION_BUDGET}

Turn limit:
{TURN_LIMIT}

No reusable task-specific skill is available for this execution.

Begin execution.
```

D.4.2 EPISODE-REWARD AND BEST-OF-N EXECUTION

Purpose. Produce execution outcomes used by the EPISODE-REWARD and BEST-OF-N baselines. These baselines differ in their selection rules, not in the agent prompt.

Inputs. The task instruction, environment information, execution constraints, and one candidate skill from {CANDIDATE_POOL}.

Expected Output. A tool-action trajectory and terminal output scored by the task-specific verifier.

Prompt.

```
You are an autonomous language-model agent. Complete the task using
the available tools and the candidate skill below.

Task instruction:
{TASK_INSTRUCTION}

Initial observation:
{INITIAL_OBSERVATION}

Available tools:
{TOOL_SPECIFICATIONS}

Execution budget:
{EXECUTION_BUDGET}

Turn limit:
{TURN_LIMIT}

Candidate skill:
{CANDIDATE_SKILL}

Use the candidate skill when applicable. Respect all task, tool,
budget, and environment constraints. Verify the final state before
reporting completion.

Begin execution.
```

D.4.3 FRESH-AGENT AUDIT

Purpose. Evaluate whether an independent fresh agent retrieves and invokes a candidate skill while attempting the task.

Inputs. The task instruction, environment information, execution constraints, and candidate skill.

Expected Output. A normal execution trajectory. Retrieval and invocation indicators are recorded by the harness and used by the baseline audit.

Prompt.

```
You are a fresh autonomous agent with no memory of previous
executions. Complete the task using the available tools and the
reusable skill repository entry shown below.

Do not assume that the skill is relevant. Determine relevance from
its trigger and the current task. If relevant, invoke and follow the
skill. If irrelevant, solve the task without invoking it.

Task instruction:
{TASK_INSTRUCTION}

Initial observation:
{INITIAL_OBSERVATION}

Available tools:
{TOOL_SPECIFICATIONS}

Execution budget:
{EXECUTION_BUDGET}

Turn limit:
{TURN_LIMIT}

Skill repository entry:
Name: {SKILL_NAME}
Description: {SKILL_DESCRIPTION}
Content:
{SKILL_BODY}

Verify the final state and return the task result.

Begin execution.
```

D.4.4 LEAVE-ONE-OUT EXECUTION

Purpose. Estimate whole-skill contribution by comparing execution with the complete candidate skill against execution with that skill removed.

Inputs. The matched task and environment configuration, together with either the complete candidate skill or no skill.

Expected Output. A pair of independently generated execution trajectories and terminal outputs.

Prompt. The skill-present condition uses the template in Section D.3. The skill-removed condition uses the following template.

```
You are an autonomous language-model agent operating in an
interactive environment. Complete the assigned task using the
available tools.

Task instruction:
{TASK_INSTRUCTION}

Initial observation:
{INITIAL_OBSERVATION}
```

```
Available tools:
{TOOL_SPECIFICATIONS}

Execution budget:
{EXECUTION_BUDGET}

Turn limit:
{TURN_LIMIT}

The candidate skill has been removed for this execution. No content,
resource, instruction, or recovery procedure from that skill is
available. Solve the task using the base agent and the listed tools.
Verify the final state before reporting completion.

Begin execution.
```

D.4.5 COUNTERFACTUAL-UNIFORM EXECUTION

Purpose. Run the same candidate, no-skill, parent, blocked-invocation, and module-replacement interventions as CEC while allocating matched pairs uniformly across candidates.

Inputs. The matched task configuration and one intervention-specific skill representation.

Expected Output. A normal execution trajectory and terminal output for each intervention condition.

Prompt.

```
You are an autonomous language-model agent operating in an
interactive environment. Complete the task using the tools and skill
configuration provided for this execution.

Task instruction:
{TASK_INSTRUCTION}

Initial observation:
{INITIAL_OBSERVATION}

Available tools:
{TOOL_SPECIFICATIONS}

Execution budget:
{EXECUTION_BUDGET}

Turn limit:
{TURN_LIMIT}

Intervention-specific skill configuration:
{INTERVENTION_SKILL_CONTENT}

Follow only the skill content exposed in this configuration. Do not
infer, reconstruct, or use content omitted by the intervention.
Verify the final environment state before returning the result.

Begin execution.
```

D.5 Few-Shot and Demonstration Prompts

No few-shot demonstrations are included in the inference, counterfactual evaluation, baseline, or selection prompts. All execution conditions use zero-shot task instructions together with the intervention-specific skill artifact.

D.6 Reasoning and Rationale Prompts

D.6.1 Action-Level Planning and Execution

Purpose. Standardize the agent's decision process across intervention conditions without requiring a separately scored natural-language rationale.

Inputs. The current observation, task instruction, available tools, remaining budget, and exposed skill configuration.

Expected Output. Either one valid tool call or a terminal answer. Any internal rationale is not used as an evaluation target.

Prompt.

```
At each turn, use the current observation to determine the next
action.

Current task:
{TASK_INSTRUCTION}

Current observation:
{CURRENT_OBSERVATION}

Remaining tools and schemas:
{TOOL_SPECIFICATIONS}

Remaining execution budget:
{REMAINING_BUDGET}

Exposed skill configuration:
{INTERVENTION_SKILL_CONTENT}

Select the next action that most directly advances the task while
respecting the tool schema and the exposed skill configuration.
Before terminating, inspect or verify the relevant final state.

Return exactly one of the following:
1. A valid tool call in the tool interface format.
2. A terminal answer if and only if the task is complete.
```

D.6.2 Blocked-Invocation Intervention

Purpose. Implement i_{blk} by exposing identical retrieval metadata while withholding the operational skill body and associated resources after invocation.

Inputs. The task configuration and the metadata of the blocked candidate skill.

Expected Output. A normal execution trajectory. An attempted invocation is recorded, but the agent receives no operational skill content.

Prompt.

```
You are an autonomous language-model agent operating in an
interactive environment. Complete the assigned task using the
available tools.

Task instruction:
{TASK_INSTRUCTION}

Initial observation:
{INITIAL_OBSERVATION}

Available tools:
{TOOL_SPECIFICATIONS}

Execution budget:
{EXECUTION_BUDGET}

Turn limit:
{TURN_LIMIT}

Skill repository entry:
Name: {SKILL_NAME}
Description: {SKILL_DESCRIPTION}

The repository entry may be retrieved normally. However, the
operational body and associated resources of this skill are
unavailable in this execution. If you invoke the entry, record the
invocation and continue using only the task instruction, current
observations, and available tools. Do not reconstruct or assume the
missing skill content.

Verify the final state before returning the task result.

Begin execution.
```

D.6.3 PARENT-REPLACEMENT INTERVENTION

Purpose. Implement i_{par} by replacing the candidate with its immediate parent while preserving the remaining execution configuration.

Inputs. The task configuration and complete parent skill.

Expected Output. A tool-action trajectory and terminal output produced with the parent skill.

Prompt.

```
You are an autonomous language-model agent operating in an
interactive environment. Complete the assigned task using the
available tools and the parent skill provided below.

Task instruction:
{TASK_INSTRUCTION}

Initial observation:
{INITIAL_OBSERVATION}

Available tools:
{TOOL_SPECIFICATIONS}

Execution budget:
{EXECUTION_BUDGET}
```

```
Turn limit:
{TURN_LIMIT}

Parent skill:
{PARENT_SKILL}

Use only the parent skill shown above. No candidate-specific revision
is available in this execution. Apply the skill only when its trigger
conditions are satisfied, and verify the resulting environment state
before returning the final answer.

Begin execution.
```

D.6.4 MODULE-REPLACEMENT INTERVENTION

Purpose. Implement i_{-m} by replacing exactly one candidate module with the corresponding parent module or a null module.

Inputs. The task configuration, candidate skill, name of the intervened module, and replacement module.

Expected Output. A tool-action trajectory and terminal output produced with the hybrid skill artifact.

Prompt.

```
You are an autonomous language-model agent operating in an
interactive environment. Complete the assigned task using the
available tools and the intervention-specific skill below.

Task instruction:
{TASK_INSTRUCTION}

Initial observation:
{INITIAL_OBSERVATION}

Available tools:
{TOOL_SPECIFICATIONS}

Execution budget:
{EXECUTION_BUDGET}

Turn limit:
{TURN_LIMIT}

Candidate skill under module intervention:
{CANDIDATE_SKILL_WITH_REPLACEMENT}

Intervened module:
{MODULE_NAME}

Replacement content:
{REPLACEMENT_MODULE}

Treat the displayed artifact as the complete skill for this
execution. Use the replacement content only for the named module and
retain all other candidate modules unchanged. Do not reconstruct the
original candidate module. Verify the final environment state before
returning the result.
```

```
Begin execution.
```

D.6.5 NULL MODULE

Purpose. Preserve the module interface when neither the parent nor the repository contains a corresponding operational module.

Inputs. The module name and its interface description.

Expected Output. A non-operational module that introduces no task-specific behavior.

Prompt.

```
Module: {MODULE_NAME}

This module intentionally contains no operational instructions.
Preserve the surrounding skill interface, but do not trigger any
action, introduce any procedure, impose any additional constraint, or
apply any recovery behavior from this module.
```

D.7 DATA GENERATION AND PSEUDO-LABELING PROMPTS

D.7.1 CANDIDATE SKILL GENERATION

Purpose. Convert evolution trajectories and execution feedback into a structured candidate skill. Candidate generation is external to CEC and is shared by all selection methods.

Inputs. Evolution trajectories, verifier results, execution feedback, and an optional parent skill.

Expected Output. One structured candidate artifact containing retrieval metadata and the trigger, procedure, constraints, verification, and recovery modules.

Prompt.

```
Construct one reusable agent skill from the execution evidence below.
The skill must encode procedures supported by the evidence and must
not include instance-specific answers.

Parent skill, if available:
{PARENT_SKILL}

Evolution trajectories:
{EVOLUTION_TRAJECTORIES}

Verifier results and execution feedback:
{EXECUTION_FEEDBACK}

Requirements:
1. Preserve useful behavior from the parent skill unless the evidence
   supports a revision.
2. State trigger conditions that identify when the skill applies.
3. Provide an ordered operational procedure.
4. Specify tool-use and environment constraints.
5. Include checks for verifying task completion.
6. Include recovery logic only for failures observed in, or directly
   supported by, the provided evidence.
7. Do not claim that a procedure is effective solely because it
```

```
   appeared in a successful episode.
8. Do not include benchmark labels, verifier outputs, task answers, or
   references to individual evaluation instances.
9. Keep functional modules separable so that each module can be
   replaced independently.

Return exactly the following structure:

{
    "name": "<concise skill name>",
    "description": "<retrieval description>",
    "trigger": {
        "applicable_when": [
            "<condition>"
        ],
        "not_applicable_when": [
            "<condition>"
        ]
    },
    "procedure": [
        "<ordered operational step>"
    ],
    "constraints": [
        "<tool, budget, safety, or environment constraint>"
    ],
    "verification": [
        "<completion or correctness check>"
    ],
    "recovery": [
        {
            "failure_condition": "<observable failure>",
            "response": "<recovery action>"
        }
    ]
}
```

D.7.2 DECEPTIVE-SKILL CONSTRUCTION

Purpose. Instantiate the controlled uncalled, inert-call, and spurious-success candidates used to evaluate attribution failures.

Inputs. A base candidate skill, a deception type, and task metadata from the evolution split.

Expected Output. A structured skill artifact with zero intended causal contribution and the specified controlled failure mode.

Prompt.

```
Create one controlled diagnostic variant of the skill below.

Base skill:
{BASE_SKILL}

Diagnostic type:
{DECEPTION_TYPE}

Allowed diagnostic types:
- uncalled: define a trigger that is not satisfied by the target task,
  while leaving all other fields interface-compatible;
- inert_call: permit invocation but provide no instruction that changes
  the agent's action sequence;
```

```
- spurious_success: preserve non-causal metadata associated with the
  supplied easy-instance subset without adding an operational
  procedure that improves task execution.

Task metadata:
{TASK_METADATA}

Requirements:
1. Preserve the standard skill schema.
2. Do not add a genuinely useful procedure.
3. Do not insert task answers or verifier information.
4. Modify only the fields needed to realize the specified diagnostic
   type.
5. Return one skill artifact using the same JSON structure as the
   candidate-generation template.

Return only the JSON skill artifact.
```

No pseudo-labeling prompt is used. Rewards, positive paired test effects, non-positive effects, retrieval events, invocation events, and module-localization targets are obtained from programmatic verifiers and instrumented execution traces.

D.8 Evaluation Prompts

D.8.1 Execution Audit Record

Purpose. Serialize the observable outputs of each intervention run for paired credit estimation and diagnostic analysis.

Inputs. The intervention identifier, task identifier, replicate identifier, reward, retrieval and invocation events, execution trace, and terminal output.

Expected Output. A JSON audit record. The record is produced by the execution harness and is not shown to the task-solving agent.

Prompt.

```
{
    "task_id": "{TASK_ID}",
    "candidate_id": "{CANDIDATE_ID}",
    "intervention": "{INTERVENTION_ID}",
    "replicate_id": "{REPLICATE_ID}",
    "environment_snapshot_id": "{SNAPSHOT_ID}",
    "random_seed": "{RANDOM_SEED}",
    "execution_budget": {EXECUTION_BUDGET},
    "turn_limit": {TURN_LIMIT},
    "retrieved": {RETRIEVAL_INDICATOR},
    "invoked": {INVOCATION_INDICATOR},
    "reward": {VERIFIER_RESULT},
    "terminal_output": "{TERMINAL_OUTPUT}",
    "trajectory": {EXECUTION_TRACE}
}
```

D.8.2 Programmatic Evaluation

Purpose. Determine task success and paired counterfactual effects without subjective language-model judgment.

Inputs. The terminal output, final environment state, task-specific verifier, and matched intervention records.

Expected Output. A verifier score in $[0, 1]$ and the paired difference between intervention conditions.

Prompt. No natural-language evaluation prompt is used. For task x, the programmatic verifier computes

$$R(s; x, \omega) = v_x(y_T(\xi(s; x, \omega))) .$$

For matched intervention conditions i and i_0, the evaluation harness computes

$$D_{nj}^{i,i_0} = R_{nj}^{i} - R_{nj}^{i_0}.$$

The verifier does not receive skill identities, selection scores, or counterfactual credit estimates.

D.9 Ablation Prompt Variants

D.9.1 Unmatched-Seed Variant

Purpose. Evaluate the contribution of shared randomness by running the same intervention prompts with independently sampled seeds.

Inputs. The complete intervention prompt and condition-specific random seeds.

Expected Output. Two execution trajectories generated from the same observable pre-execution state but different random seeds.

Prompt.

```
Use the intervention-specific execution prompt defined in
Sections~\ref{app:prompt:main} and
\ref{app:prompt:reasoning} without modification.

Candidate condition seed:
{CANDIDATE_SEED}

Reference condition seed:
{REFERENCE_SEED}

The environment snapshot, task instruction, tool state, execution
budget, and turn limit remain matched. Only the random seed differs
between conditions.
```

D.9.2 Single-Pair Variant

Purpose. Estimate counterfactual credit from one matched execution pair.

Inputs. One candidate-condition run and one reference-condition run using the standard intervention prompts.

Expected Output. One paired reward difference.

Prompt.

```
Run exactly one matched pair using the standard candidate and
reference intervention prompts. Restore the same task input,
environment snapshot, tool state, execution budget, turn limit, and,
```

```
when controllable, random seed before each run.

Return the two audit records to the evaluation harness. Do not perform
additional replicates.
```

D.9.3 MEAN-ONLY SELECTION VARIANT

Purpose. Remove confidence-aware selection while leaving all execution prompts and interventions unchanged.

Inputs. The paired rewards collected with the standard intervention prompts.

Expected Output. An empirical mean counterfactual effect for each candidate.

Prompt. No agent-facing prompt is changed. The selection procedure ranks candidates by the empirical mean incremental credit $\widehat{\Delta}_{i_{\mathrm{cand}},i_{\mathrm{par}}}$ rather than by its lower confidence bound.

D.9.4 SKILL-LEVEL-ONLY ATTRIBUTION VARIANT

Purpose. Evaluate candidate selection without module-specific interventions.

Inputs. Candidate, no-skill, parent, and blocked-invocation configurations.

Expected Output. Whole-skill absolute and incremental credit estimates without module credit.

Prompt.

```
Use only the following intervention-specific execution templates:
1. candidate-skill execution;
2. no-skill execution;
3. parent-replacement execution; and
4. blocked-invocation execution.

Do not construct or execute module-replacement prompts.
```

D.9.5 UNIFORM-ALLOCATION VARIANT

Purpose. Remove sequential allocation while preserving the complete counterfactual intervention suite.

Inputs. The active candidate set and the remaining matched-execution budget.

Expected Output. An equal number of additional matched pairs for each unresolved candidate.

Prompt. No agent-facing prompt is changed. The evaluation harness cycles uniformly over unresolved candidates and applies the same candidate, no-skill, parent, blocked-invocation, and module-replacement templates to each candidate.

D.10 OUTPUT FORMATTING

Agent terminal outputs follow the native format required by each task suite. Tool calls must conform exactly to the schemas supplied in {`TOOL_SPECIFICATIONS`}. Skill artifacts and audit records use valid JSON with double-quoted keys and strings. Candidate-generation outputs must contain exactly the fields `name`, `description`, `trigger`, `procedure`, `constraints`, `verification`, and `recovery`. Missing operational modules are represented by empty arrays rather than omitted keys.

The task-solving agent must not include audit metadata, skill-selection decisions, verifier scores, confidence bounds, or counterfactual credit estimates in its terminal answer. Retrieval events, invocation events, trajectories, rewards, environment snapshot identifiers, and seeds are recorded separately by the execution harness. All matched conditions preserve the same task instruction, initial environment snapshot, tool state, execution budget, and turn limit. Unless explicitly modified by an ablation, matched conditions also use the same random seed and a fresh context.

E CASE STUDY

We present three representative cases to illustrate how Counterfactual Evolution Credit (CEC) separates skill presence, invocation, behavioral influence, and outcome influence. The cases are drawn from spreadsheet manipulation and retrieval-intensive tasks. For each case, the task input, environment snapshot, tool interface, and execution budget are held fixed across interventions. The excerpts below preserve the operational content of the agent traces while omitting routine tool observations.

E.1 INVOKED BUT BEHAVIORALLY INERT SKILL

The first case concerns a spreadsheet task in which the agent must repair inconsistent date formatting in a transaction worksheet without changing formulas, row order, or non-date cells. The candidate skill was generated from an earlier successful trajectory and appeared directly relevant because its trigger mentioned "mixed date formats" and "spreadsheet normalization."

The candidate contained the following procedure:

```
skill: normalize_mixed_dates

trigger:
  Use when a worksheet contains dates represented in multiple formats.

procedure:
  1. Inspect the used range and identify columns whose headers indicate dates.
  2. Infer the dominant date representation from visible values.
  3. Convert parseable values into a consistent spreadsheet date type.
  4. Preserve formulas, blank cells, and values that cannot be parsed safely.
  5. Re-open the edited range and verify that the displayed format is consistent.
```

In the candidate condition, the skill was retrieved and explicitly invoked. The agent then inspected the worksheet schema and discovered that the target date column had already been normalized by the workbook import layer. The visible inconsistency came from cell-level display formats rather than heterogeneous underlying values. Instead of following the skill's conversion procedure, the agent independently copied the number format from a neighboring valid cell and applied it to the affected range.

```
[retrieval]
Selected skill: normalize_mixed_dates

[agent reasoning]
The values in "Posting Date" are already stored as spreadsheet dates.
Re-parsing them would be unnecessary and could alter locale-sensitive entries.
The inconsistency is only in the cell display format. I will copy the format
from an adjacent valid cell and leave the stored values unchanged.

[tool action]
copy_format(
    source_cell="Posting Date!reference_cell",
    target_range="Posting Date!affected_range",
    include_value=False,
```

```
    include_formula=False
)

[verification]
The target cells now use the same date display format as the surrounding
records. Formulas and stored values are unchanged.
```

The invocation-blocked condition exposed the same skill name, description, and trigger, but withheld the body when the agent attempted to open it. The resulting trace was operationally equivalent:

```
[retrieval]
Selected skill: normalize_mixed_dates

[skill interface]
The skill body is unavailable in this intervention.

[agent reasoning]
I can determine whether this is a value problem or a formatting problem by
inspecting the cell types. The values are already dates, so only the display
format needs to be copied from a valid neighboring cell.

[tool action]
copy_format(
    source_cell="Posting Date!reference_cell",
    target_range="Posting Date!affected_range",
    include_value=False,
    include_formula=False
)
```

Both executions invoked the candidate through the retrieval interface, but access to its operational content did not change the selected tool, edited range, action arguments, or verification behavior. This is therefore an *inert invocation*: the trace satisfies an invocation-based audit, yet the candidate does not explain the successful edit. An outcome-based selector may retain the skill because it co-occurs with a correct workbook, and a fresh-agent audit may also accept it because the skill is visibly called. CEC instead compares the candidate with the invocation-blocked intervention and finds no attributable behavioral path from the skill body to the terminal output. The candidate is consequently not credited for a repair produced by the base agent's spreadsheet reasoning.

This case also clarifies why retrieval metadata must remain available in the blocked condition. Removing the skill entry entirely could change the agent's initial interpretation of the task. By preserving the trigger and blocking only the operational body, CEC asks the narrower question of whether the retrieved content, rather than its mere availability, changes execution.

E.2 Localizing a Harmful Recovery Module

The second case demonstrates hierarchical attribution. The task requires updating a summary worksheet after appending records to a source worksheet. The parent skill correctly instructs the agent to identify the source range, extend dependent formulas, refresh the summary, and verify that references remain valid. A descendant candidate adds a useful pre-edit inspection module and a new recovery module intended to handle refresh failures.

The relevant candidate modules are:

```
skill: refresh_summary_after_append

trigger:
  Use after records are appended to a workbook that contains dependent
  formulas, summaries, or pivot-like reports.
```

```
pre_edit_inspection:
  - Identify the source worksheet and the appended region.
  - Record formulas and named ranges that refer to the source region.
  - Detect whether the summary depends on formulas, cached values, or a
    refreshable report object.

procedure:
  - Extend the source range without overwriting existing formulas.
  - Propagate formulas using relative references from the preceding valid row.
  - Refresh the dependent summary object.
  - Recalculate the workbook and inspect reference errors.

recovery:
  If the summary does not visibly change after refresh:
  - clear the summary output region;
  - rebuild the output from the currently visible source records;
  - replace unresolved fields with blank cells;
  - save the workbook before running verification again.
```

The candidate condition initially follows a sound trajectory. It identifies the appended region, preserves the worksheet structure, and extends the relevant formulas. The tool then reports that the summary refresh request was accepted, but the summary view remains unchanged because the workbook uses a delayed cache update. The candidate's recovery module interprets the unchanged display as a failed refresh and clears the existing summary output.

```
[agent reasoning]
The source records and dependent formulas have been updated correctly.
The summary view still shows its previous cached content. According to the
recovery procedure, I should rebuild the visible summary.

[tool action]
clear_range(
    sheet="Regional Summary",
    range="summary_output_region",
    preserve_format=True
)

[tool action]
aggregate_visible_rows(
    source_sheet="Transactions",
    group_by=["Region", "Category"],
    output_sheet="Regional Summary",
    output_anchor="summary_anchor"
)

[tool observation]
Some source categories are represented through workbook relationships that
are not exposed as visible worksheet columns.

[agent reasoning]
The unavailable relationship fields cannot be reconstructed from visible
rows. I will leave those fields blank as specified by the recovery module.

[verification]
The summary contains no reference errors, but several relationship-derived
fields are blank.
```

The terminal workbook is superficially well formed: formulas do not display errors, formatting is preserved, and the file opens normally. However, the manually rebuilt summary omits fields computed through workbook relationships that are inaccessible through the visible source worksheet. The

harmful behavior is therefore not a malformed tool call but an over-aggressive recovery policy that destroys valid cached output before establishing that the refresh actually failed.

Whole-skill removal does not isolate this defect. Without the candidate, the agent also loses the improved pre-edit inspection module and may extend the wrong source boundary. Likewise, replacing the complete candidate with its parent can make the descendant appear mixed or unstable because the revision contains both a beneficial inspection change and a harmful recovery change.

CEC addresses this ambiguity by constructing a module-level intervention in which the candidate is left intact except that its recovery module is replaced by the parent's conservative recovery logic:

```
recovery:
  If the summary does not visibly change after refresh:
  - do not clear or overwrite the current summary;
  - recalculate the workbook;
  - re-open the summary object and inspect its refresh state;
  - if the state remains unresolved, preserve the existing output and report
    that refresh completion could not be verified.
```

Under this intervention, the execution proceeds as follows:

```
[agent reasoning]
The refresh request was accepted, but the visible summary has not updated.
An unchanged display is insufficient evidence that the refresh failed.
I will preserve the current output, recalculate, and inspect the refresh state.

[tool action]
recalculate_workbook()

[tool action]
inspect_summary_state(
    sheet="Regional Summary",
    preserve_cached_output=True
)

[tool observation]
The summary object reports a completed refresh after recalculation.

[verification]
The appended records are represented in the refreshed summary. Existing
relationship-derived fields remain populated, and the source formulas are
preserved.
```

The contrast localizes the failure to the recovery module. The candidate's trigger, pre-edit inspection, and primary procedure still guide useful actions; replacing only the recovery logic prevents the destructive branch while retaining those improvements. CEC therefore supports a repair decision that cannot be expressed by binary whole-skill retention: preserve the descendant's inspection and procedure modules, but revert its recovery module to the parent's conservative behavior.

This case further illustrates why module credits need not be additive. The inspection module increases the agent's awareness of workbook dependencies, yet the candidate recovery module fails to use that information when a delayed refresh occurs. The value of the inspection module becomes visible only when paired with recovery logic that preserves dependency-derived output. The relevant unit of analysis is thus the intervention on a structured artifact, not an assumption that independent textual sections contribute separably.

E.3 Spurious Success from Instance Difficulty

The final case concerns a retrieval-intensive question requiring the agent to identify a policy condition from a collection of technical documents and provide a source-grounded answer. A candidate skill

was evolved from trajectories in which the answer appeared in a prominently titled document. Its trigger and procedure were:

```
skill: direct_title_first_retrieval

trigger:
  Use for policy questions whose central concept appears explicitly in a
  document title or section heading.

procedure:
  1. Search for the principal noun phrase from the question.
  2. Open the highest-ranked document whose title contains that phrase.
  3. Extract the first passage that directly answers the question.
  4. Stop retrieval once a semantically matching passage is found.
```

On an easy instance, the corpus contains a document with a title closely matching the user's wording. The candidate retrieves that document immediately and produces a correct, grounded answer:

```
[user query]
Under what condition may an archived request be reopened?

[agent with candidate]
The request may be reopened when the responsible reviewer records a valid
reconsideration reason and the archive state has not been locked by the
retention process.

[evidence selected]
"Archived requests may be reopened upon a recorded reconsideration reason,
provided that retention locking has not occurred."
```

This episode makes the candidate look useful, but the same task is also straightforward without it. In the no-skill and parent conditions, the base retriever uses the distinctive phrase "retention locking," opens the same document, and returns the same supporting passage. The successful outcome therefore does not establish an incremental contribution.

The weakness becomes clearer on a semantically equivalent instance whose answer is located in an exception subsection of a broader document. A misleading document title contains the query's main noun phrase but describes the default rule rather than the exception. The candidate stops after finding a superficially matching passage:

```
[user query]
Can a request be reopened after the archive has entered a protected state?

[retrieval with candidate]
Query: "request reopened archive protected state"
Opened: "Request Reopening Policy"
Selected passage:
"Archived requests may be reopened when a reconsideration reason is recorded."

[agent answer]
Yes. A request can be reopened after archival if a reconsideration reason is
recorded.
```

The answer omits the operative exception. The matched parent condition continues retrieval because the first passage does not address the phrase "protected state":

```
[retrieval with parent]
The first passage states the general reopening rule but does not resolve the
```

```
protected-state condition. Continue searching for exceptions and retention
constraints.

Opened: "Records Lifecycle and Preservation Controls"
Selected passage:
"Once an archive enters the protected retention state, reopening is
prohibited even when a reconsideration reason has been recorded."

[agent answer]
No. The protected retention state overrides the general reopening rule, so a
recorded reconsideration reason is not sufficient after that state begins.
```

An episode-reward selector can favor the candidate if it happens to be evaluated primarily on title-aligned instances. The resulting association is spurious: easy instances both activate the skill and permit success, while the skill's early-stopping rule is not itself responsible for the answer. CEC compares the candidate and parent on the same query, corpus snapshot, retrieval interface, and tool budget. This matched contrast prevents document prominence and instance difficulty from serving as implicit credit for the candidate.

The trace also distinguishes two forms of failure. On the easy instance, the candidate is behaviorally redundant because the parent reaches the same evidence without its procedure. On the protected-state instance, the candidate is behaviorally influential but harmful because it terminates retrieval before checking exceptions. These cases would be indistinguishable if selection observed only whether the skill was invoked or whether some candidate-associated episodes succeeded.

A module-level intervention provides a direct repair. CEC retains the title-first retrieval heuristic but replaces the stopping rule with a coverage check:

```
stopping_rule:
  Do not stop after lexical or semantic agreement alone.
  Before answering, identify all conditions, exceptions, and temporal
  qualifiers in the question. Continue retrieval until the selected evidence
  addresses each of them or until the search space is exhausted.
```

With this replacement, the skill still benefits from a precise initial query when titles are informative, but it no longer treats the first matching passage as sufficient evidence. The repaired candidate explicitly checks whether retrieved text resolves the protected-state qualifier and continues to the exception subsection when it does not.

Across these cases, the decisive evidence comes from differences between matched executions rather than from successful outputs in isolation. Invocation blocking exposes skills that are called but operationally unused, parent replacement separates a descendant's incremental contribution from base or inherited behavior, and module replacement identifies the specific artifact component responsible for a beneficial or harmful branch. The resulting audit record supports an actionable evolutionary decision: reject behaviorally inert candidates, revise candidates whose apparent success is induced by easy instances, and retain useful revisions while reverting localized harmful modules.