BREAD: TRANSFER-AWARE ON-POLICY DISTILLATION FROM BOUNDARY EXPERIENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Repeated sampling reveals correct programs for tasks that a language model fails to solve greedily. These boundary tasks contrast failed decisions with verified repairs, providing experience for cross-task learning. We introduce `BREAD`—Boundary Repair Experience for Adaptive Distillation—to connect repair experience with **cross-task update utility**. `BREAD` extracts textual repair cards from boundary tasks and uses them to condition a frozen teacher on task-only student rollouts. Temporary updates measure changes in correct-program likelihood across source clusters, and cluster-level utility scores route experience to training receivers. During distillation, candidate AdamW updates start from a shared snapshot and are selected by correctness-probe improvement under **retention and displacement constraints**. The resulting student generates programs from task-only inputs. Across four model families and four programming benchmarks, `BREAD` achieves average gains of 7.88 percentage points in macro-average greedy accuracy over the initial models and 1.82 points over the strongest baseline selected separately for each model. Paired analysis separates newly solved tasks from lost solutions, while component and carrier diagnostics examine how repair extraction, utility routing, and update selection contribute to reusable task-only student behavior. The source code for `BREAD` is available through our anonymous repository.

1 INTRODUCTION

Repeated sampling can uncover correct programs that a model does not reliably produce in a single attempt (Chen et al., 2021; Yu et al., 2026). We focus on *boundary tasks*, where greedy decoding fails but a correct program appears within a fixed sampling budget. Such tasks expose accessible success that has yet to become the model's default behavior. On-policy distillation (OPD) turns stronger conditional behavior into a student's default behavior by supervising student-generated trajectories (Agarwal et al., 2024; Ye et al., 2026; Hübotter et al., 2026). Latent reasoning methods also retain and transfer inference-time improvements (Zhang et al., 2025b). The resulting question is how to turn a boundary-task repair into a capability that the student applies to other tasks.

Boundary tasks provide both a failure to explain and a successful alternative against which to examine it. Learning from successful and failed trajectories is a basis for reusable reasoning memory (Ouyang et al., 2026). Consider a one-dimensional implementation of 0–1 knapsack. Iterating over capacities in ascending order can reuse an item within the same update, whereas descending order preserves the constraint that each item is used once. A useful experience records this constraint and the condition under which it applies. It can then guide a different problem with the same state-dependency structure. The successful program establishes a working solution; the repair rule identifies the reusable decision and its applicability conditions. We call the source-linked repair card extracted from this failure–success contrast a boundary experience for cross-task learning.

The first challenge is to identify *experiences with useful cross-task learning effects*. A source-task success supplies a working trajectory from which to extract a repair. Transfer depends on the receiving task's structure and on the representation used to carry the repair (Feng et al., 2026). Text can express an invariant explicitly; a continuous latent can encode a compact conditioning signal; and a latent displacement can retain the change from initialization to an improved representation. LatentEvolve develops latent optimization, displacement-based transfer, and experience consolidation (Zhang et al., 2025b). Studies of experience modules and cross-task skills examine how execution feedback and

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.

skill granularity affect reuse (Li et al., 2026a; Feng et al., 2026). These findings motivate measuring experience-induced changes in student behavior across tasks.

The second challenge is to determine *which transferred experience should enter the student's parameters*. Context-conditioned distillation enriches supervision with accumulated experience in OPCD, feedback in SDPO, and learned soft prompts in PromptSD (Ye et al., 2026; Hübotter et al., 2026; Ma et al., 2026). The teacher's benefit from an experience and the student's benefit from learning it are distinct quantities. An experience can help a conditioned teacher yet induce a student update that interferes with previously learned decisions. Recent analyses relate distillation outcomes to teacher–student compatibility and transferable knowledge (Li et al., 2026b). Meta-reweighting and meta-distillation use learner feedback to improve supervision (Ren et al., 2018; Zhou et al., 2022), while studies of distillation gradients examine the quality of the learning signal (Armandpour et al., 2026). These studies motivate evaluating the experience-induced update itself on other training tasks.

We introduce Boundary Repair Experience for Adaptive Distillation (`BREAD`), which uses cross-task parameter utility to route experience and select student updates. A frozen teacher reads repair cards extracted from execution-verified failure–success pairs and scores the student's task-only trajectories. To measure transfer, we temporarily apply an experience-conditioned update on a receiver and evaluate the change in correct-program negative log-likelihood (NLL) on probes from other source clusters. Cluster-level utility records guide experience routing. A second comparison selects among actual AdamW trials, requiring correctness-probe improvement, non-increasing mean retention NLL, and a bounded parameter displacement. Execution-based diagnostics assess the resulting task-only behavior using verified program outcomes on independent evaluation tasks.

Our contributions connect verified boundary repair with cross-task learning and retention:

- ✦ ***Transfer-aware repair experience.*** We extract repair cards from greedy failures and verified successes, then route them by cross-task NLL changes after temporary student updates. This connects experience selection to its effect on the learner's parameters.
- ❖ ***Retention-constrained distillation.*** We compare actual AdamW trials from a shared optimizer state and select the largest correctness-probe gain under retention and displacement constraints. A null candidate preserves the current state when every proposed update fails this test.
- ★ ***Task-only transfer across model families.*** Across four model families and four benchmarks, `BREAD` improves macro-average accuracy by 7.88 points over vanilla and 1.82 points over each model's strongest baseline. The comparison spans dense and mixture-of-experts backbones. Paired and component analyses assess solution gains, retention, and the roles of extraction, routing, and update selection in task-only transfer.

2 Related Work

Experience learning and cross-task transfer. Agent memory systems organize prior interactions into linked records or hierarchical stores (Xu et al., 2025; Zhang et al., 2025a). ReasoningBank extracts reusable strategies from successful and failed trajectories (Ouyang et al., 2026), while Memento and MemRL learn utility-guided memory selection (Zhou et al., 2025; Zhang et al., 2026). Experience representation also shapes reuse: SEAM learns structured experience from execution outcomes (Li et al., 2026a), LatentEvolve transfers optimized latent displacements (Zhang et al., 2025b), and recent skill-transfer work studies the effects of granularity and representation (Feng et al., 2026). Memory Transfer Learning examines how abstraction affects cross-domain reuse in coding agents (Kim et al., 2026). These methods connect past experience to future decisions through memory organization, retrieval, and representation. `BREAD` builds on this view by extracting execution-verified boundary repairs and measuring the cross-task parameter effect of learning from them, using the same repair record to connect teacher guidance with student learning.

Context-conditioned on-policy distillation. Generalized knowledge distillation trains students on their own generated prefixes (Agarwal et al., 2024). Context-conditioned methods enrich teacher supervision with accumulated experience in OPCD (Ye et al., 2026), feedback in SDPO (Hübotter et al., 2026), and successful and unsuccessful rollouts in MOPD (Yu et al., 2026). SEED supplies hindsight skills and PromptSD learns soft-prompt teachers, allowing conditioning information to be absorbed into student parameters (Wu et al., 2026; Ma et al., 2026). Skill-SD conditions the

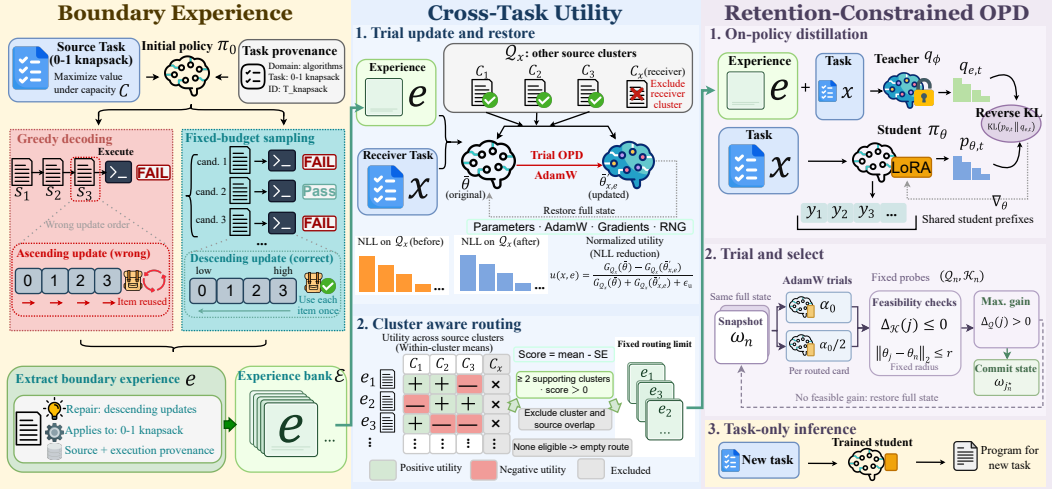


Figure 1: **Overview of BREAD.** ***Left:*** greedy failures and verified sampled successes yield boundary-repair cards. ***Middle:*** temporary OPD updates are restored after measuring cross-task NLL reduction; cluster-level utilities route cards to receivers. ***Right:*** a frozen, experience-conditioned teacher supervises student rollouts. AdamW trials share a complete snapshot and fixed probes; the selected feasible trial updates the adapter under retention and displacement constraints. The trained student generates programs from task-only inputs on new tasks.

teacher on trajectory-derived skills while the student uses task-only rollouts (Wang et al., 2026). RLCSD and COPD further shape supervision by contrasting teacher contexts (Pan et al., 2026; Ruan et al., 2026). This teacher–student interface allows additional reasoning information to guide training while the student retains its task-only input at inference. BREAD couples this interface with cross-task experience routing and learner-side update selection, linking the teacher's contextual benefit to the student's acquisition of reusable behavior across tasks.

Learning utility and distillation optimization. Meta-reweighting and MetaDistil use learner feedback to assess supervision (Ren et al., 2018; Zhou et al., 2022). Recent OPD analyses examine teacher–student compatibility (Li et al., 2026b), gradient quality (Armandpour et al., 2026), and sample difficulty and reasoning length (Hou et al., 2026). Objective-level methods adapt teacher-uncertainty handling, token supervision, and advantage design (Jin et al., 2026; Jia et al., 2026; Yang et al., 2026; Zhao et al., 2026). These studies locate learning utility at the level of teaching signals and their interaction with the student. BREAD evaluates actual parameter updates using correct-program NLL on separate training probes and constrains the mean NLL change on previously solved tasks. Stored measurements route experience, while current probe losses select the update.

3 METHOD

Figure 1 summarizes BREAD through boundary-experience extraction, cross-task utility routing, and retention-constrained distillation. Utility trials measure cross-task effects and restore the student state; a selected distillation trial supplies the persistent adapter update. Execution-based diagnostics then assess transfer on held-out tasks using task-only student prompts and verified program outcomes.

3.1 PRELIMINARIES

Let x denote a programming task and $y = (y_1, \ldots, y_T)$ its generated response. The execution verifier $V(x, y)$ returns one for a passing program, zero for a verified failure, and $\perp$ for an unresolved evaluation. A source cluster $h(x)$ groups tasks with common provenance or near-duplicate content. Experience sources $\mathcal{S}$, training receivers $\mathcal{R}$, and training probes have separate roles; cluster exclusion keeps utility measurements separate from the receiver being updated. Development and final evaluation tasks are held apart from these training operations.

Each backbone provides a frozen teacher q_ϕ, an initial student policy π_0, and its own verified experience bank. The student π_θ learns only LoRA parameters θ. A correctness-probe set $\mathcal{Q}$ supplies verified reference programs; a retention set $\mathcal{K}$ supplies references for previously solved tasks. Both are training resources, evaluated with task-only student inputs. The complete trial state comprises adapter parameters, AdamW state, gradient buffers, and random-number generator states.

3.2 Boundary Repair Experience

Boundary tasks expose a useful contrast: the initial policy fails under greedy decoding but reaches a verified solution within a fixed sampling budget. Let $\mathcal{S}^-$ contain source tasks with verified greedy failures. Sampling K_s responses from π_0 defines the boundary set:

$$\mathcal{B}(\pi_0, K_s) = \left\{ x \in \mathcal{S}^- : \sum_{k=1}^{K_s} \mathbf{1}\left\{ V(x, y_x^{(k)}) = 1 \right\} \geq 1 \right\}. \tag{1}$$

A *boundary experience* is a repair card e extracted from a source's verified greedy failure and sampled success. It records the failed decision, the correction in the passing program, applicability conditions, source identity, and execution provenance. The decision may concern a program invariant, the order in which program states are updated, or a boundary condition.

The main method uses textual cards as additional teacher context. For receiver x, the teacher reads the task and the routed experience; the student reads the task alone. Model-specific banks preserve the connection between an experience and the backbone that produced it. The carrier comparison also examines optimized latent slots and their displacement from initialization as alternative teacher-conditioning signals, following the transfer view of LatentEvolve (Zhang et al., 2025b).

3.3 Experience-Conditioned On-Policy Distillation

At training iteration n, the student generates a receiver response using $\pi_{\theta_n}(\cdot \mid x)$. The frozen teacher scores the same response prefixes with experience e. Write $p_{\theta,t} = \pi_\theta(\cdot \mid x, y_{<t})$ and $q_{e,t} = \mathrm{sg}[q_\phi(\cdot \mid x, e, y_{<t})]$, where sg fixes teacher probabilities. For a rollout batch $\mathcal{A}$, the objective averages reverse KL over response positions, using the student-prefix distillation interface of GKD and OPCD (Agarwal et al., 2024; Ye et al., 2026):

$$\mathcal{L}_{e,n}(\theta; \mathcal{A}) = \frac{1}{|\mathcal{A}|} \sum_{(x,y)\in\mathcal{A}} \frac{1}{|y|} \sum_{t=1}^{|y|} \sum_{v\in\mathcal{V}} p_{\theta,t}(v) \log \frac{p_{\theta,t}(v)}{q_{e,t}(v)}. \tag{2}$$

Here $\mathcal{V}$ is the common teacher–student vocabulary. We compute the full-vocabulary divergence on answer positions and hold generated tokens fixed during differentiation. Gradients pass through student probabilities; the teacher and its contextual input remain fixed. The same loss produces restored utility trials in Section 3.4 and candidate updates for selection in Section 3.5.

Actual optimizer trials. Let $\mathbf{g}_{e,n}$ be the gradient of Equation (2) at the saved student state. AdamW updates its first and second moments using this gradient (Loshchilov & Hutter, 2019). Denote their bias-corrected values by $\widehat{\mathbf{m}}^+_{e,n}$ and $\widehat{\mathbf{v}}^+_{e,n}$. At trial learning rate α, the candidate adapter parameters are

$$\theta_{e,\alpha} = (1 - \alpha\lambda)\theta_n - \alpha \frac{\widehat{\mathbf{m}}^+_{e,n}}{\sqrt{\widehat{\mathbf{v}}^+_{e,n}} + \epsilon_{\mathrm{a}}}. \tag{3}$$

The division and square root act elementwise; λ is weight decay and $\epsilon_{\mathrm{a}} > 0$ stabilizes the denominator. Moments use the saved AdamW history and the candidate gradient, with bias correction at the next optimizer step. Each candidate starts from the same complete snapshot, enabling a paired comparison of the experience-conditioned update and its trial learning rate.

3.4 Routing by Cross-Task Parameter Utility

The routing signal measures how learning from a receiver changes the student's likelihood on other tasks. For task–reference pairs $(x, y_x^+) \in \mathcal{P}$, define the mean reference-program NLL:

$$G_{\mathcal{P}}(\theta) = -\frac{1}{|\mathcal{P}|} \sum_{(x,y_x^+)\in\mathcal{P}} \frac{1}{|y_x^+|} \sum_{t=1}^{|y_x^+|} \log \pi_\theta(y^+_{x,t} \mid x, y^+_{x,<t}). \tag{4}$$

Reference programs have verified execution outcomes. Averaging within each program and then across tasks gives each probe equal weight. The same functional defines correctness loss on $\mathcal{Q}$ and retention loss on $\mathcal{K}$, with the two sets differing in their assigned training role.

Measuring transfer. To evaluate experience e on receiver x, temporarily apply its OPD update from a saved state $\bar{\theta}$, producing $\bar{\theta}'_{x,e}$. Let $\mathcal{Q}_x$ contain the measurement probes from other source clusters. Evaluate these same probes before and after the trial, and record the normalized loss reduction

$$u(x,e) = \frac{G_{\mathcal{Q}_x}(\bar{\theta}) - G_{\mathcal{Q}_x}(\bar{\theta}'_{x,e})}{G_{\mathcal{Q}_x}(\bar{\theta}) + G_{\mathcal{Q}_x}(\bar{\theta}'_{x,e}) + \epsilon_{\mathrm{u}}}. \tag{5}$$

We set $\epsilon_{\mathrm{u}} = 10^{-12}$ and restore parameters, optimizer state, gradients, and random state after each measurement, so routing ranks experiences without advancing the student. Positive utility means the temporary update lowers NLL on the cross-task probes. Each record is associated with its experience, receiver cluster, and measurement configuration, preserving the receiver and source-cluster context needed to interpret the measured parameter effect.

Cluster-aware routing. Average the recorded utilities within each receiver source cluster to obtain $u_{e,h}$. When routing a new receiver x, exclude its entire cluster $h(x)$ from the supporting records. Let $\mathcal{H}_e^{-x}$ be the remaining clusters, $n_{e,x} = |\mathcal{H}_e^{-x}|$, and $\bar{u}_e^{-x}$ their mean utility. The score subtracts one standard error from the equally weighted cluster mean:

$$b_e^{(-h(x))} = \bar{u}_e^{-x} - \sqrt{\frac{\sum_{h \in \mathcal{H}_e^{-x}} (u_{e,h} - \bar{u}_e^{-x})^2}{n_{e,x}(n_{e,x}-1)}}. \tag{6}$$

An experience requires at least two supporting clusters and a positive score. Eligible experiences are ranked within the fixed routing limit; an empty eligible set gives an empty route. Source exclusion also separates the experience's origin from the current receiver. Utility records are collected before distillation and remain fixed during training, ranking experiences by their measured cross-cluster learning effects on probes outside each receiver's source cluster.

Execution-level interpretation. The NLL score provides a training proxy for learning utility. On a diagnostic pool $\mathcal{D}$ with resolved execution outcomes, behavioral parameter utility measures the task-only success change after adapter displacement d:

$$U_{\mathrm{param}}(d;\theta,\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \left[\mathbb{E}_{y \sim \pi_{\theta+d}(\cdot|x)} V(x,y) - \mathbb{E}_{y \sim \pi_{\theta}(\cdot|x)} V(x,y) \right]. \tag{7}$$

This behavioral quantity is assessed through generated programs and execution. Context utility separately compares the frozen teacher's execution success with and without an experience. These diagnostics evaluate contextual guidance and learned behavior, while Equation (5) supplies the parameter-learning proxy used to construct the routing scores.

3.5 Selecting Updates with Retention Constraints

Stored cross-task utility scores route experience cards; update selection tests the resulting candidates against the current student. Following the learner-feedback perspective of meta-reweighting and meta-distillation (Ren et al., 2018; Zhou et al., 2022), we test actual optimizer updates on training probes. Each candidate uses either the original learning rate α_0 or its half, starting from the same complete snapshot. Probe sets $\mathcal{Q}_n$ and $\mathcal{K}_n$ are fixed across all candidates in that iteration.

For trial j, let θ_j be the candidate parameters from Equation (3). Its correctness-proxy gain and retention cost are measured relative to the current parameters θ_n:

$$\begin{aligned} \Delta_{\mathcal{Q}}(j) &= G_{\mathcal{Q}_n}(\theta_n) - G_{\mathcal{Q}_n}(\theta_j) \\ \Delta_{\mathcal{K}}(j) &= G_{\mathcal{K}_n}(\theta_j) - G_{\mathcal{K}_n}(\theta_n). \end{aligned} \tag{8}$$

The retention threshold is zero: the mean NLL on the selected retention probes must not increase in an accepted step. A fixed radius r also bounds the adapter displacement. The radius is the largest actual AdamW displacement on an independent training calibration trajectory. Calibration ends by resetting the state, and the resulting radius stays fixed throughout student training.

Table 1: **Task-only code generation.** Three-seed mean greedy accuracy (%); Mean weights benchmarks equally. Subscripts give changes from displayed vanilla scores (pp). Best/second best are bold/underlined. EOPD† shares `BREAD`'s teacher context; aggregation details are in Appendix B.3.

Method	Greedy accuracy ↑				Mean ↑
	APPS	CodeContests	TACO	LiveCodeBench	
`Qwen3.5-9B` (9B)					
Vanilla	42.13	22.15	9.29	44.21	29.45
Verified SFT	45.88$_{\uparrow 3.75}$	25.73$_{\uparrow 3.58}$	13.86$_{\uparrow 4.57}$	54.01$_{\uparrow 9.80}$	34.87$_{\uparrow 5.42}$
DPO (NeurIPS 2023)	45.12$_{\uparrow 2.99}$	23.27$_{\uparrow 1.12}$	15.01$_{\uparrow 5.72}$	59.16$_{\uparrow 14.95}$	35.64$_{\uparrow 6.19}$
OPCD (arXiv 2026)	46.78$_{\uparrow 4.65}$	24.83$_{\uparrow 2.68}$	14.71$_{\uparrow 5.42}$	60.90$_{\uparrow 16.69}$	36.81$_{\uparrow 7.36}$
MOPD	<u>47.89</u>$_{\uparrow 5.76}$	<u>26.17</u>$_{\uparrow 4.02}$	<u>16.78</u>$_{\uparrow 7.49}$	59.68$_{\uparrow 15.47}$	<u>37.63</u>$_{\uparrow 8.18}$
PromptSD (code)	46.21$_{\uparrow 4.08}$	23.49$_{\uparrow 1.34}$	14.82$_{\uparrow 5.53}$	**63.26**$_{\uparrow 19.05}$	36.95$_{\uparrow 7.50}$
EOPD† (ICML 2026)	47.43$_{\uparrow 5.30}$	25.95$_{\uparrow 3.80}$	16.59$_{\uparrow 7.30}$	58.46$_{\uparrow 14.25}$	37.11$_{\uparrow 7.66}$
`BREAD` **(Ours)**	**50.75**$_{\uparrow 8.62}$	**27.52**$_{\uparrow 5.37}$	**18.03**$_{\uparrow 8.74}$	<u>62.98</u>$_{\uparrow 18.77}$	**39.82**$_{\uparrow 10.37}$
`Mistral Small 3.2` (24B)					
Vanilla	27.12	11.41	7.74	44.21	22.62
Verified SFT	21.73$_{\downarrow 5.39}$	9.84$_{\downarrow 1.57}$	6.75$_{\downarrow 0.99}$	48.63$_{\uparrow 4.42}$	21.74$_{\downarrow 0.88}$
DPO	30.54$_{\uparrow 3.42}$	12.30$_{\uparrow 0.89}$	10.36$_{\uparrow 2.62}$	51.69$_{\uparrow 7.48}$	26.22$_{\uparrow 3.60}$
OPCD	28.71$_{\uparrow 1.59}$	10.74$_{\downarrow 0.67}$	10.07$_{\uparrow 2.33}$	49.22$_{\uparrow 5.01}$	24.69$_{\uparrow 2.07}$
MOPD	31.83$_{\uparrow 4.71}$	<u>12.98</u>$_{\uparrow 1.57}$	12.57$_{\uparrow 4.83}$	51.23$_{\uparrow 7.02}$	27.15$_{\uparrow 4.53}$
PromptSD (code)	**33.94**$_{\uparrow 6.82}$	11.19$_{\downarrow 0.22}$	11.06$_{\uparrow 3.32}$	<u>57.80</u>$_{\uparrow 13.59}$	<u>28.50</u>$_{\uparrow 5.88}$
EOPD†	28.97$_{\uparrow 1.85}$	10.96$_{\downarrow 0.45}$	<u>12.83</u>$_{\uparrow 5.09}$	56.48$_{\uparrow 12.27}$	27.31$_{\uparrow 4.69}$
`BREAD` **(Ours)**	<u>32.59</u>$_{\uparrow 5.47}$	**13.42**$_{\uparrow 2.01}$	**14.16**$_{\uparrow 6.42}$	**59.96**$_{\uparrow 15.75}$	**30.03**$_{\uparrow 7.41}$

Let $\mathcal{C}_n$ contain the evaluated experience and learning-rate candidates. Feasibility requires both retention and bounded displacement relative to the same saved student state:

$$\mathcal{T}_n = \{j \in \mathcal{C}_n : \Delta_{\mathcal{K}}(j) \leq 0 \ \wedge \ \|\theta_j - \theta_n\|_2 \leq r\}\,. \tag{9}$$

The null trial uses $\theta_\varnothing = \theta_n$ and has zero correctness gain. Selection maximizes the measured correctness-probe gain over the feasible trials and this null option:

$$j_n^\star = \underset{j \in \{\varnothing\} \cup \mathcal{T}_n}{\arg\max} \left[G_{\mathcal{Q}_n}(\theta_n) - G_{\mathcal{Q}_n}(\theta_j)\right]. \tag{10}$$

The null trial wins when no feasible candidate gives a strict reduction in correctness-probe NLL. An accepted trial commits its complete candidate state; rejection restores the saved state. The retention comparison is a per-step average relative to the current student. Execution-based accuracy and lost solves measure how these proxy decisions affect generated programs after training.

Algorithm 1 in Appendix E summarizes training with complete trial state ω. The learned adapter generates programs from task-only prompts on new tasks.

4 Experiments

We evaluate task-only performance on four programming benchmarks, analyze the components enabling repair transfer, and relate parameter utility to execution gains.

4.1 Experimental Setup

Backbones. We use `Qwen3.5-9B` (9B) (Qwen Team, 2026) for the main table and diagnostics, and `Mistral Small 3.2` (24B) (Mistral AI, 2025) for a larger family. `Phi-4-mini-instruct` (3.8B) provides a compact dense backbone (Appendix C.1); `Gemma` provides a mixture-of-experts backbone (Appendix C.2). Each backbone supplies its teacher, experience bank, and LoRA student; methods share the same initialization and adapter capacity.

Table 2: Component ablations. Three-seed mean accuracy (%) on `Qwen3.5-9B`; subscripts give changes from full **BREAD** (pp).

Variant	Mean ↑
BREAD (Ours)	**39.82**
Success pool	33.85$_{\downarrow 5.97}$
Positive only	29.36$_{\downarrow 10.46}$
NN routing	32.45$_{\downarrow 7.37}$
No selection	30.72$_{\downarrow 9.10}$
No retention	31.69$_{\downarrow 8.13}$
First-order	27.56$_{\downarrow 12.26}$

Table 3: Carrier diagnostics: three-seed mean use rate (%) and context/parameter accuracy changes (pp); 512 trials per pool per seed.

Carrier	Use	Δ_{ctx} ↑	Δ_{param} ↑
Text (NN)	100	+4.10	+1.56
Text	100	+6.25	+4.69
$\mathbf{z}^\star$	100	**+7.03**	+1.56
$\Delta\mathbf{z}$ (NN)	100	+3.12	+2.34
$\Delta\mathbf{z}$	100	+3.91	**+6.25**
$\mathbf{z}_0$	100	+0.78	−0.39
Random	100	−0.59	−0.98

Benchmarks. Training uses the training splits of APPS (Hendrycks et al., 2021), CodeContests (Li et al., 2022), and TACO (Li et al., 2023), partitioned into source, receiver, and probe pools. Evaluation covers their test pools and LiveCodeBench `release_v6` (Jain et al., 2025), using the same 6,916 eligible tasks in each run. Appendix B.1 gives dataset counts and exclusions.

Baselines. We compare **direct supervision**: verified SFT and DPO (Rafailov et al., 2023); **context-conditioned distillation**: OPCD (Ye et al., 2026), MOPD (Yu et al., 2026), and PromptSD (Ma et al., 2026); and **objective adaptation**: EOPD (Jin et al., 2026). MOPD retains same-task rollout contexts, PromptSD (code) learns task-group soft prompts, and EOPD† shares **BREAD**'s routed context. Methods share permissible data; Appendix B.2 details the coding adaptations.

Metrics. Main results average task-only greedy accuracy over independent training seeds 42, 123, and 97011; Mean weights the four benchmarks equally. All models are evaluated in no-thinking mode. Paired analysis separates newly solved tasks from lost solutions (Appendix B.4). The training-rule comparison is a single-seed diagnostic. Appendices A.1, B.3, and B.5 give training configurations, rules for averaging and rounding scores, and diagnostic sampling protocols.

4.2 Task-Only Performance

Consistent gains in mean accuracy. Table 1 shows mean accuracy rising from 29.45% to 39.82% on `Qwen` and from 22.62% to 30.03% on `Mistral`, exceeding their strongest baselines by 2.19 and 1.53 points. For `Phi` (Table 8), gains over vanilla and the strongest baseline are 6.72 and 1.62 points; for `Gemma` (Table 9), they are 7.00 and 1.94 points. Across all four families, equally weighted average gains over vanilla and the strongest baseline are 7.88 and 1.82 points.

BREAD leads in thirteen of sixteen model–benchmark combinations and exceeds OPCD in all sixteen. PromptSD (code) leads on LiveCodeBench with `Qwen` and APPS with `Mistral`; EOPD† leads on APPS with `Gemma` under the shared greedy evaluation protocol.

Gains exceed losses on every benchmark. Averaged over three runs, `Qwen` gains 777 solves and loses 87 against vanilla; `Mistral` gains 553 and loses 73 (Table 10). Against OPCD, the mean counts are 361 versus 112 and 424 versus 90. Net gains are positive on every benchmark (Figure 2). LiveCodeBench gives the largest gains over vanilla: 18.77 and 15.75 points.

4.3 Experience and Component Analysis

Experience extraction and update selection both matter. Table 2 and Figure 3 compare six variants against the complete method's 39.82% mean accuracy. Success-only extraction costs 10.46 points, while using initially solved sources costs 5.97 points, supporting repairs extracted from paired failures and successes. Nearest-neighbor routing costs 7.37 points; removing selection or retention costs 9.10 or 8.13 points. First-order ranking incurs the largest drop, 12.26 points, favoring measured optimizer trials. Appendix C.4 defines the variants and settings.

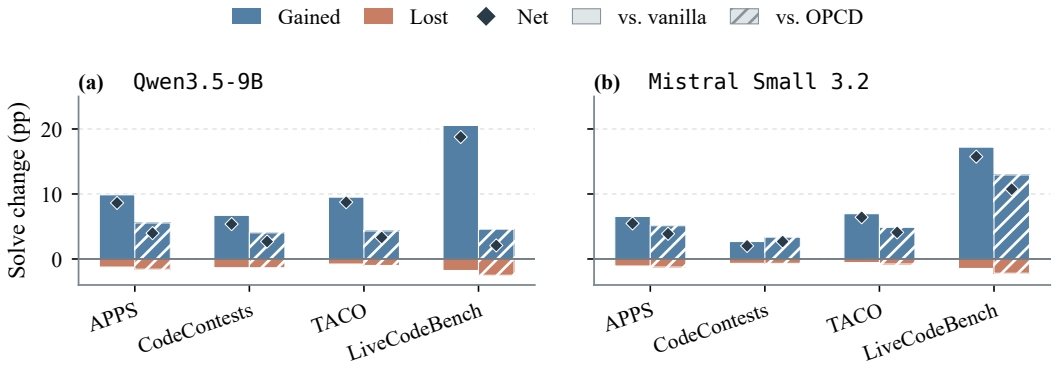


Figure 2: **Newly acquired and lost solutions.** Bars show three-seed mean gained solves above zero and lost solves below zero, normalized by benchmark size. Paired bars compare **BREAD** with vanilla (solid) and OPCD (hatched); diamonds mark the net change. Runs share 6,916 tasks per backbone.

Table 4: Training-rule comparison on Qwen. Single-seed scores (%) with shared training settings, bank, router, and probes; subscripts give changes from OPCD (pp).

Training rule	Greedy	Pass@1	Oracle@8
OPCD	29.12	28.27	47.72
+ replay	24.53$_{\downarrow 4.59}$	22.57$_{\downarrow 5.70}$	43.66$_{\downarrow 4.06}$
+ probe losses	15.46$_{\downarrow 13.66}$	17.34$_{\downarrow 10.93}$	42.02$_{\downarrow 5.70}$
Meta-reweight	31.95$_{\uparrow 2.83}$	28.13$_{\downarrow 0.14}$	45.59$_{\downarrow 2.13}$
Norm-matched	30.74$_{\uparrow 1.62}$	27.41$_{\downarrow 0.86}$	47.42$_{\downarrow 0.30}$
EOPD†	30.39$_{\uparrow 1.27}$	28.92$_{\uparrow 0.65}$	50.28$_{\uparrow 2.56}$
BREAD (Ours)	**33.41**$_{\uparrow 4.29}$	**30.12**$_{\uparrow 1.85}$	**52.87**$_{\uparrow 5.15}$

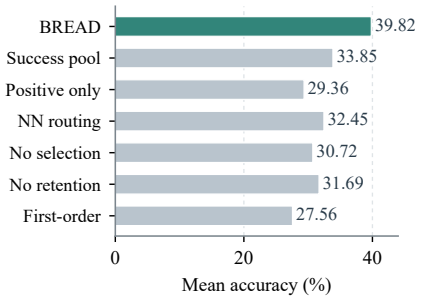


Figure 3: Ablation accuracy (%) on Qwen. Table 2 gives changes from full **BREAD** for the six variants.

Context benefit and parameter transfer favor different carriers. Table 3 tests context benefit and task-only parameter transfer in separate pools of 512 trials per seed. Absolute slots $\mathbf{z}^{\star}$ lead in context with 7.03 points but yield 1.56 after an update; displacements $\Delta\mathbf{z}$ yield 3.91 and 6.25, respectively. Utility routing raises parameter transfer from 1.56 to 4.69 points for text and from 2.34 to 6.25 for displacements. This ranking reversal motivates judging experience by the student's update utility. Latent carriers share the text-derived router; Appendix C.5 gives the comparison protocol.

4.4 Parameter Utility and Update Selection

Higher utility bins show stronger execution transfer. Figure 4 groups 30 candidate updates per seed into five NLL-utility bins. Execution-change bounds rise from $[-3.00, -2.42]$ to $[5.62, 5.92]$ points: negative-utility bins have negative bounds, while nonnegative bins have positive bounds. This ordering associates higher proxy utility with stronger transfer on independent execution tasks. Appendix C.6 describes the diagnostic and its unresolved-outcome bounds.

Update selection improves greedy and sampled quality. In the single-seed comparison of training rules (Table 4), **BREAD** reaches 33.41% greedy accuracy, 30.12% pass@1, and 52.87% oracle@8, exceeding OPCD by 4.29, 1.85, and 5.15 points. It also surpasses replay, joint probe losses, meta-reweighting (Ren et al., 2018), norm matching, and EOPD†. The 2.59-point oracle@8 gain over EOPD† extends eight-sample solution coverage. Appendix C.7 visualizes all three metrics.

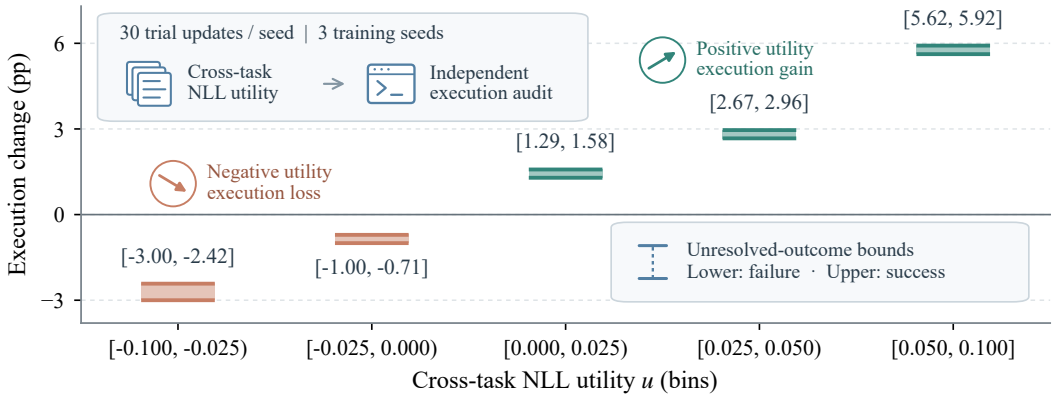


Figure 4: **Parameter utility and execution transfer.** Each of three training seeds supplies 30 candidate updates, evaluated on an independent execution pool. Intervals bound the mean change within NLL-utility bins by assigning unresolved outcomes to failure or success.

4.5 Case Study: Transferring a Repair Invariant

Figure 5 uses a constructed example to connect a reusable repair to routing and update selection. **Extraction** contrasts knapsack programs: reading updated states reuses one item and returns 6, while previous-stage reads give the correct value 3. **Routing** illustrates cross-cluster NLL scoring with source exclusions. The same invariant applies to the one-hop graph, where previous-layer reads replace the invalid two-edge cost of 2 with the valid one-edge cost of 5. **Selection** depicts AdamW trials from one snapshot under correctness, retention, and displacement criteria. Appendix D works through the state updates and their corrections in both tasks.

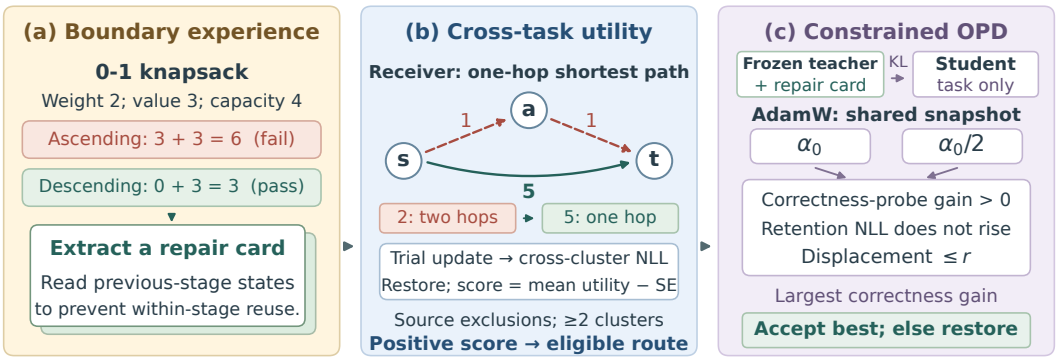


Figure 5: **Constructed repair example.** A knapsack repair yields a rule for reading previous-stage states. The two-edge path costs 2 but violates the one-hop budget; the valid direct path costs 5. Routing and constrained trials illustrate the method's decision rules.

5 Conclusion

BREAD transfers boundary repairs into task-only student behavior through cross-task utility routing and constrained distillation. Repair cards capture reusable decisions, and optimizer trials select updates under retention and displacement constraints. Across four model families and four benchmarks, mean gains are 7.88 points over vanilla and 1.82 over each model's strongest baseline, covering dense and mixture-of-experts backbones. Paired and component analyses connect these results to repair extraction, routing, and update selection. Future work will extend utility estimation to a broader range of programming tasks and explore richer representations of repair experience.

AI USE STATEMENT

We used language models to correct grammar and refine wording in author-written prose and to identify a subset of the related work discussed in Section 2. AI coding assistants helped convert author-provided Excel data into the LaTeX format of Tables 2, 3, 4, and 10, and helped draw the bar chart in Figure 2 and the visualization in Figure 4. AI assistance also supported manuscript-level checks of the clarity and consistency of methodological descriptions and experimental-result presentation, as well as translation for an internal Chinese reading version. AI tools were not used to originate the research ideas, formulate the research objectives, run experiments, or generate the reported numerical results. The authors retain responsibility for the scientific judgments, interpretation of experimental findings, and the accuracy and integrity of the final manuscript.

ETHICS STATEMENT

This work studies how verified program repairs can improve code generation, using established programming benchmarks. More reliable code generation can support software development and programming education. Generated programs can still contain functional errors or security vulnerabilities, so practical use requires independent testing, security review, and appropriate human oversight. Execution-based benchmark evaluation measures correctness on the specified tests; application-specific validation remains essential when generated code is used in consequential settings. Responsible use also requires respecting the terms of the underlying models and datasets.

REPRODUCIBILITY STATEMENT

Section 3 specifies repair extraction, cross-task utility routing, and constrained distillation, with the complete workflow in Algorithm 1. Section 4.1 describes the experimental setup; Appendices B.1 and B.2 detail the benchmark task pools and baseline adaptations. Appendix A.1 records training configurations, random seeds, checkpoint selection, and no-thinking evaluation, with backbone-specific settings in Appendices C.1 and C.2. Appendices A.2 and A.3 describe the experience-card format, policy inputs, routing rules, calibration, and optimizer-state handling. Appendices B.3, B.4, and B.5 define result aggregation, paired comparisons, and sampled metrics. Component, carrier, and utility diagnostics are specified in Appendices C.4, C.5, and C.6, respectively. We provide the implementation through an anonymous code repository.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/5be69a584901a26c521c2b51e40a4c20-Abstract-Conference.html.

Mohammadreza Armandpour, Fatih Ilhan, David Harrison, Ajay Jaiswal, Duc N. M. Hoang, Fartash Faghri, Yizhe Zhang, Minsik Cho, and Mehrdad Farajtabar. Unmasking on-policy distillation: Where it helps, where it hurts, and why. *arXiv preprint arXiv:2605.10889*, 2026. doi: 10.48550/arXiv.2605.10889. URL https://arxiv.org/abs/2605.10889.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. doi: 10.48550/arXiv.2107.03374. URL https://arxiv.org/abs/2107.03374.

Yiyang Feng, Biddut Sarker Bijoy, Niranjan Balasubramanian, and Jiawei Zhou. Break it down, pass it on: Cross-task skill transfer in LLM agents. *arXiv preprint arXiv:2608.20274*, 2026. doi: 10.48550/arXiv.2608.20274. URL https://arxiv.org/abs/2608.20274.

Google DeepMind. Gemma 4 26B-A4B-IT model card. Official model card, 2026. URL https://huggingface.co/google/gemma-4-26B-A4B-it.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with APPS. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/hash/c24cd76e1ce41366a4bbe8a49b02a028-Abstract-round2.html.

Zhinan Hou, Jiaqi Zhang, Xunliang Cai, and Keyou You. What matters in on-policy distillation? a perspective on data efficiency and data selection. *arXiv preprint arXiv:2609.05198*, 2026. URL https://arxiv.org/abs/2609.05198.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and Andreas Krause. Reinforcement learning via self-distillation. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2601.20802.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida I. Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/94074dd5a072d28ff75a76dabed43767-Abstract-Conference.html.

Nan Jia, Haojin Yang, Xing Ma, Jiesong Lian, Shuailiang Zhang, Weipeng Zhang, Ke Zeng, Xunliang Cai, and Zequn Sun. Asymmetric on-policy distillation: Bridging exploitation and imitation at the token level. *arXiv preprint arXiv:2605.06387*, 2026. doi: 10.48550/arXiv.2605.06387. URL https://arxiv.org/abs/2605.06387.

Woogyeol Jin, Taywon Min, Yongjin Yang, Dennis Wei, Yi Zhou, Swanand Ravindra Kadhe, Nathalie Baracaldo, and Kimin Lee. Entropy-aware on-policy distillation of language models. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2603.07079.

Kangsan Kim, Minki Kang, Taeil Kim, Yanlai Yang, Mengye Ren, and Sung Ju Hwang. Memory transfer learning: How memories are transferred across domains in coding agents. *arXiv preprint arXiv:2604.14004*, 2026. doi: 10.48550/arXiv.2604.14004. URL https://arxiv.org/abs/2604.14004.

Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, Guang Liu, Zhi Jin, and Ge Li. TACO: Topics in algorithmic COde generation dataset. *arXiv preprint arXiv:2312.14852*, 2023. doi: 10.48550/arXiv.2312.14852. URL https://arxiv.org/abs/2312.14852.

Xuancheng Li, Haitao Li, Yujia Zhou, Yiqun Liu, and Qingyao Ai. Beyond experience retrieval: Learning to generate utility-optimized structured experience for frozen LLMs. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 39467–39482. Association for Computational Linguistics, 2026a. doi: 10.18653/v1/2026.acl-long.1831. URL https://aclanthology.org/2026.acl-long.1831/.

Yaxuan Li, Yuxin Zuo, Bingxiang He, Jinqian Zhang, Chaojun Xiao, Cheng Qian, Tianyu Yu, Huanang Gao, Wenkai Yang, Zhiyuan Liu, and Ning Ding. Rethinking on-policy distillation of large language models: Phenomenology, mechanism, and recipe. *arXiv preprint arXiv:2604.13016*, 2026b. doi: 10.48550/arXiv.2604.13016. URL https://arxiv.org/abs/2604.13016.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158. URL https://arxiv.org/abs/2203.07814.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Yingzi Ma, Zichen Zhu, Ming Jiang, and Chaowei Xiao. One student, many teachers: Multi-task on-policy distillation via soft-prompt privileged context. *arXiv preprint arXiv:2607.18293*, 2026. doi: 10.48550/arXiv.2607.18293. URL https://arxiv.org/abs/2607.18293.

Microsoft. Phi-4-mini-instruct model card. Official model card, 2025. URL https://huggingface.co/microsoft/Phi-4-mini-instruct.

Mistral AI. Mistral-Small-3.2-24B-Instruct-2506. Model card, 2025. URL https://huggingface.co/mistralai/Mistral-Small-3.2-24B-Instruct-2506.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=jL7fwchScm.

Leyi Pan, Shuchang Tao, Yunpeng Zhai, Lingzhe Zhang, Zhaoyang Liu, Bolin Ding, Aiwei Liu, and Lijie Wen. RLCSD: Reinforcement learning with contrastive on-policy self-distillation. *arXiv preprint arXiv:2606.11709*, 2026. doi: 10.48550/arXiv.2606.11709. URL https://arxiv.org/abs/2606.11709.

Qwen Team. Qwen3.5-9B model card. Official model card, 2026. URL https://huggingface.co/Qwen/Qwen3.5-9B.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, pp. 53728–53741, 2023. doi: 10.52202/075280-2338. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/a85b405ed65c6477a4fe8302b5e06ce7-Abstract-Conference.html.

Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4334–4343. PMLR, 2018. URL https://proceedings.mlr.press/v80/ren18a.html.

Jiacheng Ruan, Jun Tang, Wenzhen Yuan, Ting Liu, Shuai Bai, Dayiheng Liu, Zhibo Yang, and Yuzhuo Fu. Contrastive on-policy distillation. *arXiv preprint arXiv:2607.19046*, 2026. doi: 10.48550/arXiv.2607.19046. URL https://arxiv.org/abs/2607.19046.

Hao Wang, Guozhi Wang, Han Xiao, Yufeng Zhou, Yue Pan, Jichao Wang, Ke Xu, Yafei Wen, Xiaohu Ruan, Xiaoxin Chen, and Honggang Qi. Skill-SD: Skill-conditioned self-distillation for multi-turn LLM agents. *arXiv preprint arXiv:2604.10674*, 2026. doi: 10.48550/arXiv.2604.10674. URL https://arxiv.org/abs/2604.10674.

Jinyang Wu, Shuo Yang, Zhengxi Lu, Fan Zhang, Yuhao Shen, Lang Feng, Haoran Luo, Zheng Lian, Shuai Zhang, Zhengqi Wen, and Jianhua Tao. SEED: Self-evolving on-policy distillation for agentic reinforcement learning. *arXiv preprint arXiv:2607.14777*, 2026. doi: 10.48550/arXiv.2607.14777. URL https://arxiv.org/abs/2607.14777.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-0593. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/19909c36f51abc4856b4560aff3d36d6-Abstract-Conference.html.

Run Yang, Runpeng Dai, Jie Sun, Jielei Zhang, Fan Zhou, Hongtu Zhu, Peiyi Li, and Longwen Gao. Influence-directed distillation: Solving the diversity bottleneck in sampled-token on-policy distillation. *arXiv preprint arXiv:2608.29846*, 2026. URL https://arxiv.org/abs/2608.29846.

Tianzhu Ye, Li Dong, Xun Wu, Shaohan Huang, and Furu Wei. On-policy context distillation for language models. *arXiv preprint arXiv:2602.12275*, 2026. doi: 10.48550/arXiv.2602.12275. URL https://arxiv.org/abs/2602.12275.

Weichen Yu, Xiaomin Li, Yizhou Zhao, Xiaoze Liu, Ruowang Zhang, Haixin Wang, Yinyi Luo, Chen Henry Wu, Gaurav Mittal, Matt Fredrikson, and Yu Hu. Multi-rollout on-policy distillation via peer successes and failures. *arXiv preprint arXiv:2605.12652*, 2026. doi: 10.48550/arXiv.2605.12652. URL https://arxiv.org/abs/2605.12652.

Guibin Zhang, Muxin Fu, Kun Wang, Frank Wan, Miao Yu, and Shuicheng Yan. G-Memory: Tracing hierarchical memory for multi-agent systems. In *Advances in Neural Information Processing Systems*, volume 38, 2025a. doi: 10.52202/085713-0439. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/136a45cd9b841bf785625709a19c6508-Abstract-Conference.html.

Guibin Zhang, Fanci Meng, Guancheng Wan, Zherui Li, Kun Wang, Zhenfei Yin, Lei Bai, and Shuicheng Yan. LatentEvolve: Self-evolving test-time scaling in latent space. *arXiv preprint arXiv:2509.24771*, 2025b. doi: 10.48550/arXiv.2509.24771. URL https://arxiv.org/abs/2509.24771.

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Zhuo Li, Yujie Zheng, Weinan Zhang, Ying Wen, Zhiyu Li, Feiyu Xiong, Yutao Qi, Bo Tang, and Muning Wen. MemRL: Self-evolving agents via runtime reinforcement learning on episodic memory. *arXiv preprint arXiv:2601.03192*, 2026. doi: 10.48550/arXiv.2601.03192. URL https://arxiv.org/abs/2601.03192.

Hanyang Zhao, Haoxian Chen, Han Lin, Genta Indra Winata, David Yao, and Wenpin Tang. OPD+: Rethinking the advantage design for on-policy distillation. *arXiv preprint arXiv:2606.01039*, 2026. doi: 10.48550/arXiv.2606.01039. URL https://arxiv.org/abs/2606.01039.

Huichi Zhou, Yihang Chen, Siyuan Guo, Xue Yan, Kin Hei Lee, Zihan Wang, Ka Yiu Lee, Guchun Zhang, Kun Shao, Linyi Yang, and Jun Wang. Memento: Fine-tuning LLM agents without fine-tuning LLMs. *arXiv preprint arXiv:2508.16153*, 2025. doi: 10.48550/arXiv.2508.16153. URL https://arxiv.org/abs/2508.16153.

Wangchunshu Zhou, Canwen Xu, and Julian McAuley. BERT learns to teach: Knowledge distillation with meta learning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7037–7049. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.acl-long.485. URL https://aclanthology.org/2022.acl-long.485/.

A Implementation Details

A.1 Backbone-Specific Configuration

Each backbone supplies a model-specific repair bank and a frozen teacher. We train three independent students with seeds 42, 123, and 97011, holding the base-model revision, teacher, bank, receiver pool, correctness and retention pools, and test tasks fixed across runs. The two main-table backbones use 32 training iterations, LoRA rank 8 with scaling 16, and 1,024-token training rollouts (Table 5). All methods share a 4,096-token evaluation budget within each backbone.

Evaluation mode and checkpoints. All backbones use **no-thinking mode** for both greedy and sampled evaluation. We evaluate the final checkpoint from each training run. The three-seed comparisons average these final-checkpoint results; the training-rule diagnostic uses results from one of the three training seeds for every method and metric.

Table 5: Configuration of the main-table `BREAD` checkpoints. Probe counts distinguish the available pool from the subset evaluated during update selection.

Setting	`Qwen3.5-9B`	`Mistral Small 3.2`
Training seeds	42, 123, 97011	42, 123, 97011
Training iterations	32	32
LoRA rank / scaling	8 / 16	8 / 16
Training response limit	1,024 tokens	1,024 tokens
Candidate learning rates	2×10^{-4}, 10^{-4}	2×10^{-4}, 10^{-4}
Correctness / retention pool	8 / 8	8 / 8
Selection probes per iteration	4 / 4	8 / 8
Retention tolerance	0	0
Radius calibration	8 receivers	8 receivers
Evaluation response limit	4,096 tokens	4,096 tokens

A.2 Experience Cards and Policy Inputs

A repair card connects a source-specific failure to a reusable programming decision. Its content describes the failed decision, the correction in the passing program, and the conditions under which that correction applies; source identity and execution provenance associate the card with its originating task. The failure–success pair supplies the contrast used for extraction, while the applicability description guides the conditioned teacher on a different receiver. Appendix D expands the state-update invariant in Figure 5 as a concrete example.

Table 6 distinguishes the inputs used during distillation and evaluation. The teacher and student score the same student-generated response prefixes, with the repair card supplied to the teacher. Reverse KL is computed over the shared vocabulary and averaged over response positions, with teacher probabilities held fixed when differentiating the loss. Probe evaluation instead scores verified reference programs under the task-only student. This separates the experience-conditioned teaching signal from the task-only behavior measured by the probes and final evaluation.

Table 6: **Information used at each stage.** $y_{<t}$ is a student-generated prefix; $y^{+}_{<t}$ is a verified reference prefix. Experience conditioning is part of the training-time teacher interface.

Operation	Policy input	Output used
Student rollout	Receiver task x	Candidate response y
Student distillation score	x and $y_{<t}$	$\pi_\theta(\cdot \mid x, y_{<t})$
Frozen teacher score	x, repair card e, and $y_{<t}$	$q_\phi(\cdot \mid x, e, y_{<t})$
Correctness / retention probe	Probe task x and $y^{+}_{<t}$	Reference-program NLL
Final evaluation	Test task x	Generated program for execution

A.3 Routing and Candidate Transactions

Routing aggregates cross-task utility within source clusters and scores an experience by the cluster mean minus one standard error. Supporting records exclude the receiver's cluster, and the experience source must satisfy the source-overlap exclusion. A route requires at least two supporting clusters and a positive score. Eligible experiences are ranked up to the fixed routing limit, with an empty route when none qualifies. `Mistral` uses four correctness probes for utility and a routing limit of two.

Cluster weighting. Equation (6) first averages measurements within each source cluster and then weights the remaining clusters equally. This prevents a cluster with many recorded receivers from dominating the route solely through its size. Subtracting the standard-error term lowers the score of experiences whose measured benefit varies across supporting clusters. The receiver-cluster exclusion is applied before computing both the mean and this penalty.

Shared optimizer state. Each temporary update starts from a saved transaction state containing adapter parameters, AdamW moments and counters, gradient buffers, and random-number generator states. The teacher remains frozen while the student gradient is computed from the response-position reverse KL. Candidate trials use the original learning rate and its half-rate alternative. Restoring the complete state makes each comparison refer to the same optimizer history and starting parameters.

Calibration and acceptance. For the main-table backbones, the displacement radius is calibrated on eight training receivers using actual AdamW steps. The largest measured displacement sets r, after which the calibration state is reset and r remains fixed. During selection, displacement is measured in the trainable adapter parameters relative to the current saved student.

The selector evaluates candidates on fixed correctness and retention probes for that iteration. It chooses the candidate with the largest positive correctness gain among those whose mean retention NLL does not increase and whose displacement stays within the calibrated radius. The selected transaction, including optimizer state, is committed. If every candidate fails these conditions, the original transaction is restored and training proceeds to the next receiver without a parameter change.

B Experimental Details

B.1 Dataset Inventory

Table 7 gives the fixed test pools shared by all methods and training seeds. We exclude tasks overlapping with training sources. APPS (Hendrycks et al., 2021), CodeContests (Li et al., 2022), and TACO (Li et al., 2023) provide training and test splits; LiveCodeBench `release_v6` (Jain et al., 2025) supplies a separate evaluation pool. Macro means weight the four benchmarks equally.

Exclusion sampling. The excluded tasks are randomly sampled from boundary tasks identified during initial-policy screening (Section 3.2). Selection requires verified failure and success within the screening budget; tasks whose screened responses all pass or all fail are ineligible.

Roles of the task pools. Source tasks provide the verified failure–success pairs used to build repair cards. Receiver tasks provide student rollouts for distillation. Correctness and retention probes supply training-time reference losses for measuring and selecting updates. The execution pools in the utility and carrier diagnostics assess generated-program behavior independently of these selection probes. Final benchmark tasks evaluate the trained student with task-only prompts and remain outside the training operations. These roles separate training feedback from evaluation on held-out tasks.

B.2 Baseline Adaptations

The comparisons retain each baseline's supervision source while sharing permissible training data. We organize the adaptations by the information that trains the student and the role of teacher context.

Direct supervision. Verified SFT learns from passing reference programs, and DPO learns from verified preference pairs. They test whether direct access to successful programs or success–failure

Table 7: **Evaluation inventory.** Recorded eligible task counts define the denominators for the main paired comparisons. Exclusions are listed relative to each original test pool. The four benchmark accuracies receive equal weight in the macro mean.

Benchmark	Raw test pool	Excluded tasks	Evaluation tasks
APPS	5,000	96	4,904
CodeContests	165	16	149
TACO	1,000	96	904
LiveCodeBench	1,055	96	959
Total	**7,220**	**304**	**6,916**

preferences accounts for the improvement. Vanilla evaluates the initial backbone through the same task-only interface on the shared benchmark task pools.

Context-conditioned distillation. OPCD supplies experience-conditioned teacher distributions on student prefixes. MOPD retains same-task rollout contexts, whereas PromptSD (code) learns soft prompts for algorithmic task groups. These comparisons vary how useful information enters the teacher while evaluating the resulting students on task-only inputs.

Objective adaptation. EOPD† uses `BREAD`'s routed teacher context. The dagger identifies this shared-context adaptation, allowing objective comparisons within the same experience interface.

Training-rule controls. The training-rule controls distinguish access to probe information from candidate selection. Probe replay incorporates probe solutions into training; probe objectives add correctness and retention NLL losses; meta-reweighting uses learner feedback to weight supervision. Norm-matched OPCD follows an update-count and displacement schedule fixed from training traces. The controls compare solution quality under shared training settings, experiences, and probes.

B.3 Evaluation and Aggregation

Main, component, and carrier results report arithmetic means over training seeds 42, 123, and 97011. For each run, task-only greedy accuracy is computed separately on each benchmark. In the main result tables, the four displayed benchmark means receive equal weight in the Mean column. The training-rule comparison in Table 4 reports one training seed. Table 7 lists fixed task counts.

Let $a_{s,b}$ denote percentage accuracy for seed s and benchmark b, and let round_2 denote half-up rounding to two decimal places. With $\mathcal{S}_{\mathrm{seed}} = \{42, 123, 97011\}$, the displayed benchmark cell d_b and the equally weighted Mean column m are computed as

$$d_b = \mathrm{round}_2\left(\frac{1}{3}\sum_{s\in\mathcal{S}_{\mathrm{seed}}} a_{s,b}\right), \quad m = \mathrm{round}_2\left(\frac{1}{4}\sum_{b=1}^{4} d_b\right). \tag{11}$$

Thus each benchmark contributes equally despite the different test-pool sizes. The four-family summary averages `Qwen`, `Mistral`, `Phi`, and `Gemma` with equal weight and selects each backbone's strongest baseline separately by reported Mean. Summary gains use these reported Means.

Paired comparisons match `BREAD` and its reference on the same tasks within each run, then average gained and lost counts across runs. All main paired outcomes are resolved. Component, carrier, and training-rule analyses compare conditions within their respective diagnostic settings; Appendix C.4 defines the component variants. The utility diagnostic groups candidate updates by NLL utility and reports execution-change bounds for unresolved outcomes.

B.4 Execution Outcomes and Paired Comparisons

Execution produces a verified success, a verified failure, or an unresolved outcome. With s verified successes and u unresolved judgments among N tasks, the accuracy interval is $[100s/N, 100(s+u)/N]$. Macro averaging gives the four benchmarks equal weight. An interval collapses to a point

when every judgment is resolved, as in the main paired comparisons, while the utility-bin intervals retain uncertainty for candidates with unresolved outcomes.

For each benchmark and run, gained solves change from reference failure to **BREAD** success, and lost solves change in the opposite direction. Their difference divided by benchmark size gives the paired accuracy change. Table 10 averages gained and lost counts across runs after pooling benchmarks. Its Mean column averages the four benchmark accuracy changes equally, then averages across runs. Paired Mean uses unrounded outcomes and can differ by 0.01 points from differences between displayed Means in Table 1 because the displayed values are rounded.

Writing these counts as $g_{s,b}$ and $\ell_{s,b}$ for a benchmark with N_b tasks gives $\Delta_{s,b} = 100(g_{s,b} - \ell_{s,b})/N_b$. The count columns summarize how many tasks change outcome, while the macro difference summarizes their benchmark-balanced accuracy effect. A positive net change can coexist with lost solutions, which is why both directions are reported.

B.5 Sampling and Training-Rule Diagnostics

The training-rule diagnostic uses one greedy response and eight sampled responses per task. For resolved sample outcomes $v_k = V(x, y^{(k)})$, its per-task sampled metrics are

$$\widehat{p}_1(x) = \frac{1}{8}\sum_{k=1}^{8} v_k, \qquad \widehat{o}_8(x) = \mathbf{1}\left\{\sum_{k=1}^{8} v_k > 0\right\}. \tag{12}$$

Pass@1 measures average success under sampling, and oracle@8 measures whether the eight responses contain a solution. Greedy accuracy evaluates the separate deterministic response. An unresolved sample contributes uncertainty to pass@1. Oracle@8 is known positive after any verified success, and remains unresolved when no sample passes and at least one judgment is unknown.

Table 4 compares solution quality under the same training settings using one common seed. Methods share the experience bank, router, and probes. Total computation depends on candidate updates and probe evaluations; the diagnostic has no common compute cap and reports no end-to-end timing.

C Additional Results

C.1 Performance on a Compact Backbone

`Phi-4-mini-instruct` is a 3.8B dense decoder-only model (Microsoft, 2025), extending the comparison to a smaller parameter scale. It uses a model-specific experience bank, four correctness and four retention probes per selection step, and a routing limit of one. Training uses 32 iterations, LoRA rank 8 with scaling 16, and the two learning rates in Table 5. We report mean task-only greedy accuracy over three independent training runs with seeds 42, 123, and 97011.

Table 8: **Compact backbone: `Phi-4-mini-instruct` (3.8B).** Three-seed mean greedy accuracy (%); Mean weights benchmarks equally. Subscripts give changes from displayed vanilla scores (pp). Best and second-best scores are bold and underlined, respectively.

Method	APPS	CodeContests	TACO	LiveCodeBench	Mean ↑
Vanilla	14.05	4.70	4.20	24.92	11.97
Verified SFT	16.95$_{\uparrow 2.90}$	5.37$_{\uparrow 0.67}$	3.76$_{\downarrow 0.44}$	28.61$_{\uparrow 3.69}$	13.67$_{\uparrow 1.70}$
DPO	16.41$_{\uparrow 2.36}$	6.04$_{\uparrow 1.34}$	5.94$_{\uparrow 1.74}$	33.92$_{\uparrow 9.00}$	15.58$_{\uparrow 3.61}$
OPCD	15.67$_{\uparrow 1.62}$	6.49$_{\uparrow 1.79}$	6.64$_{\uparrow 2.44}$	32.33$_{\uparrow 7.41}$	15.28$_{\uparrow 3.31}$
MOPD	17.51$_{\uparrow 3.46}$	6.94$_{\uparrow 2.24}$	<u>7.30</u>$_{\uparrow 3.10}$	<u>36.53</u>$_{\uparrow 11.61}$	<u>17.07</u>$_{\uparrow 5.10}$
PromptSD (code)	<u>19.09</u>$_{\uparrow 5.04}$	5.15$_{\uparrow 0.45}$	5.90$_{\uparrow 1.70}$	35.94$_{\uparrow 11.02}$	16.52$_{\uparrow 4.55}$
EOPD†	16.67$_{\uparrow 2.62}$	<u>7.38</u>$_{\uparrow 2.68}$	7.04$_{\uparrow 2.84}$	34.86$_{\uparrow 9.94}$	16.49$_{\uparrow 4.52}$
BREAD (Ours)	**19.93**$_{\uparrow 5.88}$	**7.61**$_{\uparrow 2.91}$	**9.29**$_{\uparrow 5.09}$	**37.92**$_{\uparrow 13.00}$	**18.69**$_{\uparrow 6.72}$

Table 8 extends the mean-accuracy improvement to the smaller backbone. **BREAD** reaches 18.69%, compared with 11.97% for vanilla and 17.07% for MOPD, the strongest baseline by macro mean. It

leads on all four benchmarks. The largest gain over vanilla is on LiveCodeBench, where task-only greedy accuracy increases from 24.92% to 37.92%.

C.2 Performance on a Mixture-of-Experts Backbone

`Gemma 4 26B-A4B-IT` (Google DeepMind, 2026) extends the comparison to a mixture-of-experts backbone with approximately 26B total parameters and 4B active per token. Table 9 reports the same eight methods and four benchmarks, averaging independent training runs with seeds 42, 123, and 97011. Evaluation uses the no-thinking protocol in Appendix A.1.

Table 9: **Mixture-of-experts backbone: `Gemma 4 26B-A4B-IT`.** Three-seed mean greedy accuracy (%); Mean weights benchmarks equally. Subscripts give changes from displayed vanilla scores (pp). Best and second-best scores are bold and underlined, respectively.

Method	APPS	CodeContests	TACO	LiveCodeBench	Mean ↑
Vanilla	54.05	30.20	19.21	65.55	42.25
Verified SFT	55.95$_{\uparrow 1.90}$	31.77$_{\uparrow 1.57}$	23.67$_{\uparrow 4.46}$	67.36$_{\uparrow 1.81}$	44.69$_{\uparrow 2.44}$
DPO	53.75$_{\downarrow 0.30}$	31.99$_{\uparrow 1.79}$	21.46$_{\uparrow 2.25}$	69.20$_{\uparrow 3.65}$	44.10$_{\uparrow 1.85}$
OPCD	57.69$_{\uparrow 3.64}$	37.36$_{\uparrow 7.16}$	24.12$_{\uparrow 4.91}$	66.49$_{\uparrow 0.94}$	46.42$_{\uparrow 4.17}$
MOPD	56.72$_{\uparrow 2.67}$	36.69$_{\uparrow 6.49}$	<u>24.71</u>$_{\uparrow 5.50}$	<u>70.18</u>$_{\uparrow 4.63}$	47.08$_{\uparrow 4.83}$
PromptSD (code)	58.56$_{\uparrow 4.51}$	38.70$_{\uparrow 8.50}$	23.60$_{\uparrow 4.39}$	67.19$_{\uparrow 1.64}$	47.01$_{\uparrow 4.76}$
EOPD†	**59.92**$_{\uparrow 5.87}$	<u>39.37</u>$_{\uparrow 9.17}$	20.98$_{\uparrow 1.77}$	68.96$_{\uparrow 3.41}$	<u>47.31</u>$_{\uparrow 5.06}$
`BREAD` (Ours)	<u>58.83</u>$_{\uparrow 4.78}$	**41.16**$_{\uparrow 10.96}$	**25.96**$_{\uparrow 6.75}$	**71.05**$_{\uparrow 5.50}$	**49.25**$_{\uparrow 7.00}$

`BREAD` reaches 49.25% mean accuracy on `Gemma`, improving over vanilla by 7.00 points and over EOPD†, the strongest baseline by macro mean, by 1.94 points. It leads on CodeContests, TACO, and LiveCodeBench, while EOPD† leads on APPS. The largest gain over vanilla is 10.96 points on CodeContests. Together with the dense-backbone results, this comparison extends the observed cross-task transfer gains to sparse expert activation.

C.3 Paired Performance

Table 10 gives the three-run mean counts behind Figure 2, using the same 6,916 tasks per run. Gained solves exceed lost solves for both backbones under either reference. Relative to vanilla, the mean net gain is 690 solves for `Qwen` and 480 for `Mistral`; macro means weight benchmarks equally.

Table 10: **Paired solution changes.** Gained and Lost are counts pooled across benchmarks and averaged over three runs on the same 6,916 tasks. Mean is the equally weighted benchmark accuracy difference (pp), averaged across the three independent training runs.

Backbone	Reference	Gained	Lost	Mean ↑
`Qwen3.5-9B`	Vanilla	777	87	+10.38
`Qwen3.5-9B`	OPCD	361	112	+3.02
`Mistral Small 3.2`	Vanilla	553	73	+7.41
`Mistral Small 3.2`	OPCD	424	90	+5.35

C.4 Component Ablation Design

Table 11 maps each component variant to the design choice it changes. The first two variants examine the source and content of repair experience; the remaining variants examine routing and update selection. This separates repair content from its measured effect on the student.

The component comparison changes the experience source, extraction input, router, or selector while retaining the main evaluation settings. Its complete-method reference is the same three-seed mean of

Table 11: **Definition of the component ablations.** Each row changes the indicated factor within the shared component-evaluation setting. Scores and changes from full BREAD appear in Table 2.

Variant	Change from full BREAD	Design choice examined
Success pool	Use matched initially solved source tasks	Boundary failures as experience sources
Positive only	Extract from verified successes alone	Failure–success contrast in extraction
NN routing	Route by semantic similarity	Parameter utility as a routing signal
No selection	Apply ordinary same-context OPD	Testing candidate updates before acceptance
No retention	Remove the retention constraint	Controlling retention-probe loss
First-order	Approximate candidate ranking; retain measured feasibility	Ranking actual optimizer trials

39.82% used in Table 1. The training-rule diagnostic instead reports a single training seed, for which BREAD reaches 33.41%. Comparisons use the corresponding reference within each setting.

C.5 Carrier Diagnostics

Figure 6 compares two uses of experience on independent diagnostic tasks. Context benefit tests the conditioned policy, while parameter transfer tests the task-only student after a candidate update. Each seed uses 512 trials in each pool, and results average the three seeds. Utility routing raises parameter transfer from 1.56 to 4.69 points for text and from 2.34 to 6.25 points for displacements. Initial slots and random displacements give parameter changes of -0.39 and -0.98 points. Use rate measures nonempty routes; update acceptance is a separate selector decision. Latent conditions share the text-derived utility router across representations.

The latent carriers distinguish optimized slots $\mathbf{z}^\star$ from their displacement $\Delta\mathbf{z} = \mathbf{z}^\star - \mathbf{z}_0$. Absolute slots provide the strongest context benefit in this comparison, while displacement gives the largest parameter-transfer gain. The two evaluation pools therefore answer different questions: how a carrier guides the conditioned policy, and how the resulting update changes the task-only student. This ranking difference motivates measuring the learner's update when choosing transferable experience.

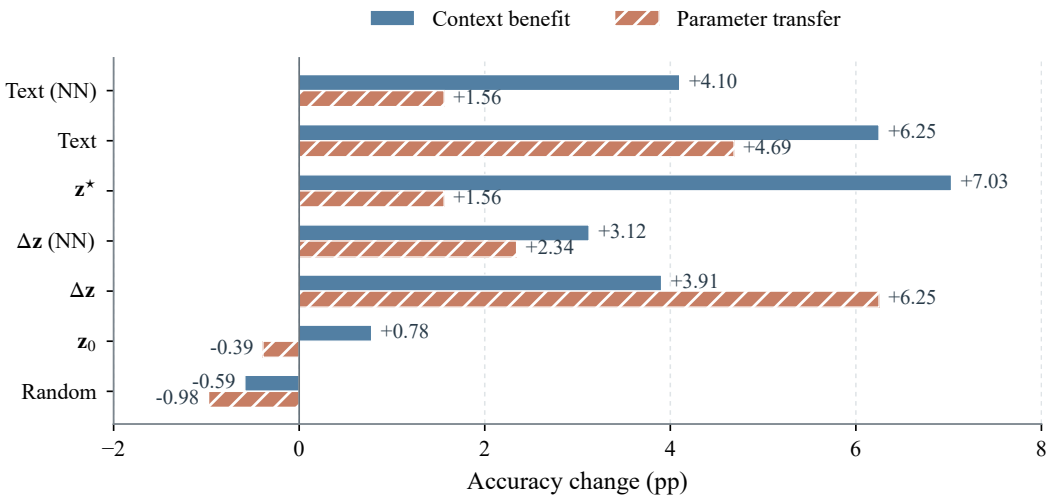


Figure 6: **Context benefit versus parameter transfer.** Bars show three-seed mean accuracy changes in independent diagnostic pools, each with 512 trials per seed. NN denotes nearest-neighbor routing; other transferable carriers share the utility router. All routes are nonempty. Initial and random carriers serve as controls for the experience representations.

C.6 Utility Diagnostic

Figure 4 evaluates 30 candidate updates from each training seed, totaling 90 updates. NLL utility is measured on training probes; execution transfer is measured on an independent task pool. The five utility bins summarize the change in task-only success. Assigning unresolved judgments to failure or success gives lower and upper bounds, which increase across bins and cross zero between negative and nonnegative utility groups. These bounds use the unresolved-outcome convention in Appendix B.4 and describe execution uncertainty rather than variation across seeds.

The two negative-utility bins lie entirely below zero, and the three nonnegative bins lie entirely above zero, including their unresolved-outcome bounds. Thus the sign agreement is visible at the bin level even after accounting for unresolved evaluations. The diagnostic connects the training-time NLL proxy to independently measured execution behavior for the sampled candidate updates.

C.7 Training-Rule Quality

Figure 7 complements Table 4 with greedy reliability, sampled success, and eight-sample coverage. Appendix B.2 defines the controls, and Appendix B.5 specifies their sampling metrics.

`BREAD` leads all three metrics in this single-seed comparison. Greedy and sampled pass@1 describe reliability under different decoding rules, while oracle@8 describes solution availability within eight attempts. Their joint improvement indicates that the observed benefit includes both single-response performance and sampled coverage. The probe-replay and probe-loss controls further distinguish giving the learner probe information from using it to evaluate an update before acceptance.

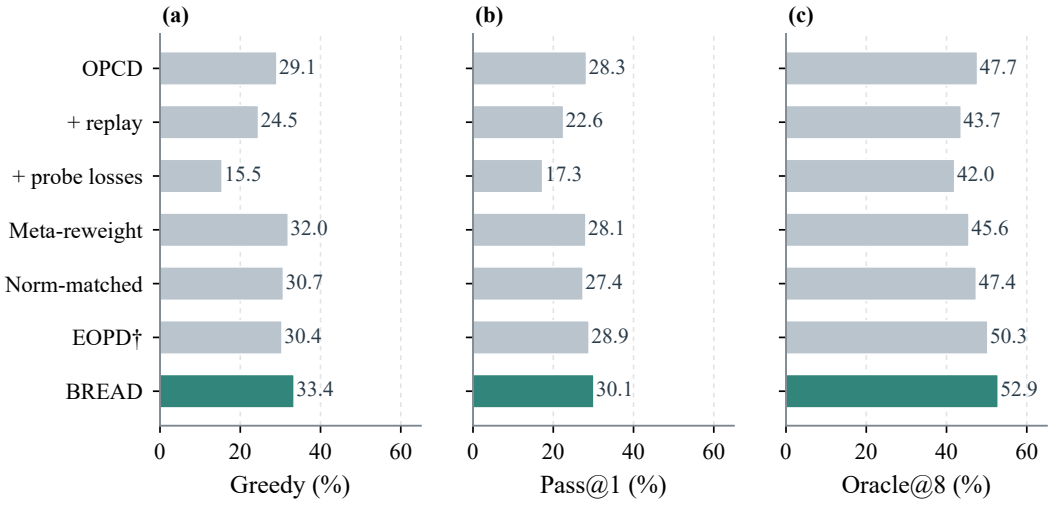


Figure 7: **Solution quality under alternative training rules.** Single-seed results compare greedy accuracy, sampled pass@1, and oracle@8 under shared training settings. `BREAD` leads each metric. OPCD shares teacher context; replay and probe-loss controls share training-probe information.

D WORKED REPAIR-TRANSFER EXAMPLE

Figure 5 gives a constructed example of transferring a state-update rule between two tasks.

Source: 0–1 knapsack. One item of weight 2 and value 3 gives an optimal value of 3 at capacity 4. From zeros, ascending in-place updates produce $[0, 0, 3, 3, 6]$: the update at capacity 4 reuses the item included at capacity 2. Descending updates produce $[0, 0, 3, 3, 3]$, using the item at most once.

Repair principle. Read predecessor values from the previous stage so each stage adds at most one item or edge. Enforce this rule with a suitable traversal order or a copy of the previous-stage states.

Receiver: a one-hop shortest path. Let $s \to a$ and $a \to t$ each cost 1, and $s \to t$ cost 5. With a one-edge budget, the correct answer is 5. Processing these edges in order with in-place updates chains $s \to a \to t$ and returns 2. Reading previous-layer distances prevents this chaining: a is initially unreachable, so only the direct edge reaches t, giving 5.

Connection to distillation. The repair card conditions the frozen teacher. Cross-task utility scores determine its routing to a receiver; AdamW trials from a shared snapshot select the update with the largest positive correctness-probe gain under retention and displacement constraints (Section 3).

E ALGORITHM WORKFLOW

Algorithm 1 assembles the stages from Section 3. Each temporary trial saves and restores the complete training state to evaluate utility and feasibility from a common starting point.

Algorithm 1: Training with `BREAD`

Input: Verified experience bank, receiver tasks, and correctness/retention probes; initial student, frozen teacher, trial rates, calibration tasks, and horizon N.
Output: Student π_{θ_N} generating programs from the task prompt alone.

/ 1. Measure cross-task parameter utility */*

Initialize ω_0; calibrate and freeze r — `// Reset after calibration`
Measure $u(x, e)$; restore state after each trial — `// Cross-task NLL, Eq. (5)`
Build $b_e^{(-h(x))}$ from cluster means — `// ≥ 2 clusters, Eq. (6)`
for $n = 0, \ldots, N-1$ **do**
  / 2. Route experience and form candidates */*
  Route experiences to receiver x by $b_e^{(-h(x))} > 0$ — `// Fixed limit`
  Sample $\mathcal{A}_n$ from $\pi_{\theta_n}(\cdot \mid x)$ — `// Task-only rollouts`
  For each routed e, compute $\mathbf{g}_{e,n} \leftarrow \nabla_\theta \mathcal{L}_{e,n}(\theta_n; \mathcal{A}_n)$ — `// Eq. (2)`
  Fix $\mathcal{Q}_n, \mathcal{K}_n$; $\omega_\varnothing \leftarrow \omega_n$, $\mathcal{T}_n \leftarrow \varnothing$ — `// Shared trial state`
  / 3. Test retention and select an update */*
  for $j = (e, \alpha) \in \mathcal{C}_n$, $\alpha \in \{\alpha_0, \alpha_0/2\}$ **do**
    $\omega_j \leftarrow \mathrm{AdamW}(\omega_n, \mathbf{g}_{e,n}, \alpha)$ — `// Same snapshot, Eq. (3)`
    Evaluate $\Delta_{\mathcal{Q}}(j), \Delta_{\mathcal{K}}(j)$ — `// Fixed probes, Eq. (8)`
    if $\Delta_{\mathcal{K}}(j) \le 0$ *and* $\|\theta_j - \theta_n\|_2 \le r$ **then** — `// Eq. (9)`
      Save ω_j; $\mathcal{T}_n \leftarrow \mathcal{T}_n \cup \{j\}$ — `// Feasible states`
    Restore ω_n — `// Full state`
  $j_n^\star \leftarrow \arg\max_{j \in \{\varnothing\} \cup \mathcal{T}_n} \Delta_{\mathcal{Q}}(j)$ — `// Null if no gain, Eq. (10)`
  $\omega_{n+1} \leftarrow \omega_{j_n^\star}$ — `// Commit complete state`
return *trained student* π_{θ_N}