CoEvolve: Traceable Reflective Memory Evolution for Prompt Optimization

Anonymous authors
Paper under double-blind review

Abstract

Reflective prompt optimization turns model failures into reusable natural-language guidance, but free-form memory is difficult to attribute, repair, and control as it grows. We introduce CoEvolve, a prompt optimizer that represents each candidate as a traceable set of reflection identifiers and evolves this memory under an explicit budget. The system couples candidate lineage with a persistent reflection store and combines automatic failure-to-policy Distill, structured memory mutations, shared evaluation, and parent-relative memory-growth-aware selection. In five-run, equal-budget locked-test comparisons spanning financial tagging and arithmetic, multi-hop question answering, and differential diagnosis, CoEvolve-Distill leads on FINER, HotpotQA, and DDXPlus. An independent-split protocol reduces the selection optimism gap from 0.023 to 0.007–0.011. Removing persistent state causes the largest ablation loss and variability, whereas score-only selection preserves quality but uses substantially more policy tokens. In controlled recovery episodes, full provenance improves harmful-memory identification and recovery while retaining at least 0.978 of task score. These results support treating reflective memory as an explicit, auditable optimization object rather than unstructured prompt text.

1 Introduction

Prompt optimization adapts language-model systems whose weights are unavailable or costly to update. Existing methods generate instructions, follow natural-language gradients, compile LM programs, or evolve prompts from execution traces (Zhou et al., 2023; Pryzant et al., 2023; Yuksekgonul et al., 2024; Khattab et al., 2024; Agrawal et al., 2026). Reflective search uses failed executions to derive reusable corrections, making language feedback an actionable source of adaptation.

This feedback loop creates a state-management problem. When the prompt serves simultaneously as memory, search state, and deployed policy, appending every lesson introduces redundant or conflicting rules and expands the solver context. Rewriting the prompt controls length but severs the link between a surviving instruction and its source failure. The final text then reveals neither which evidence survived nor how a candidate differs from its parent. Evaluation introduces another ambiguity: when sibling updates see different examples, their score difference mixes mutation quality with minibatch difficulty.

CoEvolve makes reflective memory an explicit optimization object. It separates failure evidence, search state, and solver-facing text through two persistent structures: a candidate lineage and a reflection store. The lineage records parents, operators, scores, and active reflection identifiers; the store retains source failures and subsequent evaluation events. Stable identifiers preserve the evidence boundary when reflections are distilled into a short policy. Candidate outcomes update the store, whose accumulated signals condition later proposals. This feedback defines the co-evolution.

Distill (O1) synthesizes failure evidence into a compact policy while retaining its source identifiers; Add (O2) inserts local rules; and Patch–Prune–Refresh (O3) removes weak inherited items before filling remaining gaps. Sibling proposals share a parent trace and evaluation examples, reducing repeated solver calls. A parent-relative selector balances validation improvement against net

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.

growth in active reflections under a hard memory budget. The main experiments evaluate automatic CoEvolve-Distill; additional operator diagnostics examine the wider framework.

We evaluate CoEvolve-Distill on FINER, Formula, HotpotQA (Yang et al., 2018), and DDXPlus (Tchango et al., 2022), using five independent runs and locked test sets. Under matched token budgets, its mean exceeds the strongest automatic baseline by 0.013 on FINER, 0.005 F1 on HotpotQA, and 0.013 on DDXPlus, while trailing GEPA by 0.055 on Formula. Independent evaluation reduces selection optimism; fixed-budget ablations identify persistent state as the largest quality contributor, while score-only selection expands policy-token use by more than threefold. Provenance-aware recovery improves harmful-memory identification and repair while preserving useful memory.

Our contributions are:

- We formulate reflective prompt optimization as the co-evolution of candidate lineage and persistent reflection memory, preserving provenance across prompt synthesis (Section 3).
- We introduce semantic memory mutations, matched sibling evaluation with a conditional variance characterization, and parent-relative memory-growth-aware selection (Section 3).
- We provide a four-task, five-run equal-budget evaluation, an independent-split analysis, mechanism ablations, and controlled memory-recovery experiments (Section 4).

2 Related Work

Automatic and evolutionary prompt optimization. Early methods search token or phrase edits through gradients, reinforcement learning, or local moves (Shin et al., 2020; Deng et al., 2022; Prasad et al., 2023). LLM-based methods generate instructions, condition proposals on prior scores, or plan prompt edits (Zhou et al., 2023; Yang et al., 2024; Wang et al., 2024b). ProTeGi and TextGrad use language feedback as optimization signals (Pryzant et al., 2023; Yuksekgonul et al., 2024), while DSPy compiles LM programs against task metrics (Khattab et al., 2024). Merit- and factor-based methods expose prompt qualities or components (Zhu et al., 2026; Liu et al., 2026a). EvoPrompt and PromptBreeder use evolutionary search (Guo et al., 2024; Fernando et al., 2024); GEPA combines execution-trace reflection with Pareto-guided evolution (Agrawal et al., 2026). CoEvolve changes the search state: candidate identities include persistent reflection references, rendering operators, and parentage, so similar prompt strings can retain different evidence histories.

Reflective learning and evolving context. Self-Refine revises outputs from model feedback, and CRITIC incorporates tool-grounded corrections (Madaan et al., 2023; Gou et al., 2024). Reflexion stores verbal reinforcement in episodic memory, while ExpeL extracts reusable knowledge from agent experiences (Shinn et al., 2023; Zhao et al., 2024). MetaReflection learns reusable instructions from past reflections (Gupta et al., 2024); ACE maintains a structured playbook for context adaptation (Zhang et al., 2025). Building on persistent language feedback, CoEvolve makes the optimization state explicit through stable reflection identities, candidate lineage, and memory difference sets. These records connect synthesized policies to their evidence and support parent-relative selection and targeted repair.

Persistent memory and interpretable search. MemoryBank, MemGPT, and Voyager support recall, context management, and reusable skills (Zhong et al., 2024; Packer et al., 2023; Wang et al., 2024a). VISTA separates semantic hypothesis generation from prompt rewriting and verifies hypotheses on parallel minibatches (Liu et al., 2026b); aPSF estimates prompt-component effects through interventions (Liu et al., 2026a). CoEvolve exposes a complementary structure through persistent reflection identities. Added and deleted identifiers connect failures, candidate mutations, and validation events, preserving the evidence needed to inspect compressed policies and diagnose harmful memory.

3 Method

This section develops CoEvolve from its persistent state through candidate construction, matched evaluation, selection, and the complete optimization loop.

3.1 System Overview and Problem Setup

We begin by separating the task interface from the prompt component that the optimizer may change. Let $V = \{x_j\}_{j=1}^{n}$ be a validation set. A solver M receives a fixed wrapper P_{fixed} and a dynamic prompt produced by candidate C. For a bounded sample score $s_j(C) \in [0, 1]$, the validation objective is

$$S(C) = \frac{1}{|V|} \sum_{j=1}^{|V|} s_j(C). \tag{1}$$

The fixed wrapper defines the task interface and output contract. The dynamic component contains the policy being optimized. Separating them prevents a mutation from silently changing the evaluator-facing interface.

Unlike reflective methods that search directly over prompt text (Agrawal et al., 2026), CoEvolve separates failure *evidence*, the *search state* used to compare candidates, and the *rendered prompt* given to the solver. This keeps compressed memory traceable, compares mutations on common evidence, and controls growth without rewarding deletion by itself.

3.2 Two Co-Evolving States

With the objective fixed, we next specify the state that carries information across iterations. At iteration t, CoEvolve maintains a candidate registry $\mathcal{C}_t$ and a reflection store $\mathcal{R}_t$. The registry records candidate parentage, operators, scores, and active identifiers; the store records source failures, rule text, and accumulated evaluation signals. Parent failures add evidence to the store, the updated store supplies child proposals, and child outcomes update both structures. This feedback makes stored evidence part of subsequent search decisions. Appendix A.1 gives the explicit updates.

3.3 Traceable Reflection State

The reflection store is the evidence-bearing half of this state and must remain interpretable as search proceeds. CoEvolve represents it as a persistent set $\mathcal{R} = \{(i, r_i)\}$ indexed by stable identifiers. Each reflection stores immutable source metadata, normalized rule text, and search signals such as usage, minibatch, validation, and Pareto events. A later candidate may omit, patch, or summarize a rule but never overwrites its origin. Optional helpful or harmful LLM attribution is disabled in the main Distill runs; the remaining signals index the search history.

On top of this store, a candidate records the tuple

$$C = \big(\mathrm{id}_C, \mathrm{pa}(C), R_C, o_C, A_C, D_C, P(C), q_C\big), \tag{2}$$

where $R_C \subseteq \mathrm{ID}(\mathcal{R})$ is the set of referenced reflection identifiers, o_C is the rendering operator, A_C and D_C are the identifiers added to and deleted from the parent state, and q_C holds the available aggregate and per-example scores. The explicit difference sets make a transition inspectable without diffing two long prompts, and they preserve lineage when two candidates render similar text from different evidence. The dynamic prompt itself is generated by

$$P(C) = g_{o_C}\big(\{r_i : i \in R_C\}\big). \tag{3}$$

For O2 and O3, the renderer formats selected rules into the solver-facing policy; for O1, it synthesizes a compact policy from those rules. In every case the registry retains R_C, even when individual source rules are no longer copied verbatim into $P(C)$, and a hard budget K limits $|R_C|$. Because the registry reports the scope of the supporting evidence separately from string length, two equally short policies derived from different amounts of evidence remain distinguishable.

This bookkeeping enforces a provenance invariant: for every registered candidate, each active identifier resolves to a stored source record,

$$\forall C \in \mathcal{C}_t,\ \forall i \in R_C, \quad (i, r_i) \in \mathcal{R}_t. \tag{4}$$

The invariant is preserved by each operator; Appendix A.2 gives the argument and an archived transition illustrating the source records and lineage.

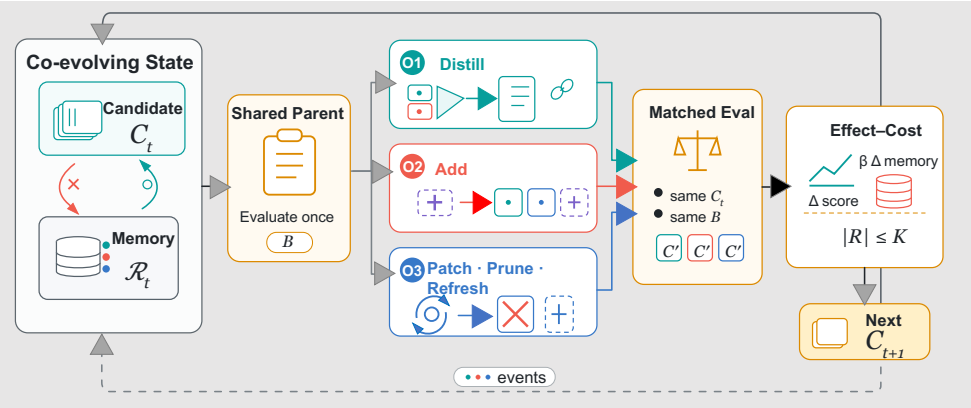


Figure 1: CoEvolve jointly updates candidate lineage and a persistent reflection store. One parent trace produces failure-derived rules for the scheduled mutations, whose outcomes update the registry and memory signals. The framework supports O1, O2, and O3; the main CoEvolve-Distill configuration schedules only O1.

3.4 From Parent Failures to Candidate Mutations

Having defined what a candidate remembers, we now describe how a parent failure produces new candidates. At each optimization step, CoEvolve selects a parent using the two-stage rule below and evaluates it on minibatch B. The reflector receives the execution trace, expected output, and task feedback, then converts failures into actionable candidate rules rather than restating the error. New rules are normalized and deduplicated against $\mathcal{R}$, and the same evidence is made available to every scheduled sibling operator. The diagnostic input is therefore fixed while the proposal family varies.

The proposal schedule is an ordered list $O_t = (o_{t,1}, \ldots, o_{t,q_t})$, with one child per entry and possible repetitions of an operator; CoEvolve-Distill schedules only Distill. Each proposal selects active reflection identifiers under K and renders their rules into a prompt. The global store supplies reusable evidence, while the selected set determines what reaches a child's renderer. Sharing a parent trace therefore does not force siblings to use the same active set or text.

Reflection inheritance and ranking. A child starts from the parent's exact active identifier set and incorporates the deduplicated failure-derived rules. Capacity-limited selection prioritizes accumulated Pareto fitness from candidate evaluations, with a small reliability adjustment favoring rules that have been evaluated more often when their scores tie. Previously untested rules receive a positive prior so that new evidence can enter selection. The selection temperature controls whether selection takes the highest-ranked rules or samples from the fitness ranking. These signals order the evidence supplied to a renderer; they do not replace the immutable source records.

The main operator, **Distill** (O1), turns selected evidence into a compact, reusable task policy. Its input contains the public task name, output contract, and failure-derived rules, but no handwritten policy. The renderer is instructed to preserve operational constraints, remove redundancy, avoid unsupported task facts, and follow the response format; outputs are capped at 180 words by default and 260 words for label-selection and calculation tasks. Distill changes textual granularity without changing the evidence boundary: the candidate retains every source identifier and records the rendering operator, so the failures behind a synthesized policy remain recoverable. The four-task main evaluation uses this fully automatic instantiation.

Two further operators exercise the same framework under different granularity assumptions. **Add** (O2) keeps the parent's active identifiers and inserts new failure-derived rules that fit under K, using the fitness ranking when capacity is tight; it preserves the wording and source granularity of precise corrections, at the cost of letting similar rules occupy multiple slots and expand the solver context. **Patch–Prune–Refresh** (O3) instead removes low-ranked inherited rules and refreshes open slots with evidence for current validation gaps. Its maintenance ranking combines cross-candidate quality and accumulated per-example validation changes with available helpfulness feedback. The optional

smoothed helpfulness estimate is

$$\theta_i = \frac{1 + h_i}{2 + h_i + m_i}, \tag{5}$$

where h_i and m_i count helpful and harmful attribution events when optional attribution is enabled. Low-ranked rules are pruned first. Because rules interact, θ_i serves as a maintenance signal, and deleted rules remain in the global store for later recall.

CoEvolve-Distill drives the four-task main comparison and its mechanism ablations. The Add+Patch diagnostics in Appendix Table S5 examine the same representation and selection machinery with a different proposal family.

3.5 Matched Sibling Evaluation

Once siblings have been constructed, they must be compared without confounding the mutation with batch difficulty. CoEvolve evaluates a parent once on minibatch B, caches its trace, and constructs every scheduled sibling from the same failures. The siblings are then scored on the same example identifiers. Parent reuse removes repeated parent trajectories, while matching identifiers removes variation caused solely by assigning an easier minibatch to one mutation.

Pairing evaluation examples is distinct from reusing generation data. For two fixed siblings C_A and C_B, define their population difference as $\Delta_{A,B} = \mathbb{E}_x[s(C_A; x) - s(C_B; x)]$. A paired estimator averages their within-example differences on one evaluation batch; an unpaired estimator subtracts averages from two separately sampled evaluation batches. In both cases below, evaluation examples are independent of candidate construction.

Proposition 1 (paired evaluation of fixed candidates). Condition on the search history and proposal randomness that produced C_A and C_B, so both candidates are fixed. Draw evaluation examples independently and identically from the target distribution, independently of this history and construction. Let $Y^A = s(C_A; x)$ and $Y^B = s(C_B; x)$, with variances σ_A^2 and σ_B^2. The paired estimator over m common examples and the unpaired estimator over two independent batches of m examples each are unbiased for $\Delta_{A,B}$. Conditional on the fixed candidates,

$$\mathrm{Var}(\widehat{\Delta}_{\mathrm{shared}}) = \frac{\sigma_A^2 + \sigma_B^2 - 2\mathrm{Cov}(Y^A, Y^B)}{m}, \tag{6}$$

$$\mathrm{Var}(\widehat{\Delta}_{\mathrm{ind}}) = \frac{\sigma_A^2 + \sigma_B^2}{m}. \tag{7}$$

Therefore, shared examples do not increase variance whenever sibling scores have nonnegative covariance across examples.

The variance identity follows by expanding paired score differences; Appendix A.3 gives the proof.

Related prompts can have positively correlated scores because they face the same example difficulty. Proposition 1 characterizes the resulting variance benefit for fixed candidates on independent evaluation data. Same-B instead uses B for construction and local scoring, so its adaptive feedback is outside the proposition's unbiasedness guarantee. Split-B separates these roles, and Table 2 measures the remaining selection optimism. Winner selection can still be optimistic even when each fixed-candidate estimate is unbiased. Separately, parent-trace reuse avoids repeated executions.

3.6 Parent-Relative Memory-Growth-Aware Selection

Matched evaluation makes sibling scores comparable; the remaining question is whether an improvement justifies additional memory. Because global prompt length alone does not reveal how a mutation changes active memory, CoEvolve measures growth relative to the parent. For child C' with parent set $\mathrm{pa}(C')$, define its validation improvement and nonnegative net growth in active reflection count as

$$\Delta(C') = S(C') - \max_{C \in \mathrm{pa}(C')} S(C), \tag{8}$$

$$a(C') = \max\left(|R_{C'}| - \max_{C \in \mathrm{pa}(C')} |R_C|, 0\right). \tag{9}$$

Candidates with $|R_{C'}| > K$ are ineligible. For a noninitial candidate, the parent-relative ranking score is

$$J(C') = \alpha\Delta(C') - \beta a(C'). \tag{10}$$

Here, $\alpha > 0$ sets the scale of behavioral improvement and $\beta \geq 0$ expresses the tolerated memory growth. The ratio β/α is the score gain required for a one-reflection increase in active memory to break even.

The growth term is $a(C')$, not the number of newly introduced identifiers $|A_{C'}|$. Replacing an equal number of inherited rules gives zero net growth, even though new evidence enters the child. Deletion likewise receives no negative growth cost and therefore no bonus by itself. Setting $\beta = 0$ yields score-only selection while the hard budget K remains active. A positive value favors smaller increases in active memory; it does not directly minimize token use, solver calls, or rendered string length. Distill may render many source reflections into a short policy, while Add may render fewer rules verbosely. We treat β as a task-dependent operating preference and report its sensitivity sweep.

Parent nomination and reselection. Parent selection has two stages. An inner per-example Pareto selector identifies candidates that attain the best observed validation score on at least one example, ranks them by the number of examples they cover, and samples a default parent from a leading shortlist. The outer selector reconsiders this nomination using J over acceptable, budget-feasible candidates in the registry; near-ties in J are resolved by validation score. Thus the parent pool includes earlier candidates as well as the latest children, and a candidate below the best global mean can remain available because it performs well on a subset of examples.

Acceptability and continuation. A child is acceptable when its validation score is within a tolerance ε of, or above, its own parent's score: $S(C') \geq \max_{C \in \mathrm{pa}(C')} S(C) - \varepsilon$. Children below this band are excluded from normal parent reselection; a fallback is used only when no acceptable candidate is available. The initial candidate C_0 remains acceptable until a child strictly improves over its parent, after which C_0 leaves the acceptable pool. This permits small parent-relative losses and globally suboptimal candidates to remain searchable without treating every recorded proposal as an accepted search state.

3.7 Optimization Procedure and Audit Trail

The state, operators, evaluator, and selector now combine into a single optimization loop. The loop separates three decisions. *Registration* preserves every proposed child's state and observed scores, including unsuccessful proposals. *Continuation* applies the two-stage parent rule to the acceptable, budget-feasible candidate pool. *Final selection* returns the compliant candidate with the highest validation score S, rather than the highest parent-relative J. The minibatch score is local search feedback, whereas S is the validation objective in Eq. 1.

Algorithm 1 preserves every proposed child's operator, parent, active identifiers, memory difference, prompt, and observed scores, including rejected proposals. Cached parent outputs are reused only while both the prompt state and example identifiers match. The audit trail connects policies to source failures and distinguishes pruning from synthesis. When reflections enter together, their outcome is recorded jointly, preserving the transition without assigning individual causal credit.

4 Experiments

We evaluate four aspects of CoEvolve: task quality, selection bias, component contributions, and the practical value of provenance.

4.1 Questions and Experimental Setup

The evaluation is organized around four questions. First, how does CoEvolve-Distill compare with established prompt optimizers under matched optimization budgets and locked test sets? Second, how much selection optimism remains when candidate selection and final evaluation reuse or split their data? Third, which mechanisms matter when the total token budget is fixed? Finally, does

Algorithm 1 CoEvolve Reflective Memory Evolution

Require: Initial prompt P_0, data, solver M, reflector L, budget K, proposal schedule $O_t = (o_{t,1}, \ldots, o_{t,q_t})$, weights α, β, tolerance ε

```
Initialize reflection store R and candidate registry with C_0
for t = 1, ..., T do
    Nominate by per-example Pareto coverage; reselect C_p using J and acceptability
    Sample shared minibatch B
    Evaluate C_p once on B; reflect on its failures
    Normalize, deduplicate, and insert new rules into R
    for k = 1, ..., q_t do
        Inherit parent IDs; select active rules under K using the scheduled operator's ranking
        Render one child C_{t,k} with operator o_{t,k}
        Evaluate C_{t,k} on B (Same-B); record its scores and lineage
    end for
    Update per-example, validation, optional attribution, and Pareto signals
    Refresh the acceptable candidate pool using the parent-relative tolerance
end for
return highest-validation compliant candidate
```

Dataset	Initial	GEPA	ACE	MetaReflection	CoEvolve-Distill (Ours)
FINER (Acc.)	0.690 ± 0.018 (0.42M)	0.719 ± 0.023 (7.87M)	0.721 ± 0.019 (8.11M)	0.724 ± 0.021 (7.94M)	0.737 ± 0.026 (8.03M)
Formula (Acc.)	0.607 ± 0.037 (0.58M)	0.820 ± 0.035 (11.72M)	0.689 ± 0.032 (12.28M)	0.757 ± 0.029 (11.95M)	0.765 ± 0.027 (12.06M)
HotpotQA (F1)	0.708 ± 0.028 (0.76M)	0.789 ± 0.022 (14.82M)	0.806 ± 0.020 (15.13M)	0.798 ± 0.020 (14.96M)	0.811 ± 0.023 (15.04M)
DDXPlus (Acc.)	0.637 ± 0.028 (0.91M)	0.688 ± 0.023 (19.76M)	0.713 ± 0.023 (20.18M)	0.705 ± 0.023 (19.89M)	0.726 ± 0.026 (20.07M)

Table 1: Equal-budget locked-test results. Scores are mean $\pm$ sample standard deviation over five runs; parentheses give mean token use per run in millions.

explicit provenance make harmful-memory diagnosis and recovery more reliable without unnecessarily deleting useful memory?

The evaluation covers FINER financial XBRL tag classification, Formula financial arithmetic, HotpotQA multi-hop question answering (Yang et al., 2018), and DDXPlus differential diagnosis (Tchango et al., 2022). Their train/validation/test protocols are listed in Appendix Table S2. We score FINER, Formula, and DDXPlus by accuracy and HotpotQA by token F1. Unless a table states otherwise, aggregates report the mean and sample standard deviation across five independent runs. Each automatic optimizer in the main comparison is rerun independently five times. Locked-test examples never generate reflections, select candidates, or trigger prompt rewrites.

Every run uses DeepSeek-v3.2 as both solver and reflection model, with deterministic solver decoding and reflection temperature 0.3. We compare the initial prompt, GEPA (Agrawal et al., 2026), ACE (Zhang et al., 2025), MetaReflection (Gupta et al., 2024), and our full CoEvolve-Distill configuration. CoEvolve-Distill retains the candidate registry, persistent reflection store, lineage, shared evaluation, and budgeted selector while using O1 Distill as its proposal family. Appendix Tables S2–S7 provide protocols, cost accounting, and additional diagnostics.

4.2 Main Results and Mechanism Analysis

We first compare task quality under closely matched optimization budgets.

The result in Table 1 is strong but not uniform. CoEvolve-Distill has the highest observed mean on FINER, HotpotQA, and DDXPlus, improving over the strongest automatic baseline by 0.013, 0.005, and 0.013, respectively. Formula is the clear exception: GEPA reaches 0.820, exceeding CoEvolve-

Dataset	Protocol	Selection score ↑	Independent-eval score ↑	Locked-test score ↑	Optimism gap ↓
HotpotQA	Same-B	0.827 ± 0.028	0.804 ± 0.025	0.811 ± 0.023	0.023
HotpotQA	Split-B	0.810 ± 0.024	0.803 ± 0.022	0.802 ± 0.024	0.007
DDXPlus	Same-B	0.744 ± 0.030	0.721 ± 0.027	0.726 ± 0.026	0.023
DDXPlus	Split-B	0.733 ± 0.026	0.722 ± 0.024	0.721 ± 0.026	0.011

Table 2: Fresh evaluation of CoEvolve-Distill. Same-B uses the same protocol as Table 1, so its locked-test values match the corresponding main results. Split-B separates the data used for candidate construction from those used to evaluate the resulting candidates.

Dataset	Variant	Test score ↑	Δ vs Full	Policy tokens
HotpotQA	Full	0.800 ± 0.022	0.000	0.117M
	Stateless	0.782 ± 0.036	-0.018	0.258M
	No-sharing	0.793 ± 0.029	-0.007	0.182M
	Score-only	0.799 ± 0.024	-0.001	0.415M
DDXPlus	Full	0.717 ± 0.024	0.000	0.178M
	Stateless	0.691 ± 0.038	-0.026	0.423M
	No-sharing	0.706 ± 0.028	-0.011	0.261M
	Score-only	0.718 ± 0.026	$+0.001$	0.647M

Table 3: Core mechanism ablation under a fixed total token budget, evaluated on independent examples. Policy tokens measure the solver-facing policy produced during optimization.

Distill by 0.055. The advantage is smallest on HotpotQA, where the mean scores of CoEvolve-Distill and ACE are close.

The optimized methods have closely matched mean token use per run: 7.87–8.11M on FINER, 11.72–12.28M on Formula, 14.82–15.13M on HotpotQA, and 19.76–20.18M on DDXPlus. The task-dependent ranking is observed within these narrow resource ranges.

Fresh evaluation and selection optimism. We next measure how much candidate selection benefits from reusing evaluation data.

Table 2 defines the optimism gap as the selection score minus the independent-evaluation score. It is 0.023 on both HotpotQA and DDXPlus under Same-B, whereas Split-B reduces it to 0.007 and 0.011. Split-B preserves the independent-evaluation scores while reducing selection optimism, and both protocols transfer to the locked test sets with similar scores.

Core mechanism ablation. We then fix the total token budget and remove one mechanism at a time.

Removing persistent state causes the largest quality loss on both tasks: 0.018 F1 on HotpotQA and 0.026 accuracy on DDXPlus. Stateless also has the largest standard deviation in each task. No-sharing incurs smaller losses of 0.007 and 0.011. Score-only selection remains numerically close to Full, including a 0.001 gain on DDXPlus, but it uses 3.5–3.6× as many policy tokens. The ablation therefore identifies persistent memory as the main quality mechanism, shared evaluation as a smaller contributor, and memory-growth-aware selection primarily as a control on policy growth.

4.3 Traceability Utility

Finally, we ask whether stored provenance helps repair a corrupted memory state.

Controlled recovery. Each episode injects harmful memory into a clean state. All variants receive the same corrupted text, recovery data, and token budget. A repair deactivates selected reflections in the current policy, which is then reevaluated. Full provenance uses stable identifiers, source meta-

Dataset	Variant	Episodes per run	Harmful-ID F1 ↑	Recovery rate ↑	Score retention ↑	Useful-memory retention ↑
HotpotQA	Full provenance	20	0.782 ± 0.081	0.840 ± 0.042	0.981 ± 0.021	0.935 ± 0.045
	Text-only	20	0.415 ± 0.112	0.560 ± 0.042	0.924 ± 0.041	0.839 ± 0.068
	LRU	20	0.150 ± 0.094	0.300 ± 0.035	0.859 ± 0.067	0.698 ± 0.102
DDXPlus	Full provenance	20	0.768 ± 0.085	0.830 ± 0.027	0.978 ± 0.020	0.922 ± 0.048
	Text-only	20	0.402 ± 0.117	0.550 ± 0.035	0.920 ± 0.043	0.825 ± 0.072
	LRU	20	0.142 ± 0.091	0.290 ± 0.022	0.855 ± 0.070	0.685 ± 0.106

Table 4: Traceability utility over five independent runs per dataset and 20 episodes per run. Recovery rates are mean $\pm$ sample standard deviation across runs. Full provenance uses source and history signals; Text-only uses rendered content; LRU uses recency.

data, lineage, and accumulated usage and validation signals. Text-only uses the rendered policy and current score, while least-recently-used removal (LRU) uses recency without quality signals. Injected corruption identifiers are hidden from every recovery method and used only by the evaluator to score harmful-memory identification.

Recovery measures. Let S_{before} and S_{after} be the scores before corruption and after repair on the same independent evaluation set. An episode succeeds when $S_{\text{after}} \geq 0.95 S_{\text{before}}$. Recovery rate averages these binary outcomes; score retention averages the continuous ratios $S_{\text{after}}/S_{\text{before}}$, which can exceed one. Both measures use this independent set throughout recovery. Appendix B.2 defines run aggregation and the supporting diagnosis measures.

Full provenance yields the highest identification and recovery scores (Table 4). Against Text-only, Harmful-ID F1 rises from 0.415 to 0.782 on HotpotQA and from 0.402 to 0.768 on DDXPlus; recovery increases by 0.280 (28 percentage points) on both tasks. Retaining at least 0.978 of the original score and 0.922 of useful memory supports targeted repair without indiscriminate deletion.

5 Discussion

Taken together, the results reveal a division of labor among CoEvolve's components. Persistent state provides the largest quality gains, shared evaluation limits repeated work, and provenance supports targeted repair; score-only selection keeps quality similar but allows policies to grow. Across the benchmark suite, Distill reaches the highest mean on three tasks, while the Formula result illustrates how the traceable state can also accommodate task-matched proposal families. The framework therefore provides a common substrate for comparing and composing proposal strategies while preserving the evidence behind each candidate.

This task dependence motivates adaptive proposal selection. More broadly, separating evidence storage from policy rendering creates a practical interface for extending the optimizer without discarding accumulated experience. The same interface can support richer memories and selectors that jointly account for quality, calls, context length, and latency. These directions retain the central advantage of CoEvolve: optimization decisions remain connected to inspectable evidence throughout the search.

6 Conclusion

CoEvolve treats candidate lineage and traceable reflection memory as a joint optimization state. Under matched budgets, CoEvolve-Distill leads on three of four tasks; independent splitting reduces selection optimism, persistent state provides the largest ablation gain, and provenance enables targeted repair. These results support reflective memory whose evidence, growth, and recovery remain inspectable. By preserving the link between an optimized policy and its supporting evidence, the framework makes subsequent updates easier to audit and revise. This provides a general foundation for prompt optimization systems that improve over time while retaining operational control.

AI USE STATEMENT

Generative AI tools were used to polish the writing, support literature search, check theoretical proofs, assist with the design and implementation of the experimental code, and support proof refinement and the analysis of synthetic experiments. The authors reviewed all AI-assisted text, code, derivations, and citations and take full responsibility for the final manuscript.

ETHICS STATEMENT

We use public benchmark datasets and study neither identities nor deployment. The authors verified all claims, results, figures, code, and references and take responsibility for the scientific and ethical integrity of the work.

REPRODUCIBILITY STATEMENT

Section 3 specifies the candidate state, reflection store, mutation operators, parent and reflection selection, shared-minibatch evaluation, and memory-growth-aware ranking, and Section 4 states the tasks, metrics, independent repetitions, locked-test protocol, fresh-evaluation design, and controlled recovery protocol. Appendix B reports evaluation protocols, supporting recovery measures, and cost accounting; Appendix C reports framework, sensitivity, and proposal-construction diagnostics; and Appendix D records retained artifacts and the scope of the evidence. All benchmarks are public, and every reported run retains its configuration, selected prompt, per-example predictions, metric log, and call/token summary.

REFERENCES

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RQm2KQTM5r.

Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric Xing, and Zhiting Hu. RLPrompt: Optimizing discrete text prompts with reinforcement learning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 3369–3391. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.emnlp-main.222. URL https://aclanthology.org/2022.emnlp-main.222/.

Chrisantha Fernando, Dylan Sunil Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 13481–13544. PMLR, 2024. URL https://proceedings.mlr.press/v235/fernando24a.html.

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. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2305.11738.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2309.08532.

Priyanshu Gupta, Shashank Kirtania, Ananya Singha, Sumit Gulwani, Arjun Radhakrishna, Gustavo Soares, and Sherry Shi. MetaReflection: Learning instructions for language agents using past reflections. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 8369–8385. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.477. URL https://aclanthology.org/2024.emnlp-main.477/.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *International Conference on Learning Representations*, pp. 54928–54958, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/f1cf02ce09757f57c3b93c0db83181e0-Paper-Conference.pdf.

Haoyue Liu, Zhichao Wang, Yongxin Guo, Haoran Shou, and Xiaoying Tang. Adaptive prompt structure factorization: A framework for self-discovering and optimizing compositional prompt programs. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11690–11714. Association for Computational Linguistics, 2026a. doi: 10.18653/v1/2026.acl-long.537. URL https://aclanthology.org/2026.acl-long.537/.

Shiyan Liu, Qifeng Xia, Qiyun Xia, Yisheng Liu, Xinyu Yu, and Rui Qu. Reflection in the dark: Exposing and escaping the black box in reflective prompt optimization. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, pp. 76–109. Association for Computational Linguistics, 2026b. doi: 10.18653/v1/2026.acl-srw.8. URL https://aclanthology.org/2026.acl-srw.8/.

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. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards llms as operating systems, 2023. URL https://arxiv.org/abs/2310.08560.

Archiki Prasad, Peter Hase, Xiang Zhou, and Mohit Bansal. GrIPS: Gradient-free, edit-based instruction search for prompting large language models. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 3845–3864. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.eacl-main.277. URL https://aclanthology.org/2023.eacl-main.277/.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7957–7968. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.494. URL https://aclanthology.org/2023.emnlp-main.494/.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. AutoPrompt: Eliciting knowledge from language models with automatically generated prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pp. 4222–4235. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.emnlp-main.346. URL https://aclanthology.org/2020.emnlp-main.346/.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

Arsene Fansi Tchango, Rishab Goel, Zhi Wen, Julien Martel, and Joumana Ghosn. DDXPlus: A new dataset for automatic medical diagnosis, 2022. URL https://arxiv.org/abs/2205.09148.

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. *Transactions on Machine Learning Research*, 2024a. URL https://openreview.net/forum?id=ehfRiF0R3a.

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P. Xing, and Zhiting Hu. PromptAgent: Strategic planning with language models enables expert-level prompt optimization. In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=22pyNMuIoa.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2309.03409.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380. Association for Computational Linguistics, 2018. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

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

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models, 2025. URL https://arxiv.org/abs/2510.04618.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936. URL https://ojs.aaai.org/index.php/AAAI/article/view/29936.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19724–19731, 2024. doi: 10.1609/aaai.v38i17.29946. URL https://ojs.aaai.org/index.php/AAAI/article/view/29946.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=92gvk82DE-.

Zixiao Zhu, Hanzhang Zhou, Zijian Feng, Tianjiao Li, Chua Jia Jim Deryl, Lee Onn Mak, Gee Wah Ng, and Kezhi Mao. Rethinking prompt optimizers: From prompt merits to optimization. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 864–892. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.eacl-long.38. URL https://aclanthology.org/2026.eacl-long.38/.

A Additional Method Details

This section expands the formal and traceability details that support the main method. It makes the two-state update explicit, records the provenance invariant, and completes the paired-evaluation argument.

A.1 Explicit State Updates

We first write the feedback loop from Section 3 as explicit state updates. At iteration t, CoEvolve maintains a candidate registry $\mathcal{C}_t$ and a reflection store $\mathcal{R}_t$:

$$\mathcal{C}_t = \{\text{candidate, parent, operator, score, active IDs}\}, \tag{11}$$

$$\mathcal{R}_t = \{\text{reflection, source, usage, validation, Pareto events}\}. \tag{12}$$

The two states form a feedback loop. A selected candidate produces failures F_t, which update the reflection store; the updated store proposes and renders children; and child evaluations Q_{t+1} are written back to both the lineage and the reflection records:

$$\widetilde{\mathcal{R}}_{t+1} = U_{\text{evidence}}(\mathcal{R}_t, F_t), \tag{13}$$

$$\widetilde{\mathcal{C}}_{t+1} = U_{\text{proposal}}(\mathcal{C}_t, \widetilde{\mathcal{R}}_{t+1}), \tag{14}$$

$$Q_{t+1} = \text{Eval}(\widetilde{\mathcal{C}}_{t+1}), \tag{15}$$

$$\mathcal{C}_{t+1} = U_{\text{lineage}}(\widetilde{\mathcal{C}}_{t+1}, Q_{t+1}), \tag{16}$$

$$\mathcal{R}_{t+1} = U_{\text{feedback}}(\widetilde{\mathcal{R}}_{t+1}, Q_{t+1}). \tag{17}$$

A static log would record search without affecting it. In CoEvolve, reflection identities and their accumulated signals affect later construction and selection, while candidate outcomes update the memory state. This bidirectional dependence defines the co-evolution.

A.2 Provenance Invariant and Recorded Transition

The state updates are useful only if active identifiers remain connected to their evidence. The invariant in Eq. 4 holds initially because R_{C_0} is empty. Add and Patch–Prune–Refresh insert only identifiers already registered in $\mathcal{R}$, while Distill rewrites the rendered text but retains its source identifier set, so candidate registration preserves the property by induction. Since every child also records its parent and its added and deleted identifiers, one can reconstruct why a candidate was proposed and which evidence survived selection.

Archived field	Example from one HotpotQA transition
Failure sources	seven archived training examples
Reflection records	five reflection records retaining links to their sources
Candidate lineage	child C_1, parent C_0, operator O1 Distill
Memory difference	new identifiers added; none deleted
Rendered policy	identify question type, ground in context, and extract the exact answer
Observed validation	the child improves over its parent

Table S1: Example of a real machine-readable transition from an archived HotpotQA run. The child records its parent, operator, and added identifiers alongside the source failures behind its rendered policy.

A.3 Proof of Proposition 1

We close the method supplement by justifying the variance comparison used for matched sibling evaluation. All expectations and variances below condition on the fixed candidates and their construction history, as stated in Proposition 1.

Proof. Independence from candidate construction makes the expected score difference on each evaluation example equal to $\Delta_{A,B}$. Linearity of expectation gives unbiasedness. Independence across

evaluation examples and expansion of $\mathrm{Var}(Y^A - Y^B)$ give the first expression; independent batches remove the covariance term in the second. □

B Experimental Protocols and Detailed Results

This section supplies the evaluation details needed to interpret the main tables. We first state the common protocol, then define recovery aggregation and report the supporting controls and resource measurements.

B.1 Final Evaluation Protocol

We begin with the dataset splits and repetition counts shared by the main comparison.

Dataset	Task	Train used	Val	Test	Optimization runs
FINER	XBRL tag classification	50	10	441	5
Formula	Financial arithmetic	500	300	200	5
HotpotQA	Multi-hop QA	8500	500	1000	5
DDXPlus	Differential diagnosis	241 stratified	1000	1000	5

Table S2: Final evaluation protocols. "Train used" is the number of examples available to the optimizer, not the original dataset size. Each automatic optimizer in the four-task main comparison is evaluated over five independent optimization runs.

The held-out split is never used to choose a prompt. All experiments use DeepSeek-v3.2 as both solver and reflection model with fixed decoding settings across methods. Distill receives failure-derived reflections, the public task name, and the output contract, but no handwritten task policy.

Run aggregation. Table 1 reports the mean and sample standard deviation across $n = 5$ independent runs, with the standard deviation computed using denominator $n - 1$.

B.2 Supporting Recovery Measures

The main text gives the recovery intuition; here we define how the reported quantities are aggregated. Each dataset is evaluated over five independent runs, each containing 20 recovery episodes. For run j, let u_j be the number of episodes reaching the recovery threshold. Its recovery rate is $r_j = u_j/20$. Table 4 reports $\bar{r} = \frac{1}{5}\sum_{j=1}^{5} r_j$ and the sample standard deviation $s_r = \sqrt{\frac{1}{4}\sum_{j=1}^{5}(r_j - \bar{r})^2}$. Score retention uses continuous after/before score ratios; reaching the binary recovery threshold and retaining a high score are distinct outcomes.

Harmful-ID precision and recall compare the reflections identified as harmful with the hidden injected-corruption identifiers; Harmful-ID F1 is their harmonic mean. These identifiers serve only as evaluator labels. Useful-memory retention measures how much uncorrupted memory remains active after repair. The recovery procedure directly edits the current policy text, and the resulting policy supplies the post-repair evaluation score.

B.3 Task-Aware Policy Capacity Control

The first control asks whether a manually specified task policy reveals proposal capacity beyond the automatic methods.

Dataset	Initial	ACE	Task-aware policy	CoEvolve-Distill
FINER	0.690	0.721	0.729	**0.737**
Formula	0.607	0.689	0.748	**0.765**
HotpotQA	0.708	0.806	0.809	**0.811**
DDXPlus	0.637	0.713	**0.761**	0.726

Table S3: Comparison with a predefined task-aware policy. This manually specified oracle measures proposal-capacity headroom; it is not an automatically discovered method and is excluded from automatic-method rankings.

B.4 HotpotQA Cost and Output Length

We next complement the quality comparison with the available HotpotQA resource and output-length traces.

Method	Cost runs	Val F1	Test F1	Test EM	Opt. calls	Memory calls	Output tokens
Initial	1	—	0.708	0.642	0	n/r	n/r
Pure GEPA	1	0.799	0.789	0.640	1094	n/r	n/r
ACE ckpt-15	1	—	0.806	0.656	1186	41.1	—
CoEvolve-Distill	5	0.827 ± 0.028	0.811 ± 0.023	0.659 ± 0.018	1098 ± 12.5	2.7 ± 1.1	110.8 ± 14.7

Table S4: Supplementary HotpotQA cost and output-length statistics. The run counts in this table describe this resource diagnostic, separately from the five-run main comparison in Table 1. Complete token traces are retained for two runs. Under the same tokenizer, ACE output is $5.81\times$ as long as CoEvolve-Distill output.

C Additional Mechanism and Operator Analyses

The main ablation isolates the Distill pipeline. The analyses below probe related framework choices with the fixed proposal families and run counts stated in each table.

C.1 Core Mechanism Supplement

We first check whether sharing and the memory-growth preference show similar behavior under the Add+Patch proposal pool.

Dataset	Full (shared, $\beta = 0.002$) Score/Calls/Refs	$\beta = 0$ (shared) Score/Calls/Refs	No-sharing Score/Calls
FINER	0.687 ± 0.052 / 332 / 29.1	0.701 ± 0.041 / 337 / 30.7	0.685 ± 0.021 / 517
HotpotQA	0.774 ± 0.018 / 495 / 20.9	0.774 ± 0.024 / 408 / 20.2	0.779 ± 0.005 / 692

Table S5: Three-run framework diagnostic using the fixed Add+Patch proposal pool. Scores are reported as mean $\pm$ sample standard deviation; the remaining entries give solver calls and reflection counts. Shared evaluation reduces calls on both tasks while the mean scores remain close.

C.2 Memory-Growth Preference Sensitivity

Because β changes which candidates continue the search, we report the corresponding single-run sensitivity rather than assuming a monotonic response.

β	FINER Val/Test/Calls/Refs/Chars	HotpotQA Val/Test/Calls/Refs/Chars
0	0.752 / 0.739 / 356 / 24 / 1879	0.791 / 0.799 / 428 / 16 / 2307
0.00025	0.768 / 0.726 / 276 / 26 / 4388	0.836 / 0.792 / 559 / 16 / 1791
0.0005	0.738 / 0.678 / 356 / 24 / 2477	0.793 / 0.745 / 361 / 13 / 892
0.001	0.760 / 0.709 / 278 / 31 / 3231	0.818 / 0.757 / 460 / 13 / 914
0.002	0.763 / 0.716 / 356 / 19 / 1716	0.820 / 0.767 / 559 / 15 / 2319

Table S6: Single-run sensitivity to the memory-growth preference. $\beta = 0$ removes the soft growth penalty while retaining the hard memory constraint.

C.3 Formula Proposal-Family Diagnostic

Finally, we examine the task on which native GEPA outperforms CoEvolve-Distill and compare how its proposals are constructed.

Proposal construction	Correct/300	Val Acc. (3 d.p.)	Calls	Reflections
Matched GEPA native	253/300	**0.843**	1236	0
Native + Distill + Distill	250/300	0.833	1298	29
Distill + Distill + Distill	245/300	0.817	1504	59
Native + Native + Distill	246/300	0.820	1298	31

Table S7: Single-run Formula proposal-construction diagnostic on the full 300-example validation set. Accuracy is the fraction of correct answers, rounded to three decimal places; the integer counts retain the exact values. Native GEPA proposals perform best, consistent with Table 1.

Native denotes a proposal generated by GEPA's native proposal mechanism. Native + Distill + Distill starts from one native proposal and applies two successive Distill rewrites. Distill + Distill + Distill first generates a proposal with Distill and then rewrites it twice. Native + Native + Distill generates two native proposals separately and synthesizes them into one proposal with Distill. Thus, the first two combinations use successive rewrites, while the last combines two proposals before producing a single synthesized proposal. This diagnostic compares proposal constructions on the full validation set and reports validation results only.

D Reproducibility and Scope

We conclude the appendix by summarizing the retained artifacts and the boundaries of the present evidence.

D.1 Reproducibility Notes

The reported results are backed by retained run records. Run configurations and metric summaries are stored in the experiment archive, while the anonymous code supplement provides the core optimization implementation and task runners.

We report realized calls and tokens rather than configured ceilings as measured cost.

D.2 Limitations and Scope

These artifacts make the reported setting inspectable, but they do not remove its empirical limits. The evidence spans one solver/reflection model family. The main comparison uses five independent runs for every automatic optimizer; additional framework and operator diagnostics have the run counts stated in their tables. The Formula diagnostic shows that proposal-family fit is task dependent: Distill is not uniformly superior to native prompt evolution. The traceability study evaluates five independent runs per task, with 20 recovery episodes per run, and should be extended to more tasks, corruption types, and intervention strengths.

Persistent identifiers preserve provenance but do not by themselves establish causal credit because reflections and prompt components interact. Reflection count, policy tokens, and total tokens are complementary resource proxies; none directly captures latency or provider-specific monetary cost. Future work should jointly select proposal families and optimize quality, context length, calls, latency, and recovery robustness.