EVOSAMPLER: UTILITY-GUIDED EXPERIENCE SAMPLING FOR PROMPT EVOLUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Prompt evolution allows LLM agents to accumulate experience across rounds of revision and evaluation without updating model weights. Under a limited budget, an optimizer must choose which training instances to expose to the revision process, and this choice determines what experience can shape the next prompt. We argue for a relational view of training-instance value, contingent on whether the revisions it tends to induce address the failures the current prompt still exhibits. This view motivates an objective that scores candidates by their expected contribution to the validation gain of the post-revision prompt. We introduce **EvoSampler**, a utility-guided experience sampler that resolves this problem by learning from the history of past accept/reject outcomes and re-evaluating that history against the errors the current prompt still exhibits. EvoSampler estimates each training instance's *reparability* (its reliability in producing revisions that survive evaluation) and the validation failures its accepted revisions tend to resolve. These effects are captured by two complementary components: a *Repairability Vector*, which estimates each training instance's ability to induce successful revisions, and a *Transfer Matrix*, which characterizes the training-validation attribution between training instances and validation failures. An upper-confidence-bound rule further supplies exploration when evidence is sparse. Across five benchmarks spanning mathematical reasoning, interactive decision-making, and instruction following, EvoSampler achieves the highest average held-out score among the compared optimizers under both model configurations. With DeepSeek V4 Flash as the target LLM, it improves the HMMT and ALFWorld scores by 28.33 and 42 percentage points, respectively, over the unoptimized prompt. The implementation is available at `https://anonymous.4open.science/r/EvoSampler`.

1 INTRODUCTION

Learning from experience matters as LLM agents face tasks their initial instructions cannot foresee (Chen et al., 2023; Shinn et al., 2023; Amballa et al., 2026). Prompt evolution offers a way to carry that experience into future behavior without changing model weights (Zhou et al., 2023b; Yang et al., 2024; Fernando et al., 2024). It proceeds through repeated rounds of experience, revision, and evaluation. In each round, an optimizer selects training instances, turns the resulting trajectories and feedback into revised instructions, and evaluates the candidates to determine how the prompt should be updated (Pryzant et al., 2023; Khattab et al., 2024; Agrawal et al., 2026). Since only a small number of instances can be explored under a limited budget, the selection determines which experiences can shape the next revision.

An observed failure, however, need not translate into progress. As illustrated in Figure 1, feedback from some training instances seldom yields a revision worth retaining. Even when a revision is retained, its gains may extend to only a narrow range of problems. Nor does past success guarantee future value — its earlier revisions may bear little on the errors the current prompt still makes.

The value of a training instance depends not only on the revisions it tends to induce, but also on how those revisions relate to the failures that remain. This view calls for an objective that values a training instance by its downstream effect on prompt evolution. This creates a prediction problem — the outcome needed to assess a task is observed only after the resulting revision has been generated and evaluated, while the task must be selected beforehand.

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.

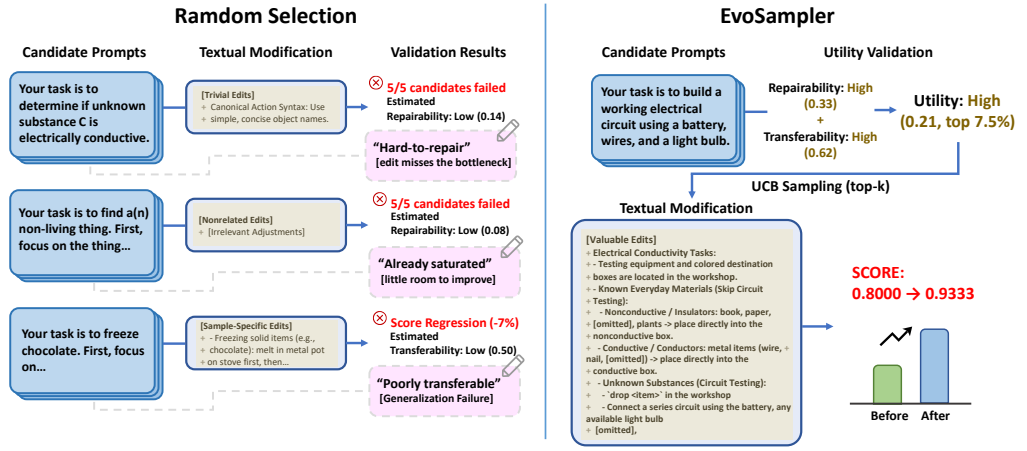


Figure 1: EvoSampler scores training samples by repairability (training improvement) and transferability (validation generalization), selecting high-utility samples to improve prompt performance.

We address this problem by estimating a training instance's expected downstream effect from earlier updates, considering both how reliably it can improve training instances and how its effects transfer to the validation failures of the current prompt. The resulting utility, reflecting both training-set improvement and validation transfer, evolves with the prompt rather than remaining fixed throughout optimization. This reframes training data selection as online, prompt-dependent data valuation for prompt evolution.

We introduce **EvoSampler**, a utility-guided experience sampler for prompt evolution. From past accept-reject outcomes, it estimates how reliably feedback from each training instance produces a revision that survives evaluation, forming a *Repairability Vector*. From the problem-level outcomes of accepted revisions, it learns how likely each training instance is to improve individual unresolved validation problems, forming a *Transfer Matrix*. The Transfer Matrix is estimated only over validation problems that remain unresolved for the current prompt. As the error set changes, the utility computed from the two components can lead to a different sampling priority from the same historical evidence. An upper-confidence-bound rule combines estimated utility with exploration when observations are sparse. EvoSampler thereby adapts the experience used to generate each revision while preserving the underlying revision generator and candidate-evaluation protocol.

We evaluate EvoSampler on MATH-500 (Lightman et al., 2024), HMMT (Dekoninck et al., 2026), ALFWorld (Shridhar et al., 2021), ScienceWorld (Wang et al., 2022), and IFBench (Pyatkin et al., 2025), covering mathematical reasoning, interactive decision-making, and instruction following. Across two executor–candidate-generator configurations and comparable optimization budgets, EvoSampler improves held-out test performance over GEPA (Agrawal et al., 2026), APEX (Wang et al., 2026), TextGrad (Yüksekgönül et al., 2024), and MIPRO (Opsahl-Ong et al., 2024). Statistical analyses support the use of update histories to anticipate the value of later training choices, while ablations isolate the contributions of acceptance estimation, training–validation attribution, conditioning on current errors, and exploration. Together, these results show that an agent's ability to evolve depends not only on how it revises its prompt, but also on which experiences it chooses to learn from.

2 Preliminary

2.1 Task Formulation: Prompt Evolution

A prompt is the interface through which task instructions and behavioral requirements are conveyed to an LLM, and different prompts can elicit markedly different behavior from the same frozen model (Kojima et al., 2022). This sensitivity makes the prompt a natural substrate for self-evolution: an LLM system can improve its behavior by revising the prompt in response to task feedback, with-

out updating its parameters (Pryzant et al., 2023; Yüksekgönül et al., 2024). Prompt evolution seeks a prompt for a fixed target LLM that maximizes its expected task performance:

$$P^* = \arg\max_P \ \mathbb{E}_{(x,m)\sim\mathcal{D}} \left[\sigma(f_\theta(x; P), m)\right]. \tag{1}$$

Here, f_θ denotes the target LLM with frozen parameters θ, (x, m) is a task instance with model input x and evaluation information m, and σ is a task-specific evaluator that maps the model output and m to a score. To study the role of experience selection, we partition the available task instances into two disjoint sets. The training set $\mathcal{D}_{\text{train}} = \{t_i\}_{i=1}^K$ provides the experience from which prompt revisions are generated, whereas the validation set $\mathcal{D}_{\text{val}} = \{v_j\}_{j=1}^N$ assesses whether those revisions generalize beyond the instances that induced them. Because both the target LLM and the evaluator are treated as black boxes, gradients with respect to the prompt are unavailable. Prompt optimizers therefore search the prompt space without gradients, typically through an iterative two-stage pipeline: prompt revision followed by candidate selection.

Prompt revision. The optimizer selects a parent prompt P from the candidate pool $\mathcal{P}$ and samples a small batch $\mathcal{B} \subset \mathcal{D}_{\text{train}}$. Running the agent with P on $\mathcal{B}$ yields trajectories, task outcomes, and evaluator feedback, collectively denoted by $\mathcal{E}(P, \mathcal{B})$. A reflection model $\mathcal{G}_\phi$ with fixed parameters ϕ then interprets this feedback and proposes a revised candidate:

$$\widetilde{P} \sim \mathcal{G}_\phi(P, \mathcal{E}(P, \mathcal{B})). \tag{2}$$

Candidate selection. The candidate is evaluated on $\mathcal{D}_{\text{val}}$ and compared with the prompts currently in $\mathcal{P}$. Writing $\mathcal{P}^+$ for the updated pool, the generic selection step is

$$\mathcal{P}^+ = \text{Select}\Big(\mathcal{P} \cup \{\widetilde{P}\}; \mathcal{D}_{\text{val}}\Big), \tag{3}$$

where $\text{Select}(\cdot; \mathcal{D}_{\text{val}})$ retains a subset of candidates according to their validation performance. The instance-level validation outcomes of the retained prompts then guide subsequent search. Thus, $\mathcal{B}$ determines which weaknesses are exposed to the reflection model, whereas $\mathcal{D}_{\text{val}}$ determines whether the resulting candidate generalizes and is retained. Our method intervenes only in the first stage by selecting $\mathcal{B}$; the candidate-generation and candidate-selection procedures remain unchanged and the resulting data-value estimates are developed in the next section.

2.2 Limitations of Uniform Batch Sampling

Although prompt evolution has shown promise as a way to improve LLM behavior without updating model parameters, its effectiveness under a fixed optimization budget depends critically on which task instances provide feedback for each revision. Existing evolutionary prompt optimizers construct revision batches by sampling training instances uniformly at random, typically without replacement within each batch (Agrawal et al., 2026). By assigning every instance the same marginal probability of influencing the next revision, this strategy ignores historical evidence about which experiences are useful. Under a fixed optimization budget, an uninformative batch can therefore consume the cost of feedback, revision, and evaluation without advancing the search.

Heterogeneous reparability. Feedback from different training instances has different prospects of producing an actionable prompt revision. An instance that the current prompt already solves may provide little diagnostic information, whereas an instance whose failure is fundamentally beyond the reach of prompt revision may provide a noisy or uninformative signal. Between these extremes are instances whose trajectories expose concrete and correctable weaknesses. The probability that an instance induces a revision that improves training-set performance varies across instances and evolves during optimization. We call this probability its *reparability*, which uniform sampling ignores by assigning all instances equal sampling probability.

Prompt-agnostic transferability. An actionable revision need not generalize beyond the training experience that induced it. Its gains may overfit to the local training distribution, while improvements on some validation instances coexist with regressions on others. Uniformly weighting training instances overlooks their effects on validation problems and cannot target the current prompt's unresolved deficiencies. The same transfer pattern may therefore be valuable for one prompt but irrelevant for another after the corresponding validation errors have been resolved. Uniform sampling lacks this prompt-conditioned, instance-level account of training-to-validation transfer.

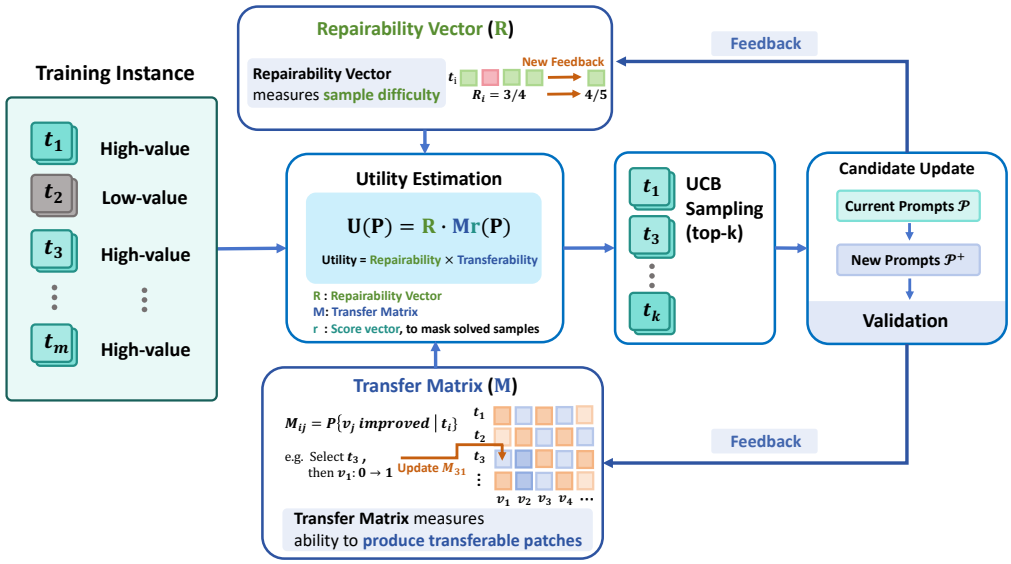


Figure 2: The overall framework of **EvoSampler**. It scores training instances with two components: (1) the *Repairability Vector*, estimating how likely a revision based on each instance is to improve it; and (2) the *Transfer Matrix*, estimating how likely the resulting revision is to improve unresolved validation instances. Their estimates are combined with a UCB exploration bonus to select the next training batch.

3 EvoSampler: Utility-Guided Experience Sampling for Prompt Evolution

In this section, we introduce **EvoSampler**, a prompt-dependent experience-sampling method for data-efficient prompt evolution, as illustrated in Figure 2. EvoSampler represents historical outcomes with two complementary components: (1) the *Repairability Vector*, whose entries estimate how reliably batches containing each training instance yield a retained revision; and (2) the *Transfer Matrix*, whose entries estimate the propensity of revisions associated with each training instance to resolve each validation failure. At each round, a UCB-based sampler combines the Repairability Vector with the Transfer Matrix evaluated on the failures that remain for the current prompt, then selects the next training batch while balancing exploitation and exploration.

3.1 Training-Instance Utility

To construct informative revision batches, we need a criterion that ranks training instances by their expected contribution to validation gain after candidate selection. We operationalize this gain as the number of validation instances that are unresolved under the current prompt but solved by a retained revision. For a prompt P, let

$$\mathbf{s}(P) = [s_1(P), \ldots, s_N(P)]^\top, \tag{4}$$

$$\mathbf{r}(P) = \mathbf{1}_N - \mathbf{s}(P) = [r_1(P), \ldots, r_N(P)]^\top, \tag{5}$$

where $\mathbf{1}_N$ is the all-ones vector, $s_j(P) \in \{0, 1\}$ indicates whether P solves validation instance v_j, and $r_j(P) = 1 - s_j(P)$ indicates that v_j remains unresolved. If $\widetilde{P}$ is generated from revision batch $\mathcal{B}$ and A indicates whether it is retained, define the utility of training instance t_i under P by

$$U_i(P) = \mathbb{E}\Big[A \left\langle \mathbf{s}(\widetilde{P}), \mathbf{r}(P) \right\rangle \Big| t_i \in \mathcal{B}\Big]. \tag{6}$$

The inner product counts the validation instances that are unresolved under P and solved by the retained candidate. Consider round q with parent prompt $P^{(q)}$, revision batch $\mathcal{B}^{(q)} \subset \mathcal{D}_{\text{train}}$, and

candidate $\widetilde{P}^{(q)}$ generated from the batch feedback. Let

$$B_{qi} = \mathbb{I}\Big[t_i \in \mathcal{B}^{(q)}\Big], \tag{7}$$

$$\mathbf{b}^{(q)} = [B_{q1}, \ldots, B_{qK}]^\top \tag{8}$$

indicate which training instances participate in the revision, and let $A^{(q)}$ indicate whether $\widetilde{P}^{(q)}$ is retained. Because the entire batch produces a single candidate, all included instances receive shared credit for the same validation progress.

Theorem 1 (Realized utility under shared credit) *For round q, let $\mathbf{u}^{(q)} = [u_{q1}, \ldots, u_{qK}]^\top$ collect the batch-attributed utilities. Then*

$$\mathbf{u}^{(q)} = A^{(q)}\mathbf{b}^{(q)} \left\langle \mathbf{s}\Big(\widetilde{P}^{(q)}\Big), \mathbf{r}\Big(P^{(q)}\Big) \right\rangle, \tag{9}$$

The inner product measures the candidate's progress on validation instances unresolved by the parent prompt, while $\mathbf{b}^{(q)}$ assigns this shared outcome to the instances that supplied the feedback. The proof is provided in Appendix B.

To estimate $U_i(P)$, we separate the probability that t_i induces a retained revision from the probability that the revision transfers to a particular validation instance. For each training instance t_i and validation instance v_j, define:

$$\rho_i = \mathbb{P}(A = 1 \mid t_i \in \mathcal{B}), \tag{10}$$

$$\mu_{ij} = \mathbb{P}\Big(s_j(\widetilde{P}) = 1 \mid A = 1,\ t_i \in \mathcal{B},\ r_j(P) = 1\Big). \tag{11}$$

Here, ρ_i is the repairability of t_i: the probability that a batch containing it yields a retained revision. The quantity μ_{ij} measures how likely that retained revision is to solve v_j when v_j is unresolved. The current prompt enters through $\mathbf{r}(P)$, which selects the relevant entries of each training instance's transfer profile. We collect the per-instance probabilities into the Repairability Vector $\boldsymbol{\rho} = [\rho_1, \ldots, \rho_K]^\top$ and the pairwise transfer propensities into the Transfer Matrix $\mathbf{M} = [\mu_{ij}]_{K\times N}$. To estimate these components from a changing optimization trajectory, we pool observations across prompt states and treat repairability and pairwise transfer propensity as stable over the optimization horizon.

Theorem 2 (Expected utility decomposition) *Under this pooled factorization, let $\mathbf{U}(P) = [U_1(P), \ldots, U_K(P)]^\top$ and $\mathbf{R} = \mathrm{diag}(\boldsymbol{\rho})$. Then the expected utility vector satisfies:*

$$\mathbf{U}(P) = \mathbf{R}\mathbf{M}\mathbf{r}(P). \tag{12}$$

The detailed derivation is provided in Appendix B. We estimate both components online from outcomes observed before round q. Their posterior means give the utility estimate for round q:

$$\widehat{\mathbf{U}}^{(q)}(P) = \widehat{\mathbf{R}}^{(q)}\,\widehat{\mathbf{M}}^{(q)}\,\mathbf{r}(P). \tag{13}$$

3.2 Repairability Vector

The i-th entry ρ_i of the Repairability Vector is the marginal probability that a batch containing t_i yields a candidate retained by selection, averaged over the prompt and batch contexts encountered during optimization. In round q, the candidate's retention indicator $A^{(q)}$ provides an observation for every training instance included in the batch. As a working model, we treat the shared-credit outcomes observed when t_i is included as conditionally independent Bernoulli observations with parameter ρ_i, and place a uniform Beta prior, $\rho_i \sim \mathrm{Beta}(1, 1)$. Let $\mathcal{H}_{q-1}$ denote the optimization history through round $q-1$. The observations in this history yield the posterior parameters:

$$\alpha_i^{(q)} = 1 + \sum_{\ell<q} B_{\ell i} A^{(\ell)}, \tag{14}$$

$$\beta_i^{(q)} = 1 + \sum_{\ell<q} B_{\ell i} \left(1 - A^{(\ell)}\right). \tag{15}$$

Under this model, the Beta posterior mean gives the repairability estimate:

$$\rho_i \mid \mathcal{H}_{q-1} \sim \mathrm{Beta}\Big(\alpha_i^{(q)}, \beta_i^{(q)}\Big), \tag{16}$$

$$\widehat{\rho}_i^{(q)} = \mathbb{E}[\rho_i \mid \mathcal{H}_{q-1}] = \frac{\alpha_i^{(q)}}{\alpha_i^{(q)} + \beta_i^{(q)}}. \tag{17}$$

The $\mathrm{Beta}(1,1)$ prior implements Laplace smoothing through one pseudo-success and one pseudo-failure, keeping estimates away from zero and one when observations are sparse. Since the candidate is generated from the batch's joint feedback, this estimate reflects shared-credit evidence associated with t_i, rather than an isolated effect of that instance.

3.3 Transfer Matrix

The Repairability Vector indicates how likely a batch containing each training instance is to yield a retained revision, but not which validation failures that revision may resolve. For each training-validation pair (t_i, v_j), we model whether a retained candidate solves v_j as a Bernoulli observation with parameter μ_{ij}, conditional on v_j being unresolved by the parent prompt. Because the candidate is generated from joint batch feedback, this outcome provides shared-credit evidence for every sampled instance. With the same $\mathrm{Beta}(1,1)$ prior and conditional-independence working model, we update μ_{ij} only when t_i is sampled, the candidate is retained, and v_j is unresolved by the parent:

$$\alpha_{ij}^{(q)} = 1 + \sum_{\ell<q} B_{\ell i} A^{(\ell)} r_j\Big(P^{(\ell)}\Big)\, s_j\Big(\widetilde{P}^{(\ell)}\Big), \tag{18}$$

$$\beta_{ij}^{(q)} = 1 + \sum_{\ell<q} B_{\ell i} A^{(\ell)} r_j\Big(P^{(\ell)}\Big)\Big[1 - s_j\Big(\widetilde{P}^{(\ell)}\Big)\Big]. \tag{19}$$

Conditioned on $\mathcal{H}_{q-1}$, μ_{ij} has a Beta posterior, whose mean we use as the corresponding Transfer Matrix entry:

$$\mu_{ij} \mid \mathcal{H}_{q-1} \sim \mathrm{Beta}\Big(\alpha_{ij}^{(q)}, \beta_{ij}^{(q)}\Big), \tag{20}$$

$$\widehat{\mu}_{ij}^{(q)} = \mathbb{E}[\mu_{ij} \mid \mathcal{H}_{q-1}] = \frac{\alpha_{ij}^{(q)}}{\alpha_{ij}^{(q)} + \beta_{ij}^{(q)}}. \tag{21}$$

Collecting these estimates gives the Transfer Matrix estimate $\widehat{\mathbf{M}}^{(q)} = [\widehat{\mu}_{ij}^{(q)}]_{K\times N}$. The current prompt affects utility through $\mathbf{r}(P)$: transfer estimates for resolved validation instances remain available but contribute nothing to the current sampling score.

3.4 UCB-Based Batch Selection

The utility estimates are less reliable for training instances with few observations, so we augment them with an exploration bonus. Let

$$n_i^{(q)} = \sum_{\ell<q} B_{\ell i}, \tag{22}$$

$$T^{(q)} = \sum_{i=1}^{K} n_i^{(q)}. \tag{23}$$

Here, $n_i^{(q)}$ counts the previous batches containing t_i, and $T^{(q)}$ is the total number of training-instance selections before round q. For $n_i^{(q)} > 0$, define the UCB-style exploration bonus:

$$b_i^{(q)} = \sqrt{\frac{2\log\big(\max\{T^{(q)}, 2\}\big)}{n_i^{(q)}}}. \tag{24}$$

For exploration weight $\lambda \geq 0$, define the selection index:

$$I_i^{(q)} = \begin{cases} +\infty, & n_i^{(q)} = 0, \\ \widehat{U}_i\big(P^{(q)}\big) + \lambda b_i^{(q)}, & n_i^{(q)} > 0. \end{cases} \tag{25}$$

The infinite index prioritizes every unobserved instance, including when $\lambda = 0$. For batch size k, we select the k instances with the largest indices:

$$\mathcal{B}^{(q)} = \text{Top-k}_{t_i \in \mathcal{D}_{\text{train}}} I_i^{(q)}. \tag{26}$$

The selected instances are then used to generate the next candidate, and the sampling counts are updated as

$$n_i^{(q+1)} = n_i^{(q)} + \mathbb{I}\left[t_i \in \mathcal{B}^{(q)}\right]. \tag{27}$$

At each round, the bonus is larger for less-sampled instances and depends logarithmically on the total number of selections. When indices are tied, random tie-breaking recovers uniform sampling among the tied instances. The parameter λ controls the trade-off between exploiting high-utility instances and exploring instances with limited history.

4 EXPERIMENTS

We first describe the experimental setup, then present the results and analyze **EvoSampler** through the following research questions:

- **RQ1:** How does **EvoSampler** perform compared with existing conventional and evolutionary baselines across tasks and model configurations?
- **RQ2:** How do the Repairability Vector, Transfer Matrix, and prompt-dependent masking contribute to **EvoSampler**'s performance?
- **RQ3:** How well do the Repairability Vector and Transfer Matrix predict revision outcomes and transfer effects?

4.1 EXPERIMENTAL SETUP

Datasets. We evaluate **EvoSampler** on five public benchmarks covering mathematical reasoning, interactive decision-making, and instruction following. MATH-500 (Lightman et al., 2024) and HMMT evaluate mathematical reasoning, ALFWorld (Shridhar et al., 2021) and ScienceWorld (Wang et al., 2022) evaluate task execution in interactive environments, and IFBench (Pyatkin et al., 2025) evaluates instruction following. Further details on the benchmarks and data splits are provided in Appendix C.1.

Baselines. We compare EvoSampler with two categories of methods for optimizing textual components: (1) evolutionary methods, including GEPA (Agrawal et al., 2026) and APEX (Wang et al., 2026); and (2) conventional optimization methods, including TextGrad (Yüksekgönül et al., 2024) and MIPRO (Opsahl-Ong et al., 2024). We describe these baselines in Appendix C.2.

Implementation Details. We consider two model configurations to evaluate the effectiveness and generalizability of our method: DeepSeek V4 Flash (DeepSeek-AI, 2026) with DeepSeek V4 Pro (DeepSeek-AI, 2026), and Gemini 3 Flash (DeepMind, 2026) with Claude Opus 4.6 (Anthropic, 2025). In each configuration, DeepSeek V4 Flash and Gemini 3 Flash serve as the base models to answer questions from the datasets, while DeepSeek V4 Pro and Claude Opus 4.6 analyze the resulting trajectories and environmental feedback to generate new prompt candidates. The optimization protocol and implementation details are given in Appendices C.3 and C.4, respectively.

4.2 OVERALL PERFORMANCE (RQ1)

Table 1 reports the results of **EvoSampler** and the baselines under the DeepSeek configuration. Our main observations are as follows: **(1) Best average performance.** Averaged equally across the five benchmarks, **EvoSampler** scores 81.67, exceeding GEPA, the strongest baseline on average, by 5.52 percentage points. It also achieves the best or joint-best score on every benchmark. **(2) Consistent gains over evolutionary prompt optimizers. EvoSampler** outperforms GEPA and APEX on all five benchmarks under this configuration. These results support the benefit of selecting the training experience used for prompt revision. We also evaluate the Gemini configuration; its results and detailed analysis are provided in Appendix D.1.

Table 1: Benchmark results for different optimizers using DeepSeek V4 Flash as the target LLM and DeepSeek V4 Pro as the reflection model. Bold indicates the best score.

	MATH-500	HMMT	ScienceWorld	ALFWorld	IFBench	Average
base	80	48.33	41	24	67.78	52.22
GEPA	91	63.33	95	57	74.44	76.15
APEX	87	68.33	70	57	75.56	71.58
TextGrad	90	66.67	67	54	**76.67**	70.87
MIPRO	82	48.33	58	42	67.78	59.62
EvoSampler	**92**	**76.67**	**97**	**66**	**76.67**	**81.67**

4.3 Ablation Study (RQ2)

In this section, we conduct ablation studies to examine the contributions of the repairability vector and the transferability matrix. Our analysis focuses on two aspects: the individual and combined effects of the two components, and the effect of masking in the transferability matrix. To assess the individual and combined contributions of the two components and the effect of masking, we evaluate the following configurations on HMMT, ScienceWorld and IFBench: (i)**EvoSampler:** Use both components and retain masking in the transferability matrix. (ii)**Without repairability vector:** Use only the transferability matrix and retain masking. (iii)**Without transferability matrix:** Use only the repairability vector. (iv)**Without both components:** Use neither the repairability vector nor the transferability matrix. (v)**Without reparability or masking:** Use only the transferability matrix and remove masking.

The first four configurations assess the individual and combined effects of the components, while the second and fifth isolate the effect of masking: both disable repairability vector and differ only in whether the transferability matrix is masked. Table 2 summarizes the results, with the following key findings:

Table 2: Ablation results on HMMT, ScienceWorld, and IFBench using DeepSeek V4 Flash as the target LLM and DeepSeek V4 Pro as the reflection model. Bold indicates the best result.

	HMMT	ScienceWorld	IFBench	Average
EvoSampler	**76.67**	**97**	**76.67**	**83.45**
w/o transfer matrix	70	**97**	75.56	80.85
w/o repairability vector	70	95	73.33	79.44
w/o both	63.33	95	74.44	77.59
Unmask	68.33	94	68.89	77.07

(1) Contributions of the individual components. Relative to the variant without either component, the Repairability Vector alone raises the average score from 77.59 to 80.85, while the Transfer Matrix alone raises it to 79.44.

(2) Benefits of combining the components. The full method achieves the highest average score of 83.45, exceeding either single-component variant. It outperforms both variants on HMMT and IFBench, and ties the Repairability Vector-only variant on ScienceWorld. These results support combining the two estimates, although their contributions vary across benchmarks.

(3) Benefits of masking. With the Repairability Vector disabled, retaining the prompt-dependent mask raises the average score from 77.07 to 79.44 relative to removing it, with gains on all three benchmarks. This controlled comparison supports conditioning transfer estimates on the validation failures unresolved by the parent prompt.

Table 3: AUC for predicting training-example improvements.

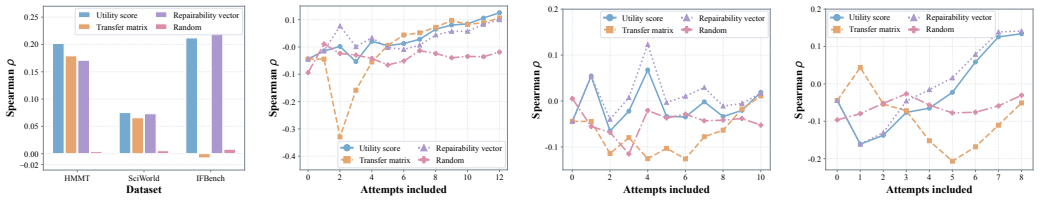
dataset	HMMT	ScienceWorld	IFBench
Repairability vector	0.637	0.555	**0.684**
Utility Score	**0.663**	**0.557**	0.680
Transfer matrix	0.640	0.537	0.494
Random	0.503	0.504	0.507

Table 4: AUC for predicting completion of unresolved validation instances.

Method	HMMT	ScienceWorld	IFBench
Transfer matrix	**0.741**	**0.640**	**0.657**
Utility Score	0.675	0.583	0.470
Repairability vector	0.528	0.520	0.389
Random	0.500	0.505	0.502

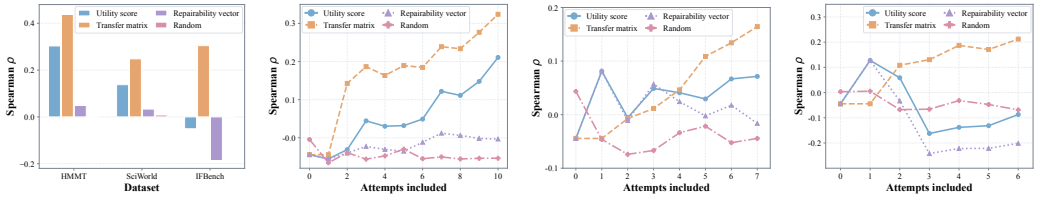
4.4 Predictive Performance of the Components (RQ3)

We analyze optimization traces on HMMT, ScienceWorld, and IFBench to test whether the Repairability Vector, Transfer Matrix, and combined utility score predict training-example improvements and completion of unresolved validation instances. We compare them with a random baseline using AUC for discrimination and Spearman's rank correlation for ranking consistency.



(a) Overall (b) HMMT (c) SciWorld (d) IFBench

Figure 3: Spearman's rank correlation for predicting training-instance outcomes with the Repairability Vector and comparison methods. (a) summarizes performance by dataset; (b), (c), and (d) plot the correlation against the number of revision attempts that include a training instance on HMMT, ScienceWorld, and IFBench, respectively.



(a) Overall (b) HMMT (c) SciWorld (d) IFBench

Figure 4: Spearman's rank correlation for predicting training-to-validation transfer with the Transfer Matrix and comparison methods. (a) summarizes performance by dataset; (b), (c), and (d) plot the correlation against the number of revision attempts that include a training instance on HMMT, ScienceWorld, and IFBench, respectively.

Training-example improvements. Table 3 shows that the combined utility score has the highest AUC on HMMT and SciWorld, while the Repairability Vector performs best on IFBench. Figure 3 shows the corresponding ranking correlations as training instances appear in more revision attempts.

Completion of unresolved validation instances. The Transfer Matrix achieves the highest AUC on all three benchmarks in Table 4. Its rank correlation generally increases with the number of revision attempts containing a training instance, as shown in Figure 4. We further examine how AUC varies with the number of revision attempts containing each training instance and report dataset-level Spearman coefficients; the results and analysis are provided in Appendix D.2.

5 Conclusion

Uniformly sampling training instances in prompt evolution overlooks two determinants of useful feedback: (i) whether an instance helps produce a revision that survives candidate selection, and (ii) whether such revisions address failures the current prompt still makes. This mismatch can waste a limited optimization budget on experience that yields little transferable progress. We introduced **EvoSampler**, a plug-and-play experience sampler that defines a training instance's utility by its expected contribution to validation progress after candidate selection. Its *Repairability Vector* and *Transfer Matrix* estimate reparability and training-to-validation transfer from optimization history; their combination is scored against the current prompt's unresolved failures, and a UCB-based rule preserves exploration. Because **EvoSampler** changes only the construction of revision batches, the underlying reflection model and candidate-selection procedure remain intact. Across two target/reflection-model configurations and five benchmarks spanning mathematical reasoning, interactive decision-making, and instruction following, **EvoSampler** achieves the highest average held-out performance among the compared optimizers.

AI USE STATEMENT

In this work, we used generative AI tools to aid and polish the writing of this paper and to assist with code checking. Other uses of generative AI covered by the ICLR 2027 AI use disclosure are not applicable to this work. We reviewed all AI-assisted work and verified the correctness of the final code and experimental results. We take responsibility for the final content of this work, including all text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not involve human participants or the collection of new personal data. We evaluate our method on publicly released benchmarks, including MATH-500, HMMT, ALFWorld, ScienceWorld, and IFBench, following their licenses and citing their original sources. We do not redistribute raw benchmark data. Interactive experiments are conducted only in simulated environments, and generated actions are not executed on real-world systems. We do not intentionally provide private or confidential information to the language-model services used in our experiments. Test sets are held out from prompt optimization and used only for final evaluation.

Our results are limited to the evaluated tasks and model configurations and should not be interpreted as evidence of safety or suitability for high-stakes deployment. Code and configuration files will be released after removing credentials and restricted data.

REPRODUCIBILITY STATEMENT

To facilitate reproducibility, we provide an anonymous implementation repository at `https://anonymous.4open.science/r/EvoSampler`, which contains the source code, environment specifications, experiment configurations, prompt templates, dataset manifests, and scripts for reproducing the reported results. The main paper and appendix provide detailed descriptions of the proposed method3, theoretical analysisB, experimental setup, datasets, evaluation protocols, and statistical analyses4.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026.

Avinash Amballa, Yashas Malur Saidutta, Chi-Heng Lin, Vivek Kulkarni, and Srinivas Chappidi. VOYAGER: A training free approach for generating diverse datasets using llms. In *ACL (1)*, pp. 17224–17245. Association for Computational Linguistics, 2026.

Anthropic. Introducing claude opus 4.6, 2025. `https://www.anthropic.com/news/claude-opus-4-6`.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. Fireact: Toward language agent fine-tuning. *CoRR*, abs/2310.05915, 2023.

Lichang Chen, Shiyang Li, Jun Yan, Hai Wang, Kalpa Gunaratna, Vikas Yadav, Zheng Tang, Vijay Srinivasan, Tianyi Zhou, Heng Huang, and Hongxia Jin. Alpagasus: Training a better alpaca with fewer data. In *ICLR*. OpenReview.net, 2024.

Sang Keun Choe, Hwijeen Ahn, Juhan Bae, Kewen Zhao, Youngseog Chung, Adithya Pratapa, Willie Neiswanger, Emma Strubell, Teruko Mitamura, Jeff G. Schneider, Eduard H. Hovy, Roger Baker Grosse, and Eric P. Xing. What is your data worth to gpt? llm-scale data valuation with influence functions. In *NeurIPS*, 2025.

Google DeepMind. Gemini 3 flash: frontier intelligence built for speed, 2026. `https://blog.google/products-and-platforms/products/gemini/gemini-3-flash/`.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence. *CoRR*, abs/2606.19348, 2026.

Jasper Dekoninck, Nikola Jovanovic, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin T. Vechev. Beyond benchmarks: Matharena as an evaluation platform for mathematics with llms. *CoRR*, abs/2605.00674, 2026.

Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric P. Xing, and Zhiting Hu. Rlprompt: Optimizing discrete text prompts with reinforcement learning. In *EMNLP*, pp. 3369–3391. Association for Computational Linguistics, 2022.

Shizhe Diao, Pengcheng Wang, Yong Lin, Rui Pan, Xiang Liu, and Tong Zhang. Active prompting with chain-of-thought for large language models. In *ACL (1)*, pp. 1330–1350. Association for Computational Linguistics, 2024.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In *ICML*, volume 235 of *Proceedings of Machine Learning Research*, pp. 13481–13544. PMLR / OpenReview.net, 2024.

Amirata Ghorbani and James Y. Zou. Data shapley: Equitable valuation of data for machine learning. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2242–2251. PMLR, 2019.

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 *ICLR*. OpenReview.net, 2024.

Or Honovich, Uri Shaham, Samuel R. Bowman, and Omer Levy. Instruction induction: From few examples to natural language task descriptions. In *ACL (1)*, pp. 1935–1952. Association for Computational Linguistics, 2023.

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 *ICLR*. OpenReview.net, 2024.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1885–1894. PMLR, 2017.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *NeurIPS*, 2022.

Yongchan Kwon, Eric Wu, Kevin Wu, and James Zou. Datainf: Efficiently estimating data influence in lora-tuned llms and diffusion models. In *ICLR*. OpenReview.net, 2024.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *ICLR*. OpenReview.net, 2024.

Wei Liu, Weihao Zeng, Keqing He, Yong Jiang, and Junxian He. What makes good data for alignment? A comprehensive study of automatic data selection in instruction tuning. In *ICLR*. OpenReview.net, 2024.

Krista Opsahl-Ong, Michael J. Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In *EMNLP*, pp. 9340–9366. Association for Computational Linguistics, 2024.

Xingyuan Pan, Chenlu Ye, Joseph Melkonian, Jiaqi W. Ma, and Tong Zhang. Daunce: Data attribution through uncertainty estimation. *CoRR*, abs/2505.23223, 2025.

Sung Min Park, Kristian Georgiev, Andrew Ilyas, Guillaume Leclerc, and Aleksander Madry. TRAK: attributing model behavior at scale. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pp. 27074–27113. PMLR, 2023.

Archiki Prasad, Peter Hase, Xiang Zhou, and Mohit Bansal. Grips: Gradient-free, edit-based instruction search for prompting large language models. In *EACL*, pp. 3827–3846. Association for Computational Linguistics, 2023.

Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. In *NeurIPS*, 2020.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *EMNLP*, pp. 7957–7968. Association for Computational Linguistics, 2023.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hanna Hajishirzi. Generalizing verifiable instruction following. In *NeurIPS*, 2025.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. In *EMNLP (1)*, pp. 4222–4235. Association for Computational Linguistics, 2020.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *NeurIPS*, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew J. Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In *ICLR*. OpenReview.net, 2021.

Hongjin Su, Jungo Kasai, Chen Henry Wu, Weijia Shi, Tianlu Wang, Jiayi Xin, Rui Zhang, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. Selective annotation makes language models better few-shot learners. In *ICLR*. OpenReview.net, 2023.

Fei Wang, Si Si, Cho-Jui Hsieh, and Inderjit S. Dhillon. APEX: automated prompt engineering expert with dynamic data selection. *CoRR*, abs/2606.11459, 2026.

Jingtan Wang, Xiaoqiang Lin, Rui Qiao, Pang Wei Koh, Chuan-Sheng Foo, and Bryan Kian Hsiang Low. Nice: Non-differentiable evaluation metric-based data selection for instruction tuning. In *ICLR 2025 Workshop on Navigating and Addressing Data Problems for Foundation Models*, 2025.

Ruoyao Wang, Peter A. Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader? In *EMNLP*, pp. 11279–11298. Association for Computational Linguistics, 2022.

Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. LESS: selecting influential data for targeted instruction tuning. In *ICML*, volume 235 of *Proceedings of Machine Learning Research*, pp. 54104–54132. PMLR / OpenReview.net, 2024.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *ICLR*. OpenReview.net, 2024.

Mert Yüksekgönül, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic "differentiation" via text. *CoRR*, abs/2406.07496, 2024.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. LIMA: less is more for alignment. In *NeurIPS*, 2023a.

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 *ICLR*. OpenReview.net, 2023b.

A Related Work

A.1 Prompt Evolution

Prompt optimization aims to improve LLM performance by automatically refining natural-language instructions. Early studies explored diverse optimization strategies, including gradient-guided search (Shin et al., 2020), reinforcement learning (Deng et al., 2022), edit-based search (Prasad et al., 2023), and instruction induction (Honovich et al., 2023), followed by LLM-driven methods that generate and optimize candidate prompts (Zhou et al., 2023b; Yang et al., 2024). Among these approaches, evolutionary optimization with natural-language feedback has emerged as an important paradigm: APO (Pryzant et al., 2023) uses error feedback to guide prompt mutation, PromptBreeder (Fernando et al., 2024) and EvoPrompt (Guo et al., 2024) combine LLMs with evolutionary search, while GEPA (Agrawal et al., 2026) extends prompt evolution to complex AI systems through a Genetic-Pareto framework. Despite these advances in search, APEX (Wang et al., 2026) identifies data efficiency as a key bottleneck, arguing that treating the development set as a static benchmark wastes computation on uninformative examples, and introduces a heuristic dynamic data selection strategy. However, how to more precisely model the information value of training examples and dynamically allocate data during prompt evolution remains underexplored.

A.2 Data Selection

LIMA (Zhou et al., 2023a) and LESS (Xia et al., 2024) show that a small number of carefully selected examples can be sufficient for effective alignment, highlighting the importance of data quality over quantity. This insight has motivated automated selection methods such as AlpaGasus (Chen et al., 2024) and DEITA (Liu et al., 2024), which filter low-quality data, and the influence-function-based method of (Choe et al., 2025), which estimates how training examples affect the outputs of large language models. This line of work naturally extends to prompt optimization. Active-Prompt (Diao et al., 2024) selects uncertain questions for human annotation of chain-of-thought exemplars, while (Su et al., 2023) use similarity-based heuristics to select representative examples for annotation and retrieve demonstrations at test time. Apex (Wang et al., 2026) extends informative-example identification and dynamic curation to black-box prompt optimization. However, these methods largely rely on heuristic criteria and provide limited theoretical interpretability, as they do not clearly explain how each example affects prompt performance; this can make the selection process unstable. To address this issue, EvoSampler selects examples that are expected to improve prompt performance.

A.3 Training Data Attribution

Training Data Attribution (TDA) seeks to establish a causal and quantitative relationship between model behavior and its training data. Existing methods mainly fall into two categories: (1) gradient-based approaches, which exploit parameter-level optimization dynamics and model curvature: existing methods estimate influence via second-order Hessian curvature (Koh & Liang, 2017), cumulative SGD gradient trajectories (Pruthi et al., 2020), projected gradient kernels (Park et al., 2023), or low-rank Hessian approximations (Kwon et al., 2024); and (2) black-box approaches, which assess data value from output behavior without internal model access: typical techniques rely on game-theoretic marginal utility (Ghorbani & Zou, 2019), loss covariances under perturbations (Pan et al., 2025), or direct modeling of non-differentiable task metrics (Wang et al., 2025). Although effective for parameter fine-tuning and dataset pruning, these methods are poorly suited to optimizing textual components of self-evolving agents because text is non-differentiable, internal model information is often inaccessible, and data collection is costly. By introducing *Repairability Vector* and *Transfer Matrix*, EvoSampler develops a predictive TDA method for text optimization.

B Method & Proof Details

This appendix derives the training-instance utility in Section 3.1, proves its two theorems, and presents the statistical principles used by the estimators and sampler. The parent prompt P is held fixed within a revision round. Before a round is run, its batch, candidate, and selection outcome may be random.

B.1 From Prompt Optimization to Masked Repair Utility

The objective in Eq. 1 concerns expected task performance. On the validation set, let $F(P) = \sum_{j=1}^{N} s_j(P)$ be the aggregate binary score used to evaluate a prompt. A revision batch $\mathcal{B}$ produces candidate $\widetilde{P}$, which is evaluated against a parent prompt P and submitted to the pool-based selection rule in Eq. 3. Let $A \in \{0, 1\}$ indicate whether $\widetilde{P}$ is retained. For the observed candidate, its retained, parent-relative score contrast is

$$C(P, \widetilde{P}) = A\big(F(\widetilde{P}) - F(P)\big). \tag{28}$$

This contrast describes the candidate relative to its parent; selection itself updates the prompt pool rather than replacing P with $\widetilde{P}$. No expectation is involved at this stage. An expectation enters when we score a training instance before its batch produces a candidate.

Because every s_j is binary, define the numbers of newly solved and previously solved but newly missed validation instances by

$$G(P, \widetilde{P}) = \sum_{j=1}^{N} \big(1 - s_j(P)\big) s_j(\widetilde{P}), \tag{29}$$

$$L(P, \widetilde{P}) = \sum_{j=1}^{N} s_j(P) \big(1 - s_j(\widetilde{P})\big). \tag{30}$$

The identity

$$s_j(\widetilde{P}) - s_j(P) = \big(1 - s_j(P)\big) s_j(\widetilde{P}) - s_j(P) \big(1 - s_j(\widetilde{P})\big) \tag{31}$$

then decomposes Eq. 28 as

$$C(P, \widetilde{P}) = A\big(G(P, \widetilde{P}) - L(P, \widetilde{P})\big). \tag{32}$$

The sampling decision aims to identify training feedback that repairs the parent's unresolved instances. We therefore use the newly solved component of a retained revision as the training-instance signal. A regression on a previously solved instance can arise from incidental changes introduced during prompt revision and can add noise to the attribution of repair progress to the sampled feedback. With $r_j(P) = 1 - s_j(P)$, masking isolates the targeted signal:

$$AG(P, \widetilde{P}) = A \sum_{j=1}^{N} r_j(P) s_j(\widetilde{P}) = A \left\langle \mathbf{s}(\widetilde{P}), \mathbf{r}(P) \right\rangle. \tag{33}$$

Candidate retention remains represented by A; the mask specifies which validation outcomes are attributed to the training batch. The paired masking ablation in Table 2 compares two variants with reparability estimation disabled and finds consistently higher scores with masking on the evaluated tasks. This result supports using the masked repair signal for sample selection.

For an instance t_i, the candidate from a future batch is not yet known. Conditioning on t_i being included, its expected utility is

$$U_i(P) = \mathbb{E}\Big[A \left\langle \mathbf{s}(\widetilde{P}), \mathbf{r}(P) \right\rangle \;\Big|\; P,\ t_i \in \mathcal{B} \Big]. \tag{34}$$

This is Eq. 6 with the fixed parent prompt made explicit.

B.2 Proof of Theorem 1: Shared-Credit Utility

The masked repair signal observed in round q is

$$g^{(q)} = A^{(q)} \left\langle \mathbf{s}\Big(\widetilde{P}^{(q)}\Big), \mathbf{r}\Big(P^{(q)}\Big) \right\rangle. \tag{35}$$

The batch generates one candidate. Under the shared-credit rule, each participating training instance receives this same outcome and every nonparticipating instance receives zero. Hence $u_{qi} = B_{qi} g^{(q)}$. Collecting the entries and using $\mathbf{b}^{(q)} = [B_{q1}, \ldots, B_{qK}]^\top$ gives

$$\mathbf{u}^{(q)} = A^{(q)} \mathbf{b}^{(q)} \left\langle \mathbf{s}\Big(\widetilde{P}^{(q)}\Big), \mathbf{r}\Big(P^{(q)}\Big) \right\rangle, \tag{36}$$

which is Eq. 9.

B.3 Proof of Theorem 2: Conditional Outcome Model

The two factors in Eq. 11 describe successive events: a batch containing t_i yields a retained revision, and that retained revision solves an unresolved validation instance. Conditioning transfer on retention accounts for the dependence between these events.

Fix P and abbreviate $\{t_i \in \mathcal{B}\}$ by $B_i = 1$. Since $r_j(P)$ is fixed, linearity of conditional expectation in Eq. 34 yields

$$U_i(P) = \sum_{j=1}^{N} r_j(P)\mathbb{E}\Big[As_j(\widetilde{P}) \mid P, B_i = 1\Big]. \tag{37}$$

Only terms with $r_j(P) = 1$ remain. For each such j, $As_j(\widetilde{P})$ is the indicator that the candidate is both retained and correct on v_j. The conditional probability chain rule gives

$$\mathbb{E}\Big[As_j(\widetilde{P}) \mid P, B_i = 1\Big] = \mathbb{P}\Big(A = 1, s_j(\widetilde{P}) = 1 \mid P, B_i = 1\Big) \tag{38}$$

$$= \rho_i(P)\mu_{ij}(P), \tag{39}$$

where

$$\rho_i(P) = \mathbb{P}(A = 1 \mid P, B_i = 1), \tag{40}$$

$$\mu_{ij}(P) = \mathbb{P}\Big(s_j(\widetilde{P}) = 1 \mid A = 1, P, B_i = 1, r_j(P) = 1\Big). \tag{41}$$

If $\rho_i(P) = 0$, the joint probability is zero and the value assigned to the conditional transfer probability is immaterial. Substitution into Eq. 37 gives

$$U_i(P) = \rho_i(P) \sum_{j=1}^{N} \mu_{ij}(P) r_j(P). \tag{42}$$

Under the pooled outcome model in Section 3.1, the conditional propensities over the optimization horizon are represented by the stable parameters ρ_i and μ_{ij}. Thus, within that model,

$$U_i(P) = \rho_i \sum_{j=1}^{N} \mu_{ij} r_j(P). \tag{43}$$

Stacking these K expressions, with $\mathbf{R} = \mathrm{diag}(\boldsymbol{\rho})$ and $\mathbf{M} = [\mu_{ij}]_{K\times N}$, proves $\mathbf{U}(P) = \mathbf{RMr}(P)$.

The conditional factorization uses the probability chain rule, without an independence assumption between acceptance and validation correctness. The pooled model summarizes previous prompt and batch contexts for scoring a new parent prompt. Eq. 13 substitutes estimates of its two factors before observing the outcome of round q.

B.4 Beta–Bernoulli Modeling and Laplace Smoothing

Consider a binary outcome $z \in \{0, 1\}$ with success probability θ. For observations $z_1, \ldots, z_m$ that are conditionally independent given θ, let $s = \sum_{h=1}^{m} z_h$ and $f = m - s$. Their Bernoulli likelihood is

$$p(z_{1:m} \mid \theta) = \prod_{h=1}^{m} \theta^{z_h}(1-\theta)^{1-z_h} = \theta^s(1-\theta)^f. \tag{44}$$

With a $\mathrm{Beta}(a, b)$ prior, whose density is proportional to $\theta^{a-1}(1-\theta)^{b-1}$, Bayes' rule gives

$$p(\theta \mid z_{1:m}) \propto \theta^{a+s-1}(1-\theta)^{b+f-1}, \tag{45}$$

$$\theta \mid z_{1:m} \sim \mathrm{Beta}(a + s, b + f), \tag{46}$$

$$\mathbb{E}[\theta \mid z_{1:m}] = \frac{a+s}{a+b+m}. \tag{47}$$

Each success therefore increments the first count and each failure increments the second. Setting $a = b = 1$ yields

$$\widehat{\theta} = \frac{s+1}{m+2}, \tag{48}$$

the Laplace add-one estimate. It starts at $1/2$ and, for finite m, remains strictly between zero and one. The uniform $\mathrm{Beta}(1,1)$ prior gives a probabilistic interpretation of this smoothing rule.

The two components in Section 3 apply this result to different eligible observations. For ρ_i, every previous round with $B_{qi} = 1$ contributes the binary retention outcome $A^{(q)}$, producing Eq. 15. For μ_{ij}, a round contributes $s_j(\widetilde{P}^{(q)})$ only when $B_{qi}A^{(q)}r_j(P^{(q)}) = 1$, producing Eq. 19. The posterior means then take the forms in Eqs. 17 and 21. They are substituted as point estimates in the utility score.

B.5 UCB-Style Sampling and Initial Coverage

Upper-confidence-bound methods motivate exploration bonuses that decrease with the number of observations and increase slowly with the sampling budget. For example, if n independent rewards lie in $[0,1]$ and have common mean m, Hoeffding's inequality gives

$$\mathbb{P}\left(m > \overline{X}_n + \sqrt{\frac{2\log T}{n}}\right) \le T^{-4}, \qquad T \ge 2. \tag{49}$$

This motivates the square-root dependence on $\log T/n$. EvoSampler uses that form as an exploration bonus alongside its prompt-dependent utility estimate, with λ controlling the relative weight. Its resulting index is Eq. 25.

One property of the batch-selection rule follows directly from its index definition. Suppose $1 \le k \le K$, each round selects k distinct instances, unobserved instances have index $+\infty$, and observed instances have finite indices. If m_q instances have not been sampled before round q, the next batch contains $\min\{k, m_q\}$ of them. Therefore,

$$m_{q+1} = \max\{m_q - k, 0\}. \tag{50}$$

Starting from $m_1 = K$, all instances have been sampled after at most $\lceil K/k \rceil$ rounds. This property holds even when $\lambda = 0$, because the infinite index for unobserved instances is specified separately. After initial coverage, Eq. 24 gives larger bonuses to less-sampled instances; random tie-breaking gives uniform selection among instances with equal indices.

C Experimental Details

C.1 Datasets and Data Splits

MATH-500 contains competition-level mathematics problems, while HMMT draws on problems from the Harvard–MIT Mathematics Tournament. ALFWorld tests household task completion in a text-based environment, and ScienceWorld tests scientific tasks requiring multi-step interaction. IFBench measures whether responses satisfy verifiable constraints in natural-language instructions. For every benchmark, training instances provide feedback for prompt revision, validation instances guide candidate selection, and held-out test instances are used only for final evaluation. The three splits are disjoint.

For MATH-500, ALFWorld, and ScienceWorld, we use 40 training, 45 validation, and 100 test instances per benchmark. HMMT has fewer available problems, so we split them by competition year: 30 problems from February 2025 for training, 30 from February 2022 for validation, and 60 from February 2023 and February 2024 for testing. The official reference solutions for the training problems are used to evaluate task outcomes during optimization. For IFBench, the training and validation sets contain 40 and 45 instances, respectively; final evaluation uses 90 instances sampled from its test split across constraint types. IFBench outputs are scored as complete responses rather than at the token or interaction-step level.

C.2 Baselines

We compare **EvoSampler** with two evolutionary prompt optimizers and two other methods for optimizing textual components:

- **GEPA** (Agrawal et al., 2026) evolves textual components and uses instance-level validation outcomes for Pareto-based candidate selection.
- **APEX** (Wang et al., 2026) adapts the data used for prompt revision and candidate evaluation based on recent optimization outcomes.
- **TextGrad** (Yüksekgönül et al., 2024) turns execution feedback into textual gradients that guide revisions through a backpropagation-like process.
- **MIPRO** (Opsahl-Ong et al., 2024) searches over instructions and few-shot examples to improve task performance.

Table 5: Generation limits and interaction constraints for different datasets.

Dataset	Token Limit	Generation Configuration
MATH-500	8,191	-
HMMT	16,383	-
IFBench	8,191	-
AlfWorld	8,191	At most 10 interaction steps
ScienceWorld	18,431	At most 40 interaction steps

C.3 Optimization and Evaluation Protocol

All experiments use random seed 42. EvoSampler uses an optimization budget of 1,000 evaluations; test-set evaluations are excluded from this budget. An optimization run stops when its budget is exhausted.

When evaluating candidate prompts, we use a sampling temperature of 0.2 by default and 1.0 for the interactive tasks ALFWorld and ScienceWorld. Models used to generate natural-language feedback are run at temperature 1.0. Generation and interaction are subject to the dataset-specific limits in Table 5. For ALFWorld and ScienceWorld, the token limits are cumulative across at most 10 and 40 interaction steps, respectively.

C.4 Implementation Details

We implement **EvoSampler** by modifying the GEPA framework to change how training instances are selected for prompt revision. The Bernoulli parameters of the Repairability Vector and Transfer Matrix each start with a uniform $\mathrm{Beta}(1,1)$ prior, equivalent to one positive and one negative pseudo-count. We update the Repairability Vector from candidate-retention outcomes for batches containing each training instance. We update the Transfer Matrix from the outcomes of retained candidates only for validation instances unresolved by the parent prompt. The UCB exploration weight is 0.2, and each revision batch contains three training instances.

Table 6: Benchmark results for different optimizers using Gemini 3 Flash as the target LLM and Claude Opus 4.6 as the reflection model. Bold indicates the best score.

	MATH-500	HMMT	ScienceWorld	ALFWorld	IFBench	Average
base	84	70.00	85	72	74.44	77.09
GEPA	**92**	71.67	85	98	83.33	86.00
APEX	85	68.33	79	88	82.22	80.51
TextGrad	**92**	**76.67**	**87**	79	82.22	83.38
MIPRO	88	**76.67**	84	72	81.11	80.36
EvoSampler	**92**	73.33	**87**	**99**	**86.67**	**87.60**

D ADDITIONAL EXPERIMENTS

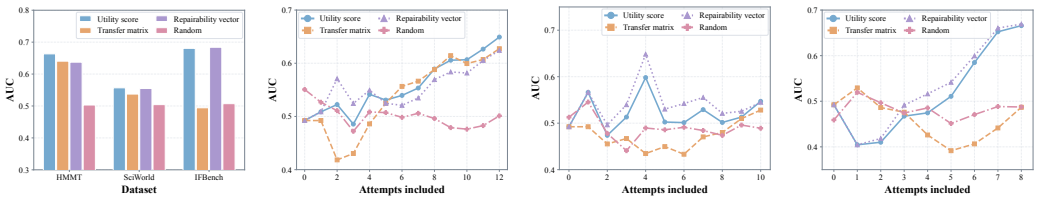
D.1 ADDITIONAL OVERALL PERFORMANCE

We repeat the benchmark comparison with Gemini 3 Flash as the target LLM and Claude Opus 4.6 as the reflection model.

Table 6 shows that **EvoSampler** achieves the highest average score of 87.60, exceeding GEPA, the strongest baseline on average, by 1.60 percentage points. It attains the best or joint-best score on four of the five benchmarks. The exception is HMMT, where **EvoSampler** scores 73.33, below the 76.67 achieved by TextGrad and MIPRO. Thus, the overall gain extends to the second model configuration, although it is not uniform across tasks.

D.2 ADDITIONAL PREDICTIVE ANALYSES

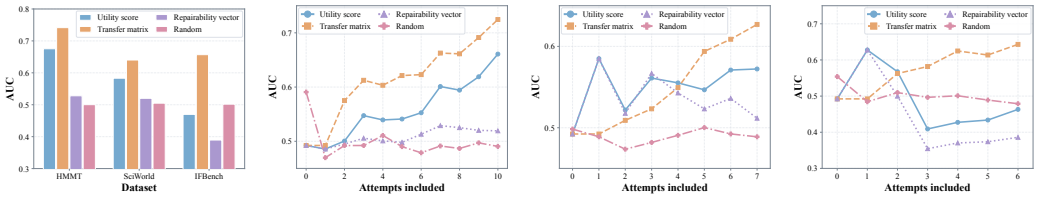
To complement the aggregate AUC results and rank-correlation curves in the main text, we examine AUC by the number of revision attempts containing each training instance and report dataset-level Spearman correlations.



(a) Overall (b) HMMT (c) SciWorld (d) IFBench

Figure 5: AUC for predicting training-instance outcomes with the Repairability Vector and comparison methods. (a) summarizes performance by dataset; (b), (c), and (d) plot AUC against the number of revision attempts that include a training instance on HMMT, ScienceWorld, and IFBench, respectively.

Training-instance improvements. Figure 5 shows that the utility score and Repairability Vector tend to have higher AUC at larger attempt counts on HMMT and IFBench. The ScienceWorld curves fluctuate more, and none of the benchmarks exhibits a strictly monotonic trend. These count-stratified results suggest that the available optimization history is informative, while also showing that predictive performance varies across tasks and observation counts.



(a) Overall (b) HMMT (c) SciWorld (d) IFBench

Figure 6: AUC for predicting training-to-validation transfer with the Transfer Matrix and comparison methods. (a) summarizes performance by dataset; (b), (c), and (d) plot AUC against the number of revision attempts that include a training instance on HMMT, ScienceWorld, and IFBench, respectively.

Completion of unresolved validation instances. In Figure 6, the Transfer Matrix generally becomes more discriminative at larger attempt counts and exceeds the other predictors in the highest-count bin of each benchmark. The intermediate bins fluctuate, particularly on ScienceWorld and IFBench, so the relationship should not be interpreted as monotonic improvement with every additional attempt.

Table 7: Spearman's rank correlation for predicting training-example improvements.

Method	HMMT	ScienceWorld	IFBench
Repairability vector	0.171	0.073	**0.221**
Utility Score	**0.202**	**0.075**	0.212
Transfer matrix	0.179	0.066	-0.008
Random	0.004	0.005	0.008

Table 8: Spearman's rank correlation for predicting completion of unresolved validation instances.

method	HMMT	ScienceWorld	IFBench
Transfer matrix	**0.438**	**0.248**	**0.305**
Utility Score	0.303	0.138	-0.051
Repairability vector	0.049	0.033	-0.186
Random	0.000	0.008	0.003

Rank-correlation summary. Tables 7 and 8 provide the corresponding dataset-level Spearman coefficients. For predicting training-instance improvements, the utility score ranks highest on HMMT and ScienceWorld, although its advantage over the Repairability Vector on ScienceWorld is small (0.075 versus 0.073); the Repairability Vector ranks highest on IFBench. For predicting completion of unresolved validation instances, the Transfer Matrix ranks highest on all three benchmarks. The modest coefficients for training-instance improvements indicate that these estimates provide a ranking signal rather than precise predictions for individual instances.