FROM TRACES TO RECIPES: CONSTRUCTING AND DISTILLING REUSABLE PROCEDURAL KNOWLEDGE INTO LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reasoning distillation trains small students on teacher solution traces, but each trace entangles a reusable procedure with the details of one problem, leaving the student to infer what is shared. Existing supervision thus trades reusability against verification: traces solve a concrete problem but are tied to it, whereas abstractions generalize across problems but are never tested on the model that must use them. We introduce **DistilRecipe**, which represents procedural knowledge as **recipes**: instance-independent procedures that state when a method applies, which operations to perform in dependency order, and how to check the result. A recipe is admitted only when a small frozen student following it answers correctly and a monitor confirms it applied and was followed. In context, a receipe bank built with one 2B anchor improves 0.8B–4B Qwen3.5 models in all 30 settings, by an average of 9.0% on MATH-500 and 5.6% on GSM8K. Used as answer-free privileged context, recipes also distill into a `Qwen3.5-2B` student that needs no recipe at test time and outperforms matched chain-of-thought distillation in all four comparisons, by 2.08–3.20%.

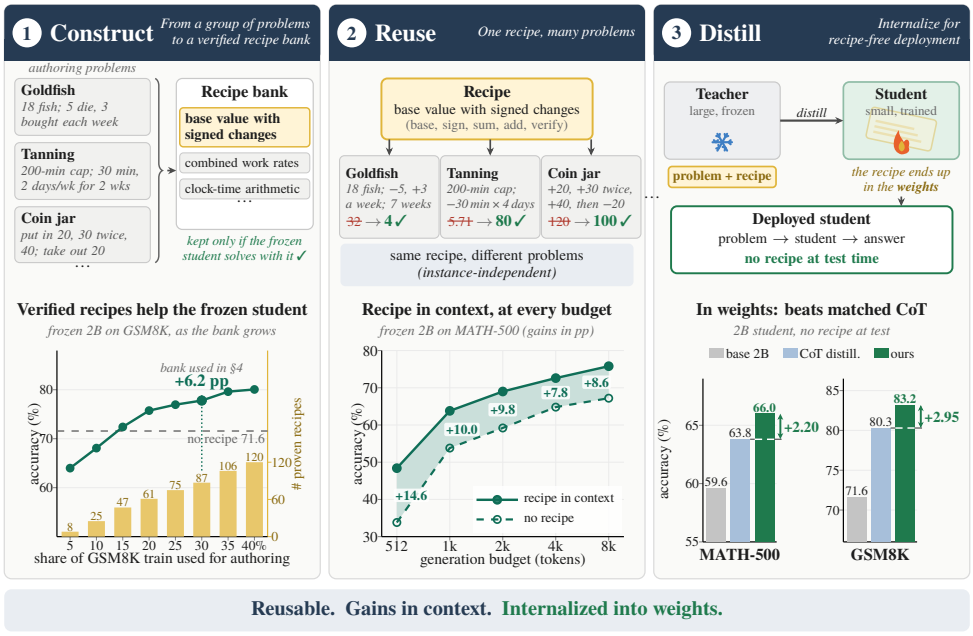


Figure 1: **Overview of DistilRecipe. (1) Construct:** problems are routed to a growing bank of instance-independent *recipes* (applicability, procedure, checks); a recipe is kept only if a frozen anchor student solves the problem with it. **(2) Reuse:** one recipe serves many related problems and, when recipe is put in the context, it helps at every generation budget. **(3) Distill:** a student that never sees the recipe beats matched CoT distillation.

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.

1 INTRODUCTION

Large language models (LLMs) have demonstrated strong capabilities in complex, multi-step reasoning (Wei et al., 2022), but deploying large models incurs substantial computational and memory costs. Reasoning distillation addresses this by transferring a strong teacher's problem-solving ability into a smaller student (Hsieh et al., 2023; Ho et al., 2023; Magister et al., 2023). Effective transfer requires the student to acquire methods that generalize beyond individual demonstrations, allowing it to solve new problems without access to the teacher at deployment.

A common approach trains students on teacher-generated chain-of-thought (CoT) traces (Ho et al., 2023; Magister et al., 2023). A trace, however, records one execution of a method on one problem, interleaving the reusable procedure with instance-specific entities, values, intermediate computations, and wording. Consider the three problems in Figure 1: a goldfish population, a capped tanning allowance, and a coin jar. Their stories share nothing, yet all three are solved by the same procedure: start from an initial quantity and apply signed changes in order. Three CoT traces present this procedure three times, each entangled with different surface details, and leave the student to infer what is shared. The sensitivity of LLMs to changes in numerical values that leave the underlying procedure intact (Mirzadeh et al., 2025) is consistent with this inference often failing, which motivates making the reusable procedural knowledge explicit in the distillation process.

Reusable reasoning abstractions, such as thought templates (Yang et al., 2024), behavior handbooks (Didolkar et al., 2025), and reasoning skills (Zhao et al., 2026a), show that reasoning knowledge can be represented above the level of a single trace. These abstractions, however, are distilled by a strong model from reasoning traces and are validated, if at all, by format and abstraction rules rather than by whether the model that will use them can actually carry them out. This gap matters in practice: reasoning skills work best for the model whose trajectories they were distilled from (Zhao et al., 2026a). For distillation, a procedure is worth transferring only if the student can execute it: one the student cannot follow to a correct answer carries no usable signal, however well it is written.

We introduce DistilRecipe, which admits a procedure only after a small frozen student has followed it to a correct answer. A *recipe* is an instance-independent specification of a solution method for a family of structurally related problems: it states when the method applies, which operations to perform in dependency order, and how to check the result, and it contains no instance-specific entities, values, worked solutions, or answers (Figure 2). Recipes accumulate in a *recipe bank* constructed online (Figure 3). A larger author model proposes or revises a recipe for an authoring problem, and a frozen small anchor student solves the problem with it. The execution counts as support only if the final answer is correct and an LLM monitor, which never sees the reference answer, confirms that the recipe applies and was followed. A revision is committed only if every previously accepted execution still passes, so the bank never regresses on its recorded evidence. Construction changes the procedures supplied to the student, never its parameters, and ends with a frozen recipe bank.

We use the frozen bank in two ways. In *recipe-conditioned inference* (RCI), a retrieved recipe is added to the prompt of a frozen recipient model, including recipients other than the anchor, which measures what the procedure alone adds. In *recipe distillation*, the recipe becomes privileged context: a teacher conditioned on the recipe supervises a student that never sees it, through either supervised fine-tuning (SFT) on correct recipe-guided solutions or on-policy distillation (OPD) along the student's own rollouts (Agarwal et al., 2024). Recipes are well suited to this role. On-policy self-distillation has typically used reference solutions as privileged context (Zhao et al., 2026b), which can let the student learn answer-dependent shortcuts it cannot reproduce at test time (Li et al., 2026). A recipe, by contrast, contains no answer and is selected from the question alone, so the advantage it gives the teacher is, in principle, available to a student that sees only the question.

Experiments with `Qwen3.5` models on GSM8K and MATH support this design. Recipe conditioning improves accuracy in all 30 combinations we evaluate (0.8B, 2B, and 4B recipients; generation budgets from 512 to 8,192 tokens; two benchmarks), by an average of 9.0% (absolute) on MATH-500 and 5.6% on GSM8K, although the bank was built with the 2B model alone. The gain depends on coverage: with a small bank, recipes lower accuracy, and they break even only once about 14–18% of the GSM8K training split has been used for authoring. This suggests that the benefit comes from matched procedures rather than from additional prompt text. In distillation, a `Qwen3.5-2B` student trained against a recipe-conditioned `Qwen3.5-9B` teacher outperforms its matched CoT-distilled counterpart in all four comparisons: by 3.20% and 2.08% under SFT and by 2.20% and

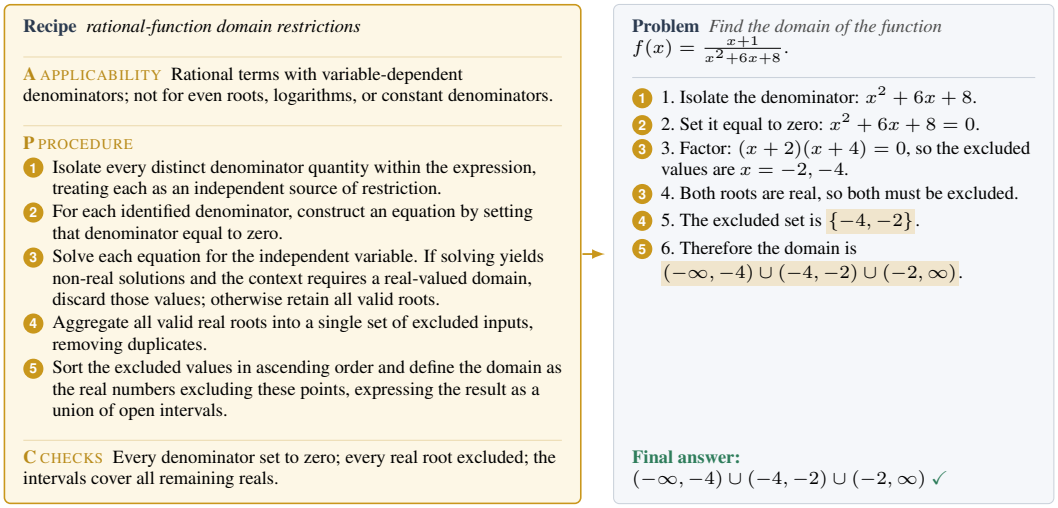


Figure 2: **A recipe and its execution.** Left: a recipe card from the MATH bank, with A and C shortened (full recipe card in Appendix E.4). Right: the frozen student's execution.

2.95% under OPD, on MATH-500 and GSM8K, respectively. The best student improves on its base model by 6.40% and 11.67% without ever seeing a recipe.

Our contributions are:

- **Recipes as an unit of procedural knowledge.** We propose instance-independent procedures for a problem family that state when a method applies, which operations to perform in dependency order, and how to check the result (§3.2)
- **A student-grounded construction.** An online construction loop that admits a recipe only when a small frozen student following it answers correctly and a monitor confirms that it applied and was followed, and that commits a revision only if every earlier accepted execution still passes (§3.3; Appendices A–B).
- **Evidence of utility and internalization.** A bank built with one 2B anchor improves frozen 0.8B–4B recipients at every budget tested (§4.2), and, used as privileged context, yields recipe-free students that beat matched CoT distillation in all four comparisons (§4.4).

2 Related Work

Distilling reasoning traces. Language models learn to reason by imitating rationales, either their own successful ones (Zelikman et al., 2022) or a larger teacher's (Ho et al., 2023; Magister et al., 2023; Hsieh et al., 2023). Later work supervises the states the student itself visits (Agarwal et al., 2024; Xu et al., 2025) or emphasizes decision-critical steps and semantic scaffolds (Dai et al., 2025; Wen et al., 2026). Throughout, the unit of transfer remains a worked solution to a single problem.

Reusable abstractions. Reasoning knowledge can also be stored above the trace level: as behaviors extracted from single solved problems (Didolkar et al., 2025), reasoning modules and thought templates (Zhou et al., 2024; Yang et al., 2024; 2025), test-time memories curated by the model itself (Suzgun et al., 2026; Ouyang et al., 2026), per-problem skill cards filtered by schema rules, whose consolidation into general skills is left open (Zhao et al., 2026a), and, closest to our recipe-conditioned inference, teacher-written guides for clusters of training questions, rated by an LLM judge (Alkiek et al., 2025). Such abstractions are rarely tested on the model that consumes them; exceptions are RLAD, which keeps an abstraction only if it raises a weaker solver's accuracy (Qu et al., 2026), and selective strategy retrieval, which selects strategies by their estimated executability for the target model (Liang et al., 2026).

Internalizing privileged context. Context distillation trains a model to retain context-induced behavior once the context is removed (Askell et al., 2021; Snell et al., 2022). Orca 2 applies it to strategy instructions through prompt erasure (Mitra et al., 2023), and fine-tuning has internalized behavior handbooks, thought templates, and cognitive behaviors (Didolkar et al., 2025; Yang et al., 2025; Sprague et al., 2026). On-policy variants minimize reverse KL to a context-conditioned teacher on the student's own rollouts; the context has been a reference solution (Zhao et al., 2026b), which risks answer-dependent shortcuts (Li et al., 2026), experiential knowledge merged into one context for all problems (OPCD; Ye et al., 2026), or retrieved skills whose signals are gated per rollout by the verifier (SGSD; Huang et al., 2026).

In our work, a recipe is a procedure for a whole problem family, accepted only when a frozen small student following it answers correctly and a monitor confirms that it applied and was followed; unlike RLAD's outcome filter, this test checks adherence and accumulates evidence across problems. For distillation we follow prompt erasure (SFT) and OPCD (on-policy); what is new is the context, which is verified on a student before training, routed from the question alone, and whose effect we isolate with a matched control that omits it.

3 METHOD

3.1 OVERVIEW

DistilRecipe separates the construction of procedural knowledge from the optimization of model parameters. Its input is an authoring set $\mathcal{D}_{\text{auth}} = \{(q_i, a_i)\}_{i=1}^{N}$, where q_i is a problem statement and a_i its reference answer. The pipeline has three stages (Figure 1). It first *constructs* a bank of recipes grounded in the executions of a small frozen student (§3.3), then *tests* the frozen bank by supplying recipes in context to recipient models whose weights stay fixed (§3.4), and finally *internalizes* recipe-guided behavior into the weights of a student that never sees a recipe (§3.5). Held-out evaluation problems are never used to construct recipes or to train students.

Construction involves five roles, each with deliberately restricted access to information (Table 3). The *anchor student* $S_{\bar{\theta}}$ is a small model whose parameters stay fixed and whose executions decide whether a recipe is accepted. The *router* R_ρ matches incoming problems to existing recipes, the *recipe author* W_ϕ writes and revises recipes, and the *monitor* M_ψ judges whether a recipe applies to a problem and whether an execution followed it. Finally, $E(y, a) \in \{0, 1\}$ is a deterministic evaluator of final-answer correctness. In our experiments, the router, author, and monitor are the same larger model queried with different prompts (Appendix B); we treat them as separate roles because they receive different information and play different parts in the loop.

3.2 RECIPES

A recipe is a structured, instance-independent procedure

$$r = (A, P, C). \tag{1}$$

The *applicability* field A states the observable structural conditions under which the method applies, together with explicit exclusions. The *procedure* $P = (p_1, \ldots, p_K)$ lists the required operations in dependency order over abstract roles such as quantities, rates, groups, and totals. The *checks* C specify mechanical tests for common execution failures, such as incompatible units, an omitted dependency, or a total that fails to reconstruct. A recipe contains no instance-specific entities or values, worked solutions, answers, or dataset labels (Appendix A.1). Figure 2 shows a card from the MATH bank. Its applicability field targets expressions whose denominators depend on the variable and excludes even roots, logarithms, and constant denominators. Its procedure isolates every denominator, sets each to zero, keeps the real roots, and writes the domain as a union of open intervals; its checks confirm that every denominator was used and that the intervals cover all remaining reals. The card names no specific function, root, or answer, so it applies to any problem with this structure; Figure 8 shows a single card reused on three GSM8K problems whose stories share nothing.

Given a problem q_i and a recipe r_i, the anchor student produces an execution

$$y_i = \text{Decode}\big[\, p_{\bar{\theta}}(\cdot \mid q_i, r_i) \,\big], \tag{2}$$

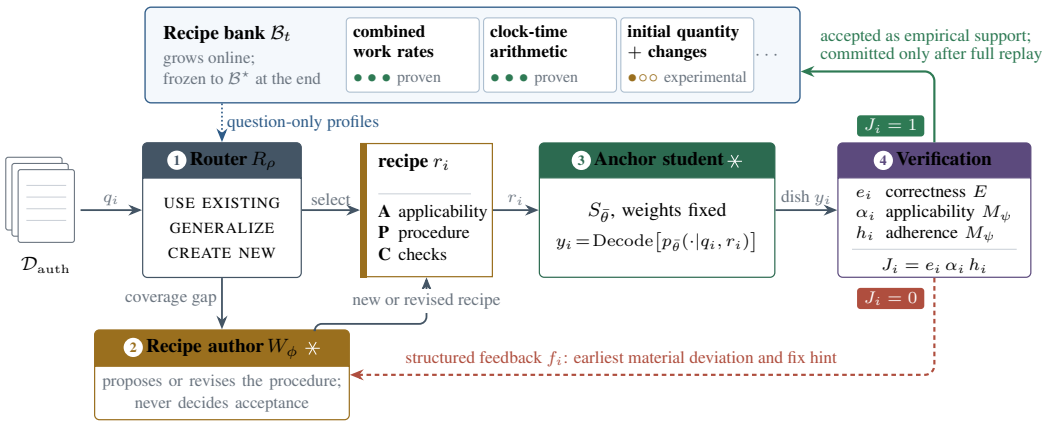


Figure 3: **Online recipe-bank construction.** For each authoring problem, the router (1) selects a recipe from question-only profiles or flags a coverage gap, and the author (2) then writes or revises one. The frozen anchor student (3) executes the recipe, and verification (4) accepts the execution only if the answer is correct and the recipe applied and was followed. Accepted executions add support, and revisions are committed only after all prior supports replay; rejections return feedback to the author. Only recipes change, never model weights.

which we call a *dish* (Figure 2). The recipe is the reusable procedure and the dish is its realization on one problem, so different dishes instantiate the same recipe with different entities, values, and intermediate computations.

3.3 Student-Grounded Bank Construction

The bank is built in a single online pass over $\mathcal{D}_{\text{auth}}$ (Figure 3; Algorithm 1 in Appendix A.3). Starting from an empty bank $\mathcal{B}_0 = \emptyset$, each authoring problem goes through routing, authoring, execution, and verification, and the bank $\mathcal{B}_t$ changes only when an execution is accepted.

Question-only routing. The router receives q_i and a shortlist of recipe profiles from $\mathcal{B}_t$, and chooses one of three actions: UseExisting when a recipe already has the required operations and dependencies, GeneralizeExisting when a recipe is structurally close and can be broadened without changing its method, and CreateNew when the shortlist has a genuine coverage gap. Profiles contain a truncated applicability field and procedure, and its support count (Appendix A.2). The router never sees reference answers, solution traces, correctness labels, or dataset categories, and it is instructed not to solve the problem (Appendix B.1).

Authoring. When a new or revised recipe is needed, the author W_ϕ receives the problem, the current recipe, if any, the failed execution and the monitor's feedback if available, and the profiles of neighboring recipes. It writes a candidate under the content constraints of §3.2. The author proposes a procedural hypothesis but has no say in its acceptance. The author is the only role that samples (temperature 0.7, top-p 0.95), so each attempt draws a fresh candidate; the router and monitor decode greedily, which keeps routing and acceptance decisions reproducible (Table 6).

Factorized verification. The anchor student executes the candidate recipe, and acceptance rests on three separate judgments. The evaluator checks the final answer, $e_i = E(y_i, a_i)$. The monitor, which observes (r_i, q_i, y_i) but never the reference answer or solution, judges applicability $\alpha_i \in \{0, 1\}$ and adherence $h_i \in \{0, 1\}$. An execution is accepted as support if and only if

$$J_i = e_i\,\alpha_i\,h_i = 1. \tag{3}$$

Keeping the three judgments separate prevents any one from standing in for another. A correct answer reached without following the recipe is no evidence for the recipe, and an incorrect answer reached by following an applicable recipe points to a flaw in the procedure itself. When $J_i = 0$, the monitor returns the earliest material deviation together with a fix hint, and the loop clarifies,

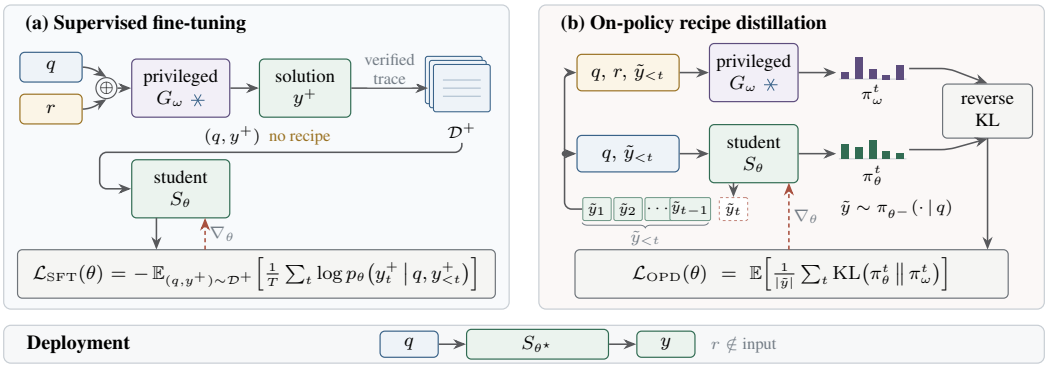


Figure 4: **Distilling recipes into weights.** A frozen privileged model G_ω, either the student's own base checkpoint or a larger teacher, sees the recipe; the student S_θ never does. (a) SFT: recipe-guided solutions from G_ω are kept only if their final answer is correct, and the student is trained on them without the recipe. (b) OPD: on the student's own rollouts, both models predict each next token, and the student minimizes reverse KL to G_ω. Dashed arrows are gradients; $\oplus$ is concatenation. At deployment, the student sees only q.

generalizes, reroutes, or replaces the candidate, up to $K = 3$ attempts per problem (failure modes in Appendix A.4).

Support accumulation and transactional revision. Each active recipe r keeps a support set $\Sigma(r) = \{i : J_i(r) = 1\}$ of the authoring problems on which its execution was accepted, so a recipe gathers evidence across distinct problems rather than only on the problem that created it. A recipe with fewer than three supports is labeled *experimental* and recipe with three or more supports as *proven*; the router may prefer a proven recipe when two are equally applicable. Revisions are transactional: a candidate r' that refines or generalizes r replaces it only if it is accepted both on the triggering problem i and on every inherited support,

$$\min_{j\in\Sigma(r)\cup\{i\}} J_j(r') = 1, \tag{4}$$

and consolidating several recipes applies the same test to the union of their support sets. Because the anchor student and the monitor decode greedily and the evaluator is deterministic, every judgment J_j is a fixed function of the recipe and the problem, so this rule ensures that the support set of a recipe lineage only grows: a committed revision never loses a problem its predecessor solved (Appendix A.5). When the pass ends, the bank is frozen as $\mathcal{B}^\star$, and no later stage modifies it.

3.4 Recipe-Conditioned Inference

RCI asks whether the frozen bank improves a model whose weights stay fixed. At test time, each held-out problem q is paired with a recipe by the same router used during construction, restricted to selection: it receives only q and the question-visible profiles of recipes in $\mathcal{B}^\star$, and it returns the existing recipe that best fits the problem (Appendix B.5). Because the bank is frozen, the router can no longer generalize or create recipes at this stage, and it never sees answers or dataset labels. Every problem receives the router's choice, even when no recipe applies well.

Recipes are plain text that contains no model-specific information, so the same bank can be given unchanged to recipient models other than the anchor. For a recipient S_m, we compare recipe-free and recipe-conditioned inference,

$$y^{(0)} = \mathrm{Decode}\big[p_m(\cdot \mid q)\big], \qquad y^{(r)} = \mathrm{Decode}\big[p_m(\cdot \mid q, r(q))\big], \tag{5}$$

under the same checkpoint, decoding policy, generation budget, and scorer; both prompts ask for a single concise solution path ending in a final-answer line, and the recipe-conditioned prompt adds the recipe and instructions for using it (Appendix B.3). Writing $\mathrm{Acc}_0^{(m)}$ and $\mathrm{Acc}_{\mathrm{RCI}}^{(m)}$ for the accuracies of the two conditions on the evaluation set, the recipe-conditioning effect is

$$\Delta_{\mathrm{RCI}}^{(m)} = \mathrm{Acc}_{\mathrm{RCI}}^{(m)} - \mathrm{Acc}_0^{(m)}. \tag{6}$$

A positive effect on the anchor shows that the bank helps the model it was grounded in, and a positive effect on other recipients shows that this help transfers, although we do not assume its size is the same across models. Neither the bank nor any recipient is updated during RCI.

3.5 Distilling Recipes into Weights

RCI requires retrieval and a recipe bank at deployment. We therefore ask whether its benefit can be transferred into the weights of a student that never observes a recipe. We cast this as distillation with privileged context (Figure 4): a frozen privileged model G_ω conditions on a problem q and a context c, whereas the student S_θ conditions on q alone. G_ω is either a frozen copy of the student's base checkpoint (self-distillation) or a larger model (teacher–student), and in both cases it is distinct from the recipe author.

Distillation objectives. We adopt two established objectives without modification. For SFT, we follow prompt erasure (Mitra et al., 2023): G_ω generates one solution per problem, incorrect solutions are discarded, and the student is trained by next-token likelihood on the remaining solutions with c removed. Since SFT supervises only fixed trajectories, we also use on-policy distillation (OPD) (Agarwal et al., 2024; Ye et al., 2026). In OPD, the student samples $\tilde{y} \sim p_{\theta^-}(\cdot \mid q)$ from a gradient-free copy of itself and minimizes the per-token reverse KL to the privileged model:

$$\mathcal{L}_{\mathrm{OPD}}(\theta) = \mathbb{E}_{q,\, \tilde{y} \sim p_{\theta^-}(\cdot|q)} \Big[\tfrac{1}{|\tilde{y}|} \textstyle\sum_t \mathrm{KL}\big(p_\theta(\cdot \mid q, \tilde{y}_{<t}) \,\big\|\, p_\omega(\cdot \mid q, c, \tilde{y}_{<t})\big) \Big]. \tag{7}$$

Implementation details are given in Appendices D.4–D.5. Neither objective depends on the content of c, so the contribution of this stage lies in the choice of context.

Recipes as privileged context. We set $c = r(q)$, the proven recipe that the test-time router (§3.4) selects for q from the question alone. Self-distillation with OPD thus reduces to on-policy self-distillation (Zhao et al., 2026b) with the reference solution replaced by a recipe. Prior work conditions the privileged model on a reference solution (Zhao et al., 2026b) or on a single experiential context shared by all problems (Ye et al., 2026). Unlike either, a recipe is shared by a problem family, contains no answer, and has been executed correctly by a frozen student before training (§3.3). The absence of an answer matters most. A reference solution gives the privileged model information that the student cannot recover from the question, which invites answer-dependent shortcuts (Li et al., 2026). A recipe adds only which procedure applies, and this is determined by the structure of the question. The recipe-conditioned model is therefore, in principle, a target that a question-only student can learn to match.

Matched recipe-free controls. Both objectives improve the student even without a recipe (CoT columns of Table 2). We therefore isolate the effect of the recipe by setting $c = \emptyset$ while holding all other components fixed. For teacher–student distillation, this yields CoT distillation from the same teacher. Under SFT, both students are trained only on problems for which both targets are correct, so they differ solely in the solutions they learn from. For self-distillation, setting $c = \emptyset$ makes the OPD loss vanish, so we compare Self-SFT against rejection-sampling fine-tuning (RFT) on the base model's own correct samples.

4 Experiments

We evaluate the stages of DistilRecipe by asking three questions. Do recipes help models whose weights stay fixed (§4.2)? Is that help due to a fitting procedure or merely to added text (§4.3)? And can it be distilled into a student that never sees a recipe (§4.4)?

4.1 Experimental Setup

Data. We build banks from a random 30% of the GSM8K training split (Cobbe et al., 2021) (2,242 problems) and from 1,000 random MATH training problems (Hendrycks et al., 2021), yielding 87 and 68 proven recipes (Appendix C). We evaluate on the GSM8K test set (1,319 problems) and on MATH-500. The authoring problems double as the distillation training pool; test problems are never used to build recipes or to train students.

Models. All models are from the `Qwen3.5` family. The 27B model serves as router, author, and monitor, and the 2B model is the frozen anchor (Table 6). RCI uses the 0.8B, 2B, and 4B models as recipients of the same bank. Distillation trains the 2B model, with a frozen copy of itself (self-distillation) or the 9B model (teacher–student) as the privileged model.

Evaluation. We report final-answer accuracy; all differences are absolute, in %. RCI solves each test problem with and without its recipe under greedy decoding at budgets from 512 to 8,192 tokens, using the prompts of Appendix B.3. Distilled students are evaluated without recipes at 2,048 tokens.

4.2 RQ1: Do Recipes Improve Frozen Models?

Recipes help frozen models at every budget, and not by saving tokens. The routed recipe improves accuracy in all 30 settings (Table 7), by 9.0% on MATH-500 and 5.6% on GSM8K on average. If recipes merely made short outputs more efficient, their benefit would vanish as the budget grows. Instead, it persists at 8,192 tokens on MATH-500 (Figure 5), and on GSM8K every recipient given a recipe at 512 tokens beats itself without one at 8,192. The recipe therefore changes what the model computes, not how compactly it writes.

The anchor gains most, as expected for recipes accepted on its own executions, but the help also reaches models the bank was never grounded in: the 0.8B and 4B models gain at every budget, and the gain is smallest only where accuracy is already near its ceiling (4B on GSM8K, +90%).

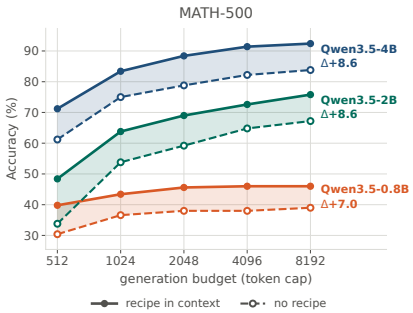


Figure 5: **Recipes help at every budget.** MATH-500 accuracy with the routed recipe (solid) and without it (dashed); Δ is the gain at 8,192 tokens.

4.3 RQ2: Is It the Procedure or the Added Text?

A recipe adds text to the prompt, and added guidance alone might help. If it did, any recipe would help, including one that does not fit the problem. Figure 6 tests this by growing the bank: since every problem receives the router's choice, a small bank forces most problems onto recipes built for other families. With 5% of the training split (8 recipes), accuracy falls below the recipe-free baseline, by 7.58% for the 2B recipient and 11.68% for the 0.8B recipient. As coverage grows, the sign flips: recipes break even at 14% (2B) and 18% (0.8B) of the split and add 8.49% and 7.81% at 40%.

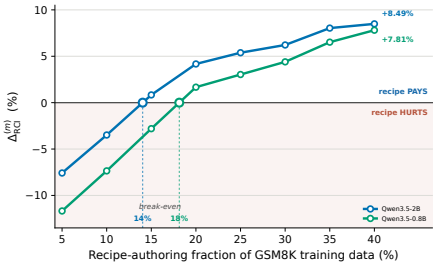


Figure 6: **Recipes help only when they match.** Gain over no recipe on GSM8K at 2,048 tokens as the bank grows.

The added text is present throughout; only its fit to the problem changes. Table 1 isolates fit directly: with the full bank, a recipe drawn uniformly at random lowers accuracy by 16.2–42.5% for every recipient and benchmark, while the routed recipe raises it by 6.2–9.4%. The weaker recipient is hurt more by a mismatch, consistent with it being less able to set aside a recipe that does not apply. Beyond break-even, the gains flatten (from $+3.3\%$ to $+0.5\%$ between consecutive snapshots for the 2B model) while the bank still grows by about four recipes per hundred problems, suggesting that later recipes serve a long tail of rarer families.

4.4 RQ3: Can Recipes Be Internalized?

Having shown that recipes help in context, we ask whether a student can absorb this advantage without ever seeing a recipe. Table 2 evaluates all students from the problem alone.

The clearest evidence comes from teacher–student distillation, where each DistilRecipe student has a matched CoT counterpart that differs only in whether the 9B teacher sees the recipe. The recipe wins all four comparisons: by 3.20% and 2.08% under SFT and by 2.20% and 2.95% under OPD

Table 1: **Fit, not text.** Accuracy (%) at 2,048 tokens with no recipe, a random recipe, or the routed recipe from the same bank; parentheses give the change from no recipe.

Benchmark	Recipient	No recipe	Random recipe	Routed recipe
GSM8K	`Qwen3.5-2B`	71.57	29.04 (–42.5)	77.79 (+6.2)
	`Qwen3.5-0.8B`	50.95	14.10 (–36.8)	57.16 (+6.2)
MATH-500	`Qwen3.5-2B`	59.60	36.00 (–23.6)	69.00 (+9.4)
	`Qwen3.5-0.8B`	38.00	21.80 (–16.2)	45.60 (+7.6)

Table 2: **Recipe-free accuracy after distillation.** `Qwen3.5-2B` student at 2,048 tokens. Each Recipe column is matched with the column to its left (same privileged model); Self-OPD has no control, since its loss would be zero.

		Self-distillation (2B → 2B)			Teacher–student (9B → 2B)			
		SFT		OPD	SFT		OPD	
	Base	RFT	Recipe	Recipe	CoT	Recipe	CoT	Recipe
MATH-500	59.60	61.20	64.00	57.00	60.40	63.60	63.80	**66.00**
GSM8K	71.57	72.36	73.16	74.45	75.89	77.97	80.29	**83.24**

on MATH-500 and GSM8K. Both comparisons are matched on problems: under OPD, the two students train on the same problems and their own rollouts, and under SFT, both train only on the problems for which both teachers produced a correct solution, so they differ only in the target solutions (Appendix D.4).

Self-distillation removes the stronger teacher, leaving the recipe text as the author's only contribution. Under SFT, this still pays off: Self-SFT outperforms rejection-sampling fine-tuning (RFT) on the model's own correct solutions by 2.80% on MATH-500 and 0.80% on GSM8K, even though RFT draws four samples per problem while Self-SFT keeps at most one correct solution. Both train on the same problems with targets from the same model, so this difference isolates the recipe, which accounts for almost two thirds of the Self-SFT gain on MATH-500 and half on GSM8K. Under OPD, the results split: Self-OPD helps on GSM8K ($+2.88\%$) but hurts on MATH-500 (-2.60%). Since both Self-SFT and on-policy transfer from the 9B teacher succeed on MATH-500, the failure lies in their combination: on-policy training supervises the student on its own prefixes, and once a rollout leaves the recipe's procedure, a 2B teacher is no better placed than the student to recover, which is more likely on harder problems. This agrees with reports that on-policy self-distillation can degrade reasoning (Kim et al., 2026).

5 Conclusion

We introduce DistilRecipe, which represents reasoning knowledge as instance-independent recipes and admits a recipe only after a small frozen student has followed it to a correct answer. In context, a bank built with one 2B anchor improves 0.8B–4B Qwen3.5 models in all 30 settings, by an average of 9.0% on MATH-500 and 5.6% on GSM8K. The gain requires a matching procedure, since a randomly drawn recipe lowers accuracy. Used as privileged context, recipes also distill into a student that never sees one and outperforms matched chain-of-thought distillation in all four comparisons, by 2.08–3.20%. Recipes help only when the bank covers the target problems: on GSM8K, they lower accuracy until about 14–18% of the training split has been used for authoring, so each new domain requires its own authoring budget. Our results challenge the practice of distilling reasoning one worked solution at a time, showing that a small bank of procedures the student can execute transfers more than matched traces from the same teacher.

AI Use Statement

Generative AI tools were used to assist with drafting and editing the manuscript, format LaTeX content, and support the design of scientific figures. All AI-assisted suggestions and outputs were re-

viewed by the authors for correctness, relevance, and originality. The authors take full responsibility for the final methodology, experiments, claims, and presentation.

REPRODUCIBILITY STATEMENT

We have taken several steps to make our results reproducible. All models are publicly available Qwen3.5 checkpoints and all datasets are public benchmarks; the authoring sets, evaluation splits, and bank-scaling snapshots are described in §4.1 and Appendix D.1. The complete construction procedure is given in Algorithm 1 (Appendix A.3), together with the recipe format and content constraints (Appendix A.1), the information available to each role and the deterministic shortlisting rule (Appendix A.2 and Table 3), and the verification criterion and answer-evaluation rules (Appendix A.4). The support-preservation property is stated with its assumption and a complete proof in Appendix A.5. The prompts of all roles are reproduced verbatim, with their version identifiers, in Appendix B. Decoding settings, software versions, and the random seed are listed in Appendix D.1 and Table 6, the evaluation protocol in Appendix D.2, and per-setting results in Table 7; the distillation data, matched baselines, and training hyperparameters are given in Appendices D.4 and D.5, and statistics of the frozen banks in Appendix C. The supplementary material includes the code for bank construction, freezing, and test-time routing (Algorithm 1 and Appendices A- B), together with the answer grader. Because the recipe author samples at non-zero temperature, rerunning construction yields a different bank. Upon acceptance, we will therefore release the two frozen banks used in all experiments, together with the code for bank construction, recipe-conditioned inference, distillation, and our answer grader.

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 *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/5be69a584901a26c521c2b51e40a4c20-Abstract-Conference.html.

Kenan Alkiek, David Jurgens, and V. G. Vinod Vydiswaran. Big reasoning with small models: Instruction retrieval at inference time. *arXiv preprint arXiv:2510.13935*, 2025.

Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Ben Mann, Nova DasSarma, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Jackson Kernion, Kamal Ndousse, Catherine Olsson, Dario Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, and Jared Kaplan. A general language assistant as a laboratory for alignment. *arXiv preprint arXiv:2112.00861*, 2021. URL https://arxiv.org/abs/2112.00861.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

Chengwei Dai, Kun Li, Wei Zhou, and Songlin Hu. Capture the key in reasoning to enhance CoT distillation generalization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 441–465, Vienna, Austria, July 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.21. URL https://aclanthology.org/2025.acl-long.21/.

Aniket Didolkar, Nicolas Ballas, Sanjeev Arora, and Anirudh Goyal. Metacognitive reuse: Turning recurring LLM reasoning into concise behaviors. *arXiv preprint arXiv:2509.13237*, 2025. URL https://arxiv.org/abs/2509.13237.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Advances in Neural Information Processing Systems*, volume 34, 2021. URL https://arxiv.org/abs/2103.03874.

Namgyu Ho, Laura Schmid, and Se-Young Yun. Large language models are reasoning teachers. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14852–14882, Toronto, Canada, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.830. URL https://aclanthology.org/2023.acl-long.830/.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 8003–8017, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.507. URL https://aclanthology.org/2023.findings-acl.507/.

Jiazhen Huang, Xiao Chen, Xiao Luo, Yong Dai, Senkang Hu, and Yuzhi Zhao. Skill-conditioned gated self-distillation for LLM reasoning. In *Conference on Empirical Methods in Natural Language Processing*, 2026. arXiv:2605.28791.

Jeonghye Kim, Xufang Luo, Minbeom Kim, Sangmook Lee, Dohyung Kim, Jiwon Jeon, Dongsheng Li, and Yuqing Yang. Why does self-distillation (sometimes) degrade the reasoning capability of LLMs? *arXiv preprint arXiv:2603.24472*, 2026. URL https://arxiv.org/abs/2603.24472.

Yunhe Li, Hao Shi, Wenhao Liu, Mengzhe Ruan, Hanxu Hou, Zhongxiang Dai, Shuang Qiu, and Linqi Song. DemoPSD: Disagreement-modulated policy self-distillation. *arXiv preprint arXiv:2607.02502*, 2026. URL https://arxiv.org/abs/2607.02502.

Weida Liang, Yiyou Sun, Shuyuan Nan, Chuang Li, Dawn Song, and Kenji Kawaguchi. Strategy executability in mathematical reasoning: Leveraging human–model differences for effective guidance. In *International Conference on Machine Learning*, 2026. arXiv:2602.22583.

Lucie Charlotte Magister, Jonathan Mallinson, Jakub Adamek, Eric Malmi, and Aliaksei Severyn. Teaching small language models to reason. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1773–1781, Toronto, Canada, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-short.151. URL https://aclanthology.org/2023.acl-short.151/.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. GSM-Symbolic: Understanding the limitations of mathematical reasoning in large language models. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/ec2e7a896f8250986b3907f57621ce94-Paper-Conference.pdf.

Arindam Mitra, Luciano Del Corro, Shweti Mahajan, Andres Codas, Clarisse Simoes, Sahaj Agarwal, Xuxi Chen, Anastasia Razdaibiedina, Erik Jones, Kriti Aggarwal, Hamid Palangi, Guoqing Zheng, Corby Rosset, Hamed Khanpour, and Ahmed Awadallah. Orca 2: Teaching small language models how to reason. *arXiv preprint arXiv:2311.11045*, 2023. URL https://arxiv.org/abs/2311.11045.

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 *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2509.25140.

Yuxiao Qu, Anikait Singh, Yoonho Lee, Amrith Setlur, Ruslan Salakhutdinov, Chelsea Finn, and Aviral Kumar. RLAD: Training LLMs to discover abstractions for solving reasoning problems. In *International Conference on Learning Representations*, 2026.

Charlie Snell, Dan Klein, and Ruiqi Zhong. Learning by distilling context. *arXiv preprint arXiv:2209.15189*, 2022. URL https://arxiv.org/abs/2209.15189.

Zayne Sprague, Jack Lu, Manya Wadhwa, Sedrick Keh, Mengye Ren, and Greg Durrett. SkillFactory: Self-distillation for learning cognitive behaviors. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/87cb65b5a16b8c87800881cbc337de33-Abstract-Conference.html.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106, Rabat, Morocco, 2026. Association for Computational Linguistics. URL https://aclanthology.org/2026.eacl-long.333/.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022. URL https://proceedings.neurips.cc/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Xiangyu Wen, Junhua Huang, Zeju Li, Min Li, Jianyuan Zhong, Zhijian Xu, Mingxuan Yuan, Yongxiang Huang, and Qiang Xu. Reasoning scaffolding: Distilling the flow of thought from LLMs. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=FcuJY1dK7s.

Wenda Xu, Rujun Han, Zifeng Wang, Long Le, Dhruv Madeka, Lei Li, William Yang Wang, Rishabh Agarwal, Chen-Yu Lee, and Tomas Pfister. Speculative knowledge distillation: Bridging the teacher-student gap through interleaved sampling. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/a2747a3844ca1e4667fbff3f558eb39b-Abstract-Conference.html.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E. Gonzalez, and Bin Cui. Buffer of thoughts: Thought-augmented reasoning with large language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-3607. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/cde328b7bf6358f5ebb91fe9c539745e-Abstract-Conference.html.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Minkai Xu, Joseph E. Gonzalez, Bin Cui, and Shuicheng Yan. SuperCorrect: Advancing small LLM reasoning with thought template distillation and self-correction. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/0967d7c8b171dd81b77c43067c02bebf-Paper-Conference.pdf.

Tianzhu Ye, Li Dong, Xun Wu, Shaohan Huang, and Furu Wei. On-policy context distillation for language models. *arXiv preprint arXiv:2602.12275*, 2026.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 15476–15488, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/639a9a172c044fbb64175b5fad42e9a5-Abstract-Conference.html.

Guangxiang Zhao, Qilong Shi, Xusen Xiao, Xiangzheng Zhang, Tong Yang, and Lin Sun. Thinking with reasoning skills: Fewer tokens, more accuracy. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pp. 2295–2308, San Diego, California, USA, July 2026a. Association for Computational Linguistics. doi: 10.18653/v1/2026.acl-industry.154. URL https://aclanthology.org/2026.acl-industry.154/.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. In *International Conference on Machine Learning*, 2026b. URL https://arxiv.org/abs/2601.18734.

Pei Zhou, Jay Pujara, Xiang Ren, Xinyun Chen, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, Denny Zhou, Swaroop Mishra, and Huaixiu Steven Zheng. SELF-DISCOVER: Large language models self-compose reasoning structures. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-4004. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/e41efb03e20ca3c231940a3c6917ef6f-Abstract-Conference.html.

A CONSTRUCTION DETAILS

This appendix gives the details needed to reproduce bank construction (§3.3): the recipe format (A.1), the information available to each role (A.2), the recipe lifecycle and the full algorithm (A.3), the verification criterion (A.4), and the support-preservation property (A.5).

A.1 RECIPE FORMAT AND CONTENT CONSTRAINTS

A recipe $r = (A, P, C)$ is serialized as a card with three fields, in this order.

Applicability (A). Observable structural conditions under which the method applies, together with explicit exclusions. The field identifies a problem family by the operations and dependencies its problems require, not by dataset labels, surface vocabulary, or story domain.

Procedure (P). The required operations in dependency order, written over abstract roles such as quantities, groups, counts, rates, states, and totals. Conditional branches are allowed when they broaden the recipe without changing its method. Step labels are written as words ("one", "two", ...) rather than digits.

Checks (C). Mechanical tests for common execution failures, such as inconsistent units, invalid ranges, omitted dependencies, incorrect operation order, or a known total that fails to reconstruct.

A card may not contain names, instance-specific values, reference answers, worked solutions, copied problem wording, hidden labels, or answer-format tricks. These constraints are part of the author prompt (Appendix B.2).

A.2 ROLES AND INFORMATION BOUNDARIES

Each role is a separate model call with deliberately restricted inputs (Table 3); decoding settings are in Table 6.

Table 3: **Information available to each role during construction.**

Role	Problem	Bank / recipe	Execution	Reference answer	Reference solution
Router	✓	profiles	–	–	–
Recipe author	✓	current and neighbors	failed only	–	–
Anchor student	✓	selected recipe	–	–	–
Monitor	✓	selected recipe	✓	–	–
Answer evaluator	–	–	final answer	✓	–

Router. The router receives the problem and a shortlist of $k = 12$ recipe profiles. To form the shortlist, every recipe in the bank is scored by the Jaccard overlap between the words of the problem and the words of the recipe's applicability field, procedure, and support cues, plus a small popularity prior $0.01 \log(1 + n_r)$, where n_r is the recipe's number of supports; ties are broken by recipe identifier, so the shortlist is deterministic. Words are lowercased alphabetic tokens of at least three characters, excluding 24 stopwords. The *support cues* of a recipe are the most frequent of these words in the questions of the problems it supports; they are computed from questions only, never from solutions or answers. Each profile shown to the router is compact: the recipe identifier, its applicability field truncated to 48 words, its procedure truncated to 32 words, its support count, and its 12 most frequent support cues. The router never sees a recipe's checks, its full text, reference answers, solution traces, correctness labels, or dataset categories. It is instructed neither to solve the problem nor to infer a hidden category, and it returns one action with a recipe identifier (Appendix B.1). At test time, the same router selects among existing recipes of the frozen bank only, from a shortlist formed in the same way (Appendix B.5).

Recipe author. Receives the problem, the current recipe if one exists, the failed execution and the monitor's feedback if available, the recipe's support cues, and the profiles of neighboring recipes. It is the only role that samples, so each attempt draws a fresh candidate.

Algorithm 1 Student-Grounded Online Recipe-Bank Construction

Require: Authoring set $\mathcal{D}_{\text{auth}}$; anchor student $S_{\bar{\theta}}$; recipe author W_ϕ; router R_ρ; monitor M_ψ; evaluator E; attempt budget K

$\mathcal{B} \leftarrow \emptyset$
for $(q_i, a_i) \in \mathcal{D}_{\text{auth}}$ **do**
  $\Pi_i \leftarrow$ question-visible profiles of recipes in $\mathcal{B}$ shortlisted for q_i
  $(u_i, r_i) \leftarrow R_\rho(q_i, \Pi_i)$ $\{u_i \in \{\text{USEEXISTING}, \text{GENERALIZEEXISTING}, \text{CREATENEW}\}\}$
  if $u_i \neq$ USEEXISTING **then**
    $r_i \leftarrow W_\phi(q_i, r_i, \Pi_i)$ {generalize r_i or write a new recipe}
  end if
  $k \leftarrow 0$; $accepted \leftarrow$ **false**
  repeat
    $k \leftarrow k + 1$
    $y_i \leftarrow \text{Decode}[p_{\bar{\theta}}(\cdot \mid q_i, r_i)]$
    $e_i \leftarrow E(y_i, a_i)$; $(\alpha_i, h_i, f_i) \leftarrow M_\psi(r_i, q_i, y_i)$
    if $e_i \alpha_i h_i = 1$ **then**
      if r_i is a new recipe **then**
        Add r_i to $\mathcal{B}$ with $\Sigma(r_i) \leftarrow \{i\}$; $accepted \leftarrow$ **true**
      else if $r_i \in \mathcal{B}$ **then**
        $\Sigma(r_i) \leftarrow \Sigma(r_i) \cup \{i\}$; $accepted \leftarrow$ **true**
      else if $J_j(r_i) = 1$ for all $j \in \Sigma(\text{parent}(r_i))$ **then**
        Replace $\text{parent}(r_i)$ by r_i with $\Sigma(r_i) \leftarrow \Sigma(\text{parent}(r_i)) \cup \{i\}$; $accepted \leftarrow$ **true**
      else
        $f_i \leftarrow$ replay failure {parent stays active}
      end if
    end if
    if not $accepted$ **then**
      Clarify, generalize, reroute, or replace r_i using f_i
    end if
  until $accepted$ **or** $k = K$
end for
$\mathcal{B}^\star \leftarrow \mathcal{B}$ {frozen; not modified by any later stage}
return $\mathcal{B}^\star$

Anchor student. Receives exactly one recipe and the problem, and solves the problem while making its main procedural steps visible (Appendix B.3).

Monitor. Receives the recipe, the problem, and the execution, but never the reference answer or solution. It returns an applicability judgment, an adherence verdict, the earliest failed step with a violation type and evidence, and a fix hint (Appendix B.4).

A.3 Recipe Lifecycle and Construction Algorithm

Lifecycle. A recipe fewer than three accepted support is *experimental*; with at least three, it is *proven*. When an experimental and a proven recipe are equally applicable, the router may prefer the proven one, but a precise experimental match remains a valid choice: support count never overrides procedural fit. The labels are operational states, not guarantees; "proven" means only that the recipe has passed the acceptance criterion on several construction problems.

Versioning. A refinement or generalization does not overwrite its parent until it is committed. A candidate r' revising an active recipe r, triggered by problem i, is committed only if

$$\min_{j \in \Sigma(r) \cup \{i\}} J_j(r') = 1,$$

where $\Sigma(r) = \{j : J_j(r) = 1\}$ is the support set of r. Consolidating several recipes applies the same test to the union of their support sets. A failed replay leaves the parent active and is returned to the author as feedback. Algorithm 1 gives the complete procedure.

A.4 FACTORIZED VERIFICATION

An execution y_i is accepted as support if and only if $J_i = e_i\,\alpha_i\,h_i = 1$, where $e_i = E(y_i, a_i)$ is final-answer correctness and α_i and h_i are the monitor's applicability and adherence judgments. Table 4 lists how each combination is interpreted. Keeping the three judgments separate matters because a correct answer is not evidence that the recipe applied or was used, and an incorrect but adherent execution points to a flaw in the procedure rather than in routing.

Table 4: **Interpretation of the verification outcome.** $*$ denotes either value.

e_i	α_i	h_i	Interpretation
1	1	1	accepted support
0	1	1	recipe applied and followed, but answer incorrect: flaw in the procedure
$*$	1	0	recipe applied but not followed: execution failure
$*$	0	$*$	recipe does not apply: routing or scope failure

Monitor rules. The monitor requires only the operations the current problem needs, not every optional operation in the recipe; accepts equivalent algebra, regrouping, or terminology when dependencies and units stay correct; ignores a scratch error that is visibly corrected; and marks a deviation at the earliest materially needed operation that is omitted, mis-scoped, mis-ordered, or executed incorrectly (Appendix B.4).

Answer evaluation. The evaluator extracts the answer following "`Final answer:`", falling back to the last numeric span when that line is missing, and normalizes it per benchmark. For GSM8K, it strips `\boxed{}`, currency symbols, and thousands separators and compares the values as exact decimals. For MATH, it tests symbolic equivalence with our grader `mathgrade`, which will be released with the code and which, for example, treats $\frac{1}{2}$, `1/2`, and `0.5` as equal. Both rules are deterministic and version-pinned. The evaluator is the only component that sees the reference answer; the router and the monitor never do.

A.5 SUPPORT PRESERVATION

Assumption 1 (Stationary verification). The anchor student's parameters, its greedy decoding, the deterministic evaluator, and the greedily decoded monitor are fixed throughout construction (Table 6). The router and author may change their decisions as the bank evolves; the assumption concerns only the components that decide whether an execution is accepted.

Proposition 1. Under Assumption 1, for any lineage of committed recipe versions $r^{(0)} \to r^{(1)} \to \cdots \to r^{(L)}$,

$$\Sigma(r^{(0)}) \subseteq \Sigma(r^{(1)}) \subseteq \cdots \subseteq \Sigma(r^{(L)}).$$

Proof. By the commit rule, $r^{(k+1)}$ is accepted on every $j \in \Sigma(r^{(k)})$. Under Assumption 1, each such judgment is a fixed function of the recipe and the problem, so every support of $r^{(k)}$ is a recorded support of $r^{(k+1)}$, giving $\Sigma(r^{(k)}) \subseteq \Sigma(r^{(k+1)})$. Induction over committed transitions yields the chain. □

Scope. The property guarantees non-regression on recorded supports only. It does not imply that a recipe is correct for every member of its family, that the supports represent the task distribution, or that the recipe generalizes to unseen problems or other recipients; these are evaluated empirically in §4.

B PROMPTS

This appendix reports the prompts of all roles, in the order in which the roles act. Each role is a separate model call that receives only the inputs listed in Table 3 (Appendix A.2); fields in angle brackets are filled at call time, and box colors follow the roles in Figure 3. Both banks were constructed with the prompt versions reported below.

B.1 Router (Construction)

The construction router uses `UNIFIED_ROUTER_PROMPT_V4`.

Recipe Router Prompt

You are the single Recipe Bank Router for an online recipe-optimization loop. Use only the NEW PROBLEM and the compact question-visible recipe profiles. Never solve the problem and never infer a hidden category.

Choose exactly one action:
- `USE_EXISTING` when one recipe already has the required operation and dependency structure;
- `GENERALIZE_EXISTING` when one recipe is structurally close and can be broadened without changing its mathematical method;
- `CREATE_NEW` only when the shortlist has a genuine procedural coverage gap.

Candidate recipes with one support are experimental. Proven recipes have at least three supports. Prefer a proven recipe when equally applicable, but do not reject a precise experimental match merely because it has fewer supports.

NEW PROBLEM
`<question>`

SHORTLISTED RECIPE PROFILES
`<profiles>`

Return one JSON object only:

```
{"action":"USE_EXISTING|GENERALIZE_EXISTING|CREATE_NEW",
"card_id":"card ID or null",
"reason":"brief visible structural reason",
"confidence":0.0}
```

Note. The lifecycle label included in each profile is computed with the three-support threshold of Appendix A.3; the prompt sentence above is reproduced as used.

B.2 Recipe Author

The recipe author uses `GENERATOR_PROMPT_V3`.

Recipe Author Prompt

You are the Recipe Generator for a frozen-weight student.
Write one method-family recipe: a reusable mathematical procedure that should transfer across different stories sharing the same operation and dependency structure. Abstract story nouns into quantities, groups, rates, states, or totals. Use the widest scope that preserves the required operation order. Do not restrict the number of objects, units, or story setting unless mathematically necessary. Use variables and conditional branches when they make the method more reusable.
The card must:
- contain `APPLICABILITY:`, `PROCEDURE:`, and `CHECKS:` exactly once and in that order;
- state observable structural cues, including when the recipe should not be used;
- give a dependency-ordered symbolic procedure and mechanical checks;
- contain no names, instance values, answers, copied wording, hidden labels, worked examples, or answer-format tricks;
- use words rather than digits for step labels.

NEW PROBLEM
`<question>`
CURRENT RECIPE OR NONE
`<current recipe>`
FAILED STUDENT DISH OR NONE
`<dish>`
MONITOR FEEDBACK OR COVERAGE GAP
`<feedback>`
SANITIZED SUPPORT CUES
`<support cues>`
NEAREST ACTIVE RECIPE PROFILES

<neighbor profiles>

Return one JSON header followed by one card block:
{"action":"CREATE|REFINE|GENERALIZE",
"reason":"brief method-family reason"}
<CARD>
APPLICABILITY: ...
PROCEDURE: ...
CHECKS: ...
</CARD>

B.3 Anchor Student and Recipe Recipients

The same template is used by the anchor student during construction and by every recipient in the recipe-conditioned condition of RCI.

Recipe-Conditioned Student Prompt

You are the student solver. Use the supplied recipe only when its APPLICABILITY conditions match the problem. Solve the problem yourself.
Do not copy or restate the recipe. Keep the reasoning concise, but make the main procedure steps visible so an auditor can verify that you followed them.

RECIPE CARD
<card>
PROBLEM
<question>

Produce one solution path. End with exactly one final line in the form
Final answer: <answer>
and write nothing after it.

The recipe-free condition of RCI uses the following prompt. Like the recipe-conditioned prompt, it asks for a single concise solution path ending in a final-answer line, but it contains no recipe and no instructions about one.

Recipe-Free Student Prompt (Baseline)

Solve the problem using one concise reasoning path. Show the necessary calculations.

PROBLEM:
{question}

End with exactly one final line in the form "Final answer: <answer>" and write nothing after it.

B.4 Procedural Monitor

The monitor uses MONITOR_PROMPT_V2.

Procedural Monitor Prompt

You are an independent recipe-procedure auditor. Determine whether the card substantively applies to the problem and whether the student's visible solution substantively followed it.
Do not judge whether the final answer is correct. Do not invent hidden reasoning.

Decision rules
- Require only operations needed by the current problem, not every optional operation in the recipe.
- Equivalent algebra, regrouping, terminology, or harmless multiplication by one is allowed when dependencies and units remain correct.
- A corrected scratch error is not a deviation when it is visibly superseded; an uncorrected material error remains a deviation even if the final answer happens to be correct.
- Mark deviated only for the earliest materially needed operation that is omitted, mis-scoped, misordered, or executed incorrectly.
- Use not_applicable only when no procedure in the card covers the problem.

RECIPE CARD
`<card>`
PROBLEM
`<question>`
STUDENT DISH
`<dish>`

Return one JSON object containing:
`applicability`, `verdict`, `first_failed_step`, `violation_type`, `evidence`, `fix_hint`, `card_artifact`, and `confidence`.

B.5 Router (Test Time)

At test time, the construction router is used in selection-only mode on the frozen bank $\mathcal{B}^\star$. Its prompt differs from Appendix B.1 in that the router must select one existing recipe; the options to generalize or create recipes are removed.

Test-Time Router Prompt

You are the single Recipe Bank Router for matching a recipe with the problem below . Use only the NEW PROBLEM and the compact question-visible recipe profiles. Never solve the problem and never infer a hidden category.

Select the one shortlisted recipe whose operation and dependency structure best matches the problem. You must choose an existing recipe.

NEW PROBLEM
`<question>`

SHORTLISTED RECIPE PROFILES
`<profiles>`

Return: `Card ID`

C Bank and Construction Statistics

This appendix describes the frozen banks and the construction runs that produced them: what the banks contain (C.1), and how the GSM8K bank grew (C.2).

C.1 Bank Contents

Table 5: **Frozen banks used in all experiments.** A recipe with fewer than three supports is experimental; with three or more, proven (Appendix A.3). Experimental recipes are discarded when the bank is frozen; coverage counts the problems supported during construction by any recipe.

	GSM8K	MATH
Authoring problems	2,242	1,000
Problems with an accepted support (coverage)	1600 (71.4%)	632 (63.2%)
Recipes in the frozen bank (proven, $\geq$3 supports)	87	68
Supports per proven recipe: mean / median / max	6.75/5/25	3.82/3/8

C.2 Growth of the GSM8K Bank

Figure 7 follows the GSM8K bank through one online construction pass. The number of proven recipes grows almost linearly with the number of authoring problems, from 8 after 374 problems (5% of the training split) to 87 at 30% and 120 at 40%. After the first snapshot, the bank gains about four proven recipes per hundred problems; the rate is lower at the start (2.1%), partly because a recipe becomes proven only after three supports, which takes longer while the bank is young. Accuracy follows a different course. With 8 recipes, the 2B recipient scores 63.99%, below its

recipe-free accuracy of 71.57%, because most test problems are routed to recipes that do not fit them. Accuracy then rises quickly, crossing the baseline near 14% and reaching 75.74% at 20%, after which each snapshot adds at most 1.8 points, ending at 80.06%. New recipes thus keep arriving at a steady rate while each adds less accuracy, which suggests that later recipes cover a long tail of rarer problem families. The 30% snapshot used in all other experiments (87 recipes, 77.79%) lies in this flattening regime, so our conclusions do not depend on a particular stopping point.

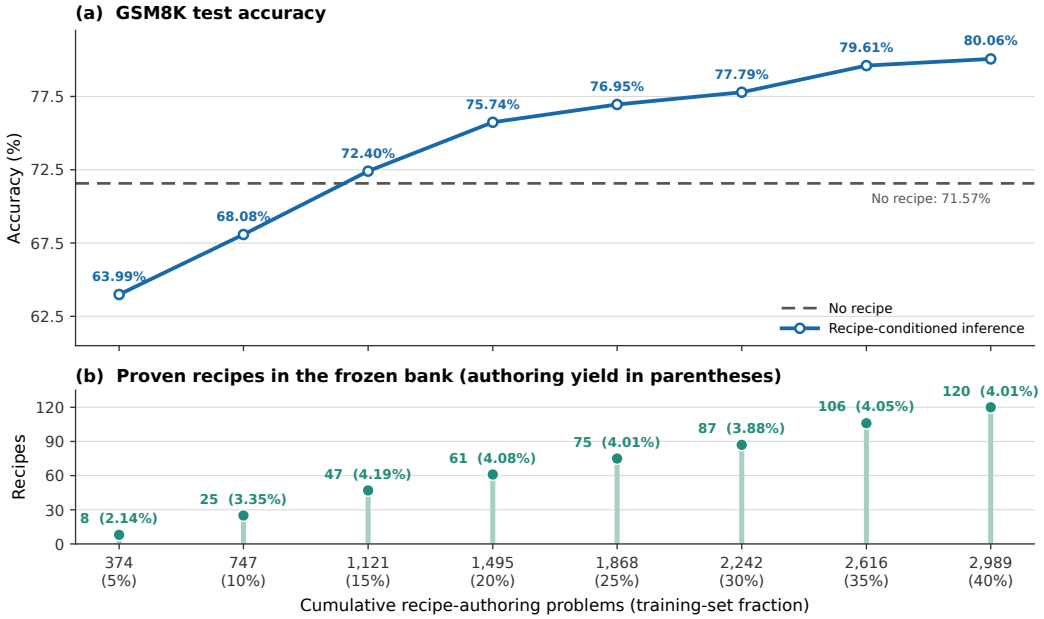


Figure 7: **Growth of the GSM8K bank.** (a) Test accuracy of the 2B recipient with routed recipes at eight snapshots of one construction pass; the dashed line is its recipe-free accuracy. (b) Proven recipes in the bank at each snapshot; percentages in parentheses are proven recipes per authoring problem. The 30% snapshot is the bank used in all other experiments.

D Experimental Details

D.1 Data, Models, and Decoding

Benchmarks. For GSM8K, construction uses 2,242 of the 7,473 official training problems (30%), and we evaluate on the official test split (1,319 problems). For MATH, we use all 1,000 problems of a pre-stratified authoring set and evaluate on MATH-500. The bank-scaling experiment (Figure 7) uses snapshots of the same GSM8K construction pass after 5%, 10%, . . . , 40% of the training split (374 to 2,989 problems); every other experiment uses the snapshot at 30%. Distillation trains on 2,473 GSM8K and 1000 MATH problems drawn from the same construction pool. No test problem is used to construct, revise, or verify recipes, or to train students.

Frozen banks. The frozen banks consist of the proven recipes: 87 for GSM8K and 68 for MATH (Appendix C). Experimental recipes are not used after construction.

Models. All models are from the `Qwen3.5` family and run with thinking disabled, using **HuggingFace Transformers 5.14.1 (PyTorch 2.13.0, CUDA 13.0)** in **bfloat16**. `Qwen3.5-27B` serves as router, recipe author, and monitor, and `Qwen3.5-2B` is the frozen anchor. RCI uses `Qwen3.5-0.8B`, `Qwen3.5-2B`, and `Qwen3.5-4B` as recipients. Distillation trains `Qwen3.5-2B`, with `Qwen3.5-9B` (teacher–student) or a frozen copy of the `Qwen3.5-2B` checkpoint (self-distillation) as the privileged model. Table 6 lists the decoding settings during construction. The anchor student additionally runs with batch size 1, seed 0, and no presence or repetition

penalty, and stops early if, after at least 256 tokens, a block of 16, 32, 64, or 128 tokens repeats three times.

Table 6: **Decoding settings during bank construction.** Only the author samples; every role that makes a decision decodes greedily.

Role	Model	Decoding	Max. new tokens
Anchor student	`Qwen3.5-2B`	greedy	2,048
Recipe author	`Qwen3.5-27B`	sampled (T=0.7, top-p=0.95)	768
Router	`Qwen3.5-27B`	greedy	128
Monitor	`Qwen3.5-27B`	greedy	256

D.2 Recipe-Conditioned Inference and Scoring

Protocol. The test-time router (Appendix B.5) selects one proven recipe for each test problem, and this choice is held fixed across all generation budgets and recipients. Both conditions use greedy decoding with maximum budgets of 512, 1,024, 2,048, 4,096, and 8,192 new tokens, the same stopping rule, including the loop detection above, and the same evaluator; they differ only in their prompts (Appendix B.3). The bank is not modified during evaluation.

Scoring. Accuracy is the fraction of test problems whose answer matches the reference under the evaluator of Appendix A.4. A response is scored from its final-answer line, or from its last numeric span if that line is missing, and counts as incorrect if it contains neither.

D.3 Per-Setting Results of Recipe-Conditioned Inference

Table 7 reports every recipient, benchmark, and generation budget; Figure 5 plots the MATH-500 half. The recipe improves accuracy in all 30 settings, by 9.00 points on MATH-500 and 5.62 on GSM8K on average; averaged over budgets, the gains are 7.76, 10.08, and 9.16 points on MATH-500 and 6.28, 6.65, and 3.91 on GSM8K for the 0.8B, 2B, and 4B recipients. Two patterns stand out. First, extra tokens and a recipe are not interchangeable: on GSM8K, every recipient given a recipe at 512 tokens outperforms itself without one at 8,192 tokens, although the larger budget is sixteen times longer, and on MATH-500 the 0.8B recipient does the same. Second, where more tokens help little, the recipe still helps: the 0.8B recipient's recipe-free accuracy on GSM8K stays between 50.04% and 50.95% across all budgets, while with the recipe it rises from 54.81% to 57.70%. The smallest gains occur for the 4B recipient on GSM8K (2.58 to 4.40 points), where its recipe-free accuracy already exceeds 90%.

D.4 Distillation

Training problems and recipes. Every distillation variant trains on all authoring problems: 2,242 for GSM8K and 1,000 for MATH. Each problem is paired with a proven recipe selected by the test-time router of Appendix B.5, which sees only the question and the compact recipe profiles. The recipe is therefore chosen without reference answers or construction outcomes, and the same pairing is used for SFT and OPD.

SFT targets. For each problem, the privileged model generates **one** solution conditioned on the problem and its recipe, with **greedy decoding** (no sampling, k=1, thinking disabled, cap 2048 tokens on GSM8K and 4096 on MATH). The recipe is then removed, and the student is trained on the problem and the solution alone. The matched CoT baseline uses the same privileged model, decoding, and correctness filter without the recipe. Each SFT pair, teacher–student and self-distillation alike, is trained on the intersection of its two filtered sets. For teacher–student, these are the problems where both the recipe-conditioned and the recipe-free solution are correct. For self-distillation, the RFT baseline samples four solutions per problem from the base 2B model without a recipe (ancestral sampling, temperature 0.8, top-$p = 0.95$), and the pair is trained on the problems where both the recipe-guided solution and at least one RFT sample are correct, with one target per problem on each side.

Table 7: **Recipe-conditioned inference in all 30 settings.** Accuracy (%) without a recipe (Acc_0) and with the routed recipe ($\mathrm{Acc}_{\mathrm{RCI}}$), and the absolute gain, for each recipient and generation budget. MATH-500 is plotted in Figure 5.

		MATH-500			GSM8K		
Recipient	Budget	Acc_0	$\mathrm{Acc}_{\mathrm{RCI}}$	Gain	Acc_0	$\mathrm{Acc}_{\mathrm{RCI}}$	Gain
0.8B	512	30.40	39.80	+9.40	50.04	54.81	+4.78
	1,024	36.60	43.40	+6.80	50.49	56.63	+6.14
	2,048	38.00	45.60	+7.60	50.95	57.16	+6.22
	4,096	38.00	46.00	+8.00	50.57	57.70	+7.13
	8,192	39.00	46.00	+7.00	50.57	57.70	+7.13
2B	512	34.00	48.60	+14.60	68.76	74.68	+5.91
	1,024	53.80	63.80	+10.00	69.60	76.72	+7.13
	2,048	59.60	69.00	+9.40	71.57	77.79	+6.22
	4,096	64.80	72.60	+7.80	71.89	78.85	+6.96
	8,192	67.20	75.80	+8.60	72.03	79.08	+7.05
4B	512	61.20	71.20	+10.00	90.90	93.48	+2.58
	1,024	75.00	83.40	+8.40	91.43	95.38	+3.94
	2,048	78.80	88.40	+9.60	91.66	95.98	+4.32
	4,096	82.20	91.40	+9.20	91.89	96.21	+4.32
	8,192	83.80	92.40	+8.60	91.89	96.29	+4.40
Mean gain				+9.00			+5.62

On-policy distillation. OPD generates no targets in advance. At every update, the student samples **one** rollout per problem from a copy of its current parameters that receives no gradient, using **ancestral sampling at temperature** 1.0 **with top-**$p = 1.0$ and a maximum length of **2048** new tokens. Rollouts are drawn for the whole accumulation window at once — 64 problems per update — and are discarded after that update rather than cached. The frozen privileged model conditions on the problem, its matched recipe, and the student's prefix, while the student conditions on the problem and the prefix. The loss is the per-token reverse KL divergence of equation 7, computed **exactly over the full vocabulary** in fp32, at temperature 1.0.

Matched baselines. Each DistilRecipe variant and its baseline share the privileged model, student initialization, training problems, optimization schedule, and objective, and differ only in whether the privileged model receives the recipe. Under SFT, the baseline is CoT for teacher–student distillation and RFT for self-distillation. Self-distillation under OPD has no recipe-free counterpart: without the recipe, the privileged model and the student coincide at initialization, and the loss is zero.

D.5 Training Details

Students are fully fine-tuned in bfloat16 with AdamW, weight decay 0, and learning rate 2×10^{-5}, using linear warmup over the first 10% of steps followed by cosine decay to zero, and gradient clipping at 1.0 for GSM8K and 10.0 for MATH. Each optimizer step accumulates gradients over 64 problems at micro-batch size 1 for OPD variants, and 16 problems at micro-batch size 1 for SFT variants, and every run lasts five epochs. Sequences are capped at 3,072 tokens (prompt and target). On-policy runs draw one rollout per problem per step by ancestral sampling (T=1.0, top-p=1.0, at most 2,048 new tokens) and minimize the full-vocabulary reverse KL divergence to the frozen privileged model, which is `Qwen3.5-9B` for teacher–student distillation and the frozen 2B checkpoint for self-distillation; no gradient flows through sampling. For each run, we select the checkpoint among the five epoch milestones by accuracy on 300 held-out training problems not used for authoring, and report its test accuracy. Students are evaluated greedily at 2,048 tokens on the GSM8K test split and on MATH-500. All experiments run on three NVIDIA RTX 6000 Ada GPUs (48 GB).

E QUALITATIVE EXAMPLES

This appendix shows how a recipe changes the execution of a fixed student. Each example gives the problem, the student's recipe-free output, the matched recipe, and the recipe-conditioned output. Recipe-free outputs use the baseline prompt of Appendix B.3, and recipe-conditioned outputs use the student prompt of Appendix B.3. Outputs keep their original wording; only the mathematics is typeset for readability.

E.1 A RUNNING EXAMPLE: ONE RECIPE, THREE PROBLEMS

Figure 8 follows the initial-quantity recipe, the example used in Figure 1, across three GSM8K problems whose stories share nothing. Without the recipe, the student fails each problem in a different way; with the recipe, given unchanged, it solves all three.

E.2 GSM8K: REPEATING ACTIVE INTERVALS

The recipe-free student confuses the number of active occurrences in a cycle with the length of the cycle.

Problem

Emma buys 2 containers of milk every school day for lunch. She does not go to school on the weekends. How many containers of milk does she buy in 3 weeks?

Recipe-free output (incorrect)

Emma buys 2 containers per school day. Since she does not go to school on weekends, she buys 2 containers per week. Over 3 weeks, the total is

$$2 \text{ containers/week} \times 3 \text{ weeks} = 6 \text{ containers}.$$

`Final answer: 6`

Matched recipe: Elapsed-Duration Schedule

APPLICABILITY
Use this method when a quantity accumulates at a constant rate during specific active intervals within a repeating time cycle, and the goal is to determine total accumulation over a duration spanning multiple cycles. This applies when event frequency is discontinuous, requiring distinction between total cycle length and active occurrences per cycle.
Do not use this method if the rate varies irregularly within the cycle, if the cycle structure is undefined, or if the goal is to find duration rather than total quantity.

PROCEDURE
one: Identify the quantity accumulated per single active occurrence and assign this to the event-quantity variable.
two: Identify the number of active occurrences strictly within one full repeating cycle and assign this to the active-count variable.
three: Calculate the total accumulation per full cycle by multiplying event-quantity by active-count to obtain the cycle-total.
four: Identify the total duration of one repeating cycle in standard time units and assign this to the cycle-length variable.
five: Identify the total scenario duration in the same standard units and assign this to the total-time variable.
six: Determine the number of full cycles by dividing total-time by cycle-length.
seven: Calculate the aggregate from full cycles by multiplying the number of full cycles by cycle-total.
eight: If a remainder exists, calculate active occurrences within that partial duration, multiply by event-quantity, and add the result to the aggregate from step seven.
nine: If the context requires whole entities and the result is fractional, apply ceiling rounding.

CHECKS

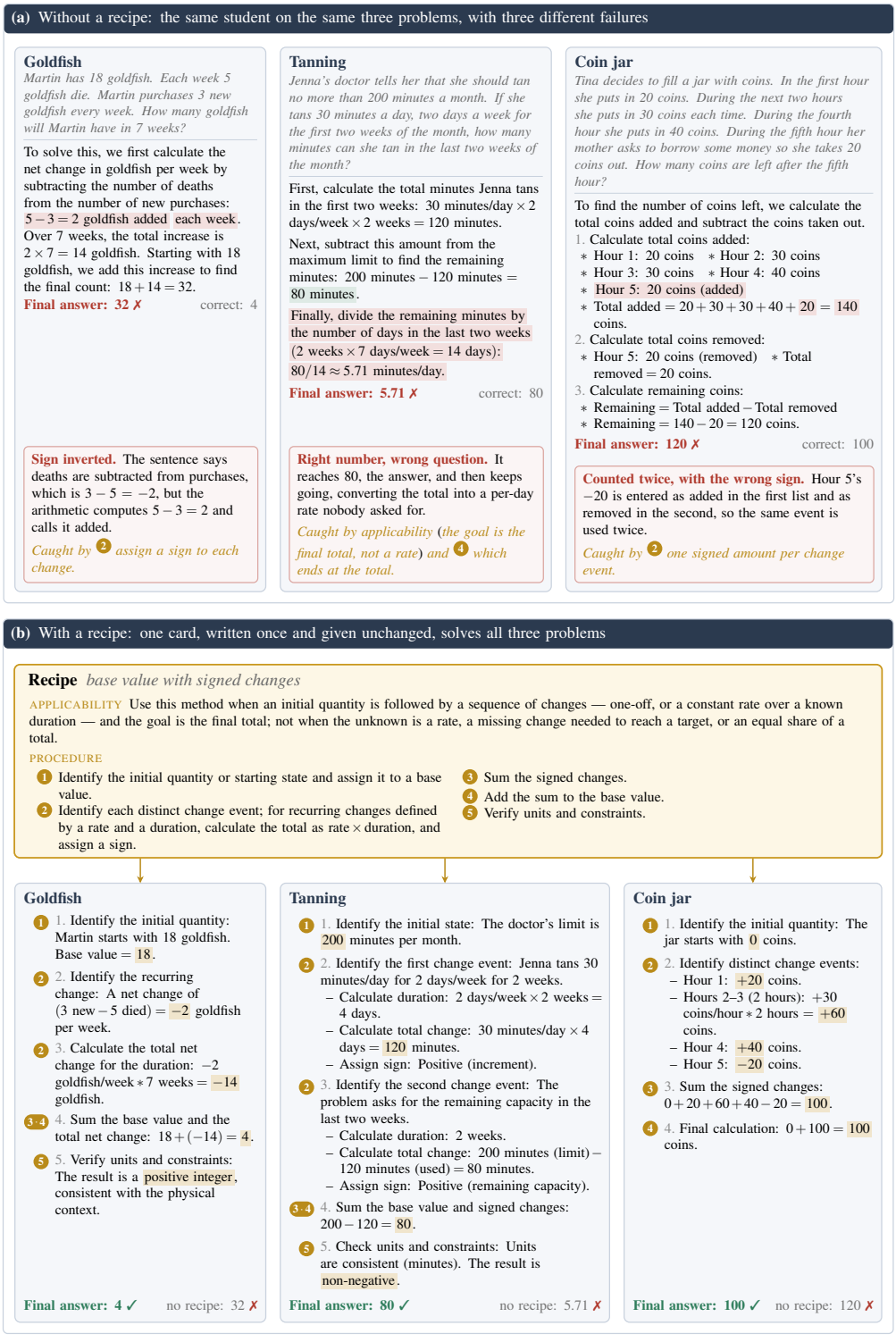


Figure 8: **From traces to a recipe.** (a) Without a recipe, the frozen anchor student fails all three problems in three different ways: it inverts a sign, answers a question that was not asked, and counts one change twice. Each callout names the recipe element that rules the error out. (b) The same recipe, given unchanged, leads the same student to the correct answer on all three. Executions are verbatim; each marker gives the recipe step a line executes, and gold highlights mark the quantity it produces. All three are GSM8K training problems used during construction.

Verify that active-count represents only intervals where the event occurs, not the total cycle length. Confirm that cycle-length includes both active and inactive periods. Ensure that the division in step six accounts for any remainder representing a partial cycle. Test the procedure by substituting total-time equal to cycle-length to ensure the calculated aggregate matches cycle-total exactly. Verify that active-count is less than or equal to the cycle-length in standard units.

Recipe-conditioned output (correct)

1. Identify the quantity per active occurrence: Emma buys 2 containers per school day.
2. Identify active occurrences per cycle: the cycle is one week, and she attends school 5 days per week. Thus,

$$\text{active-count} = 5.$$

3. Calculate the cycle total:

$$2 \times 5 = 10$$

containers per week.
4. Identify the cycle length: one week is 7 days.
5. Identify the total duration:

$$3 \text{ weeks} = 21 \text{ days}.$$

6. Calculate the number of full cycles:

$$21/7 = 3.$$

7. Calculate the aggregate:

$$3 \times 10 = 30.$$

8. There is no remainder because 21 days is exactly three full weeks.
9. No rounding is required.
`Final answer: 30`

The recipe changes the dependency structure of the solution: the student first counts the five active days in each seven-day cycle and only then accumulates over three cycles.

E.3 GSM8K: Global Total with Group-Specific Rates

The recipe-free student equates a total number of items with a total number of customers.

Problem

A watermelon stand sold 46 watermelons. Seventeen customers bought one melon, three customers bought three melons, and the rest bought two melons. How many customers bought two watermelons?

Recipe-free output (incorrect)

Let x be the number of customers who bought two watermelons. The total number of customers is 46. The number of customers who bought one melon is 17. The number of customers who bought three melons is 3.
The sum of customers buying one, two, and three melons must equal the total number of customers:

$$17 + 3 + x = 46.$$

Therefore,

$$20 + x = 46, \qquad x = 26.$$

`Final answer: 26`

Matched recipe: Global Measure with Group-Specific Rates

APPLICABILITY
Use this method-family recipe when a narrative establishes a fixed global total for a composite measure, such as total items or total cost, and defines multiple disjoint groups, each characterized by a specific count and a unique rate of contribution per member. The goal is to determine the count of a specific group where the rate is known but the count is unknown, often signaled by phrases such as "the rest."

Do not use this recipe if the global total represents a direct count of the groups themselves, if rates are unknown or variable, if groups overlap, or if the problem requires finding a rate rather than a count. This method is essential when known quantities must first be converted from group counts to measure units before subtraction.

PROCEDURE
one: Identify the narrative statement defining the global total measure and assign it to `Global_Total`.
two: Locate all explicitly stated groups with known counts and known rates. For each group, calculate the consumed measure by multiplying `Group_Count` by `Group_Rate`.
three: Sum these calculated measures to form `Partial_Measure_Sum`.
four: Subtract `Partial_Measure_Sum` from `Global_Total` to determine the `Remaining_Measure`.
five: Identify the target group's known rate and assign it to `Target_Rate`.
six: Divide `Remaining_Measure` by `Target_Rate` to isolate `Target_Count`.
seven: If multiple unknown groups exist, set up an equation in which the sum of all group measures equals `Global_Total` and solve for the required target variable.

CHECKS
Verify that multiplying the calculated `Target_Count` by its `Target_Rate` and adding it to `Partial_Measure_Sum` exactly reconstructs `Global_Total`. Confirm that `Target_Count` is a non-negative integer. Ensure all global and subgroup measures use consistent units, and verify that the groups are mutually exclusive and collectively exhaustive with respect to the global measure.

Recipe-conditioned output (correct)

1. Identify the global total measure:

$$\texttt{Global_Total} = 46$$

watermelons.
2. Convert each known customer group into watermelons consumed. Seventeen customers bought one melon:

$$17 \times 1 = 17.$$

Three customers bought three melons:

$$3 \times 3 = 9.$$

3. Calculate the partial measure:

$$17 + 9 = 26.$$

4. Determine the remaining measure:

$$46 - 26 = 20.$$

5. The target group buys two watermelons per customer, so

$$\texttt{Target_Rate} = 2.$$

6. Recover the target count:

$$20/2 = 10.$$

7. Verify:

$$(17 \times 1) + (3 \times 3) + (10 \times 2) = 17 + 9 + 20 = 46.$$

`Final answer: 10`

The recipe prevents a unit mismatch: the global total counts watermelons, while the unknown counts customers, so the student converts each known group into watermelons before solving for the remaining customers.

E.4 MATH: Domain of a Rational Function

The recipe-free student finds the excluded points but reports the wrong set.

Problem

Find the domain of the function

$$f(x) = \frac{x+1}{x^2+6x+8}.$$

Recipe-free output (incorrect)

To find the domain of the function $f(x) = \frac{x+1}{x^2+6x+8}$, we must identify all real values of x for which the expression is defined. A rational function is undefined when its denominator is zero.

Mathematical Setup: Set the denominator equal to zero and solve for x:

$$x^2 + 6x + 8 = 0$$

Method-Critical Intermediate Work: Factor the quadratic equation:

$$(x + 2)(x + 4) = 0$$

Apply the Zero Product Property:

$$x + 2 = 0 \implies x = -2, \qquad x + 4 = 0 \implies x = -4$$

The values that make the denominator zero are $x = -2$ and $x = -4$. These values must be excluded from the domain.

Verification: Substitute $x = -2$ into the denominator: $(-2)^2 + 6(-2) + 8 = 4 - 12 + 8 = 0$. Substitute $x = -4$ into the denominator: $(-4)^2 + 6(-4) + 8 = 16 - 24 + 8 = 0$. Both values result in a zero denominator, confirming they are excluded.

`Final answer: (-4, -2)`

Matched recipe: Rational-Function Domain Restrictions

APPLICABILITY
This method applies to determining the domain of a real-valued function defined as a combination of rational terms where denominators depend on the independent variable. Observable structural cues include multiple distinct fraction bars or a single fraction whose denominator is a composite expression.
The procedure identifies all inputs causing any denominator to vanish. It is not applicable to expressions involving radicals with even indices, logarithms, or cases where denominators are non-zero constants.

PROCEDURE
First: Isolate every distinct denominator quantity within the expression, treating each as an independent source of restriction.
Second: For each identified denominator, construct an equation by setting that denominator equal to zero.
Third: Solve each equation for the independent variable. If solving yields non-real solutions and the context requires a real-valued domain, discard those values; otherwise retain all valid roots.
Fourth: Aggregate all valid real roots into a single set of excluded inputs, removing duplicates.
Fifth: Sort the excluded values in ascending order and define the domain as the real numbers excluding these points, expressing the result as a union of open intervals.

CHECKS
Confirm that the expression contains a variable-dependent denominator. Ensure that every denominator has been identified independently and set equal to zero. Verify that all real roots are excluded from the final domain and that the reported union of intervals covers every remaining real number.

Recipe-conditioned output (correct)

1. Isolate the denominator:

$$x^2 + 6x + 8.$$

2. Set it equal to zero:

$$x^2 + 6x + 8 = 0.$$

3. Factor:

$$(x + 2)(x + 4) = 0,$$

so the excluded values are

$$x = -2,\ -4.$$

4. Both roots are real, so both must be excluded.
5. The excluded set is

$$\{-4, -2\}.$$

6. Therefore the domain is
$$(-\infty, -4) \cup (-4, -2) \cup (-2, \infty).$$
`Final answer:` $(-\infty, -4) \cup (-4, -2) \cup (-2, \infty)$

The recipe supplies no new roots, since the recipe-free student already found them. Its contribution is the final step: after identifying the excluded inputs, it requires the student to reconstruct the entire domain rather than report a single interval.

E.5 MATH: Least Integer Satisfying a Congruence

The recipe-free student ignores the magnitude constraint and returns a four-digit number.

Problem

What is the least five-digit positive integer which is congruent to $5 \pmod{15}$?

Recipe-free output (incorrect)

To find the least five-digit positive integer congruent to 5 modulo 15, we first determine the smallest five-digit number. The smallest five-digit integer is 10,000.

1. **Identify the range**: We are looking for n such that $10{,}000 \leq n < 10{,}000 + 15$.
2. **Set up the congruence**: We need $n \equiv 5 \pmod{15}$. This means $n = 15k + 5$ for some integer k.
3. **Test values**: We check integers k starting from 0 upwards to find the first n in the range.
   - If $k = 0$, $n = 5$ (too small).
   - If $k = 1$, $n = 20$ (too small).
   - ...
   - If $k = 649$, $n = 15(649) + 5 = 9735 + 5 = 9740$ (too small).
   - If $k = 650$, $n = 15(650) + 5 = 9750 + 5 = 9755$.
4. **Verification**: 9755 is a five-digit number ($10{,}000 \leq 9755$). Since $9755 = 15 \times 650 + 5$, it satisfies $n \equiv 5 \pmod{15}$.

`Final answer:` 9755

Matched recipe: Modular Lower-Bound Search

APPLICABILITY
Use this procedure when finding the smallest integer within a defined magnitude range that satisfies a system of simultaneous modular congruences, where each condition specifies a modulus and a required remainder. The magnitude constraint must establish a clear lower bound.
Do not use this method if the moduli require an incompatible combination, if the system has no solution, or if the problem seeks a value outside the defined magnitude range.

PROCEDURE
one: Identify the magnitude constraint and determine the integer lower bound for the search domain.
two: List each modular condition as a pair consisting of a modulus and a required remainder.
three: Combine the modular conditions into a single equivalent congruence when necessary, using an appropriate unified modulus and remainder.
four: Divide the lower bound by the unified modulus to obtain its quotient and remainder.
five: Calculate the difference required to reach the desired remainder by subtracting the lower-bound remainder from the target remainder.
six: If this difference is negative, add the unified modulus to normalize it.
seven: Add the normalized difference to the lower bound to obtain the candidate solution.

CHECKS
Confirm that the candidate satisfies the required magnitude or digit constraint. Verify that division by every original modulus yields the required remainder. Confirm that the unified modulus is valid and that no smaller integer satisfying the magnitude constraint also satisfies the congruence conditions.

Recipe-conditioned output (correct)

1. The integer must be five-digit, so the lower bound is

$$10{,}000.$$

2. The modular condition is

$$x \equiv 5 \pmod{15}.$$

3. There is only one modulus, so the unified congruence remains

$$x \equiv 5 \pmod{15}.$$

4. Divide the lower bound by 15:

$$10{,}000 = 15(666) + 10.$$

Thus the lower-bound remainder is 10.

5. The difference required to reach remainder 5 is

$$5 - 10 = -5.$$

6. Normalize the negative difference:

$$-5 + 15 = 10.$$

7. Add this offset to the lower bound:

$$10{,}000 + 10 = 10{,}010.$$

`Final answer: 10010`

The recipe turns the magnitude requirement into an explicit lower bound and performs the modular adjustment relative to that bound, which is exactly the step the recipe-free output skips.