SHARPENING TAX IN POST-TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

An emerging hypothesis about reinforcement learning (RL) post-training of large language models (LLMs) is that it merely sharpens existing behaviors of a base model, improving single-shot accuracy at the cost of solution coverage. Although this trade-off has been observed in math and coding tasks, it need not extend to agentic tasks, where multi-turn tool use and interaction may require capabilities newly acquired during post-training. Our surprising finding is that pre-trained LLMs, equipped with a light inference harness, can serve as capable agents. Despite far lower accuracy (pass@1), they often surpass their post-trained counterparts in solution coverage (pass@K) given a sufficient test-time budget. We further analyze the underlying mechanism and show that post-training pushes tasks toward two extremes, always solved or never solved, and thereby improves sampling efficiency and consistency at the cost of solution coverage. To measure this cost, we propose **Sharpening Tax**, a diagnostic metric that quantifies the loss in test-time scalability after post-training. Across 14 base/post-trained model pairs from four families and three agentic benchmarks (42 cases in total), the tax is prevalent in most settings, can be estimated from a few rollouts, and correlates well with other metrics. Finally, we present **posterior-tempered group sampling (PTGS)**, a simple plug-and-play Bayesian sampler that adapts the sampling temperature per prompt to its estimated difficulty. Applied during RL training in two agentic environments, PTGS pays a smaller tax than the fixed-temperature baseline, solving more tasks under repeated sampling while also improving single-shot accuracy. `Code:`

1 INTRODUCTION

Post-training at scale, particularly via reinforcement learning (RL) with verifiable rewards, has transformed pre-trained general text completion machines into reasoning models (Shao et al., 2024; Guo et al., 2025). Beyond competition-level math and coding reasoners, post-trained large language models (LLMs) now serve as goal-oriented agents that call tools, manage long context, and interact with an environment across multiple turns (Anthropic, 2025; OpenAI, 2025; Google, 2025; Meta Superintelligence Labs, 2026; xAI, 2026). A common belief behind this progress is that frontier-level reasoning capability emerges during intensive post-training, e.g., RL unlocks new capabilities and pushes the reasoning boundary beyond what the pre-trained base model can already reach (Uesato et al., 2022; Guo et al., 2025). However, this belief has recently been called into question.

*Does RL post-training create fundamentally new capabilities, or does it merely amplify a few rewarding behaviors that the base model already possesses, i.e., **distribution sharpening**?* This question has attracted broad attention (Chen et al., 2025b; Zhao et al., 2025; Wu et al., 2025; Yuan et al., 2026b; Wen et al., 2026; Shen et al., 2026b; Zhou, 2026). A dominant observation so far is that an RL-trained policy gains accuracy (pass@1) at the expense of solution coverage (pass@K) (Chen et al., 2025b; Zhao et al., 2025). Despite lots of reasonable positions, from advocacy to skepticism, with training-dependent (Liu et al., 2025a) and data-dependent (Zhang et al., 2025; Shen et al., 2026a) viewpoints in between (see the Appendix A), current evidence is mainly limited to math and coding tasks (Chen et al., 2025b; Zhao et al., 2025; He et al., 2025; Shao et al., 2026). Unfortunately, LLM coverage analysis in these domains does not settle the question since (1) pre-training provides enormous exposure to math and coding; (2) evaluation often checks only a final answer, so a lucky rollout with flawed reasoning earns credit (Skalse et al., 2022; Pan et al., 2024; Wen et al., 2026).

We argue that agentic tasks provide a stress testing evaluation suite to test the sharpening hypothesis. Unlike math and coding problems, for which a base model may already encode many candidate

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

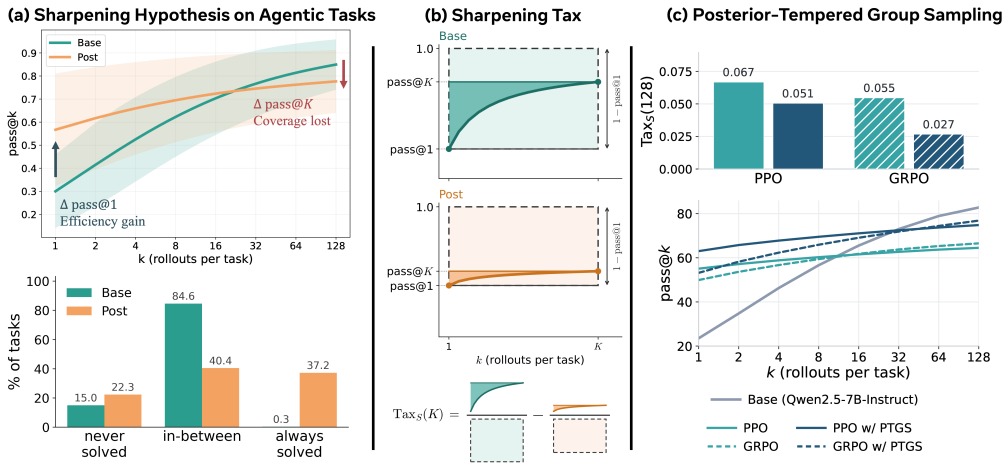


Figure 1: **Project overview**. We investigate whether post-training fundamentally broadens the agentic reasoning capacity of its base model. (a) Our observations suggest that post-training sharpens the policy to improve sampling efficiency and consistency while compromising coverage. To quantify this systematically, (b) we present **Sharpening Tax**, measuring the difference in test-time scalability between base and post-trained LLMs. To mitigate tax, (c) we then propose **Posterior-Tempered Group Sampling (PTGS)**, dynamically adjusting the sampling temperature based on task difficulty.

solution strategies, a model in agentic tasks must execute structured actions (Patil et al., 2025), invoke diverse tools through scenario-specific interfaces (Schick et al., 2023), incorporate heterogeneous environmental feedback under uncertainty (Oh et al., 2026b), and stay coherent over long-horizon trajectories (Froger et al., 2026). Those behaviors are far less common in pre-training data mix and are widely believed to be learned in post-training (Zeng et al., 2024; Su et al., 2026). Therefore, one might expect that post-training does more than redistributing probability mass among behaviors already present in the base model; it may be necessary to make successful agentic trajectories reachable at all.

To test this expectation, we systematically study LLMs' reasoning boundaries on agentic tasks before and after post-training. In particular, we first focus on the publicly available open-source checkpoint pairs of the base and post-trained LLMs to understand how large-scale general post-training affects the agentic reasoning boundary of base LLMs. Our surprising observation (§3) is that while the post-trained LLMs lead in single-shot accuracy, pre-trained base LLMs can also perform agentic reasoning when paired with a lightweight harness, and they even surpass their post-trained counterparts in terms of solution coverage given enough test-time compute. We then analyze how post-training changes the per-task success probability distribution, revealing that it bimodalizes the mass towards two extremes, always solved or never solved, to buy sampling efficiency (pass@1) and consistency (pass^K) at the expense of coverage (pass@K), diminishing the benefits of test-time parallel scaling.

Motivated by these test-time scaling dynamics, we propose **Sharpening Tax** (§4), a diagnostic metric that quantifies how much post-training shrinks the effective reasoning coverage as a single scalar. Across 14 model backbones from popular open source model families and three representative basic benchmarks for agentic tasks, we find that the tax is pervasive, implying that modern post-training consistently trades coverage for sampling efficiency and consistency; then we highlight the practical usefulness of Sharpening Tax, which can be estimated from a handful rollouts to predict the future tax with larger number of rollouts as well as other performance metrics.

Finally, we show that even task-specific RL tuning on a single domain pays Sharpening Tax. To mitigate this, we present **posterior-tempered group sampling (PTGS)**, a general sampler that balances exploration and exploitation. It adapts the sampling temperature per prompt based on difficulty to smooth policy for hard prompts and sharpen it for easy ones. Plugged into common RL algorithms (e.g., PPO and GRPO), PTGS pays a smaller tax while improving accuracy and coverage simultaneously over the baseline in agentic environments (§5). Its empirical success is further backed by theoretical analysis (§D). Fig. 1 shows the overview, and contributions are summarized as follow:

- We systematically study how post-training affects the average performance (pass@1) and solution coverage (pass@K) of LLMs on multi-turn, interactive agentic tasks, and show that harness-equipped base models can match or surpass their post-trained counterparts in terms of coverage.

- To enable diagnosis at scale, we propose *Sharpening Tax*, a metric that summarizes the effect of post-training on test-time scalability in a single number. Across 42 model-benchmark combinations, we find that the tax is substantial and pervasive, predictable from a few rollouts, and closely tied to other metrics such as the pass^K consistency gap between base and post-trained policies.
- We present *posterior-tempered group sampling (PTGS)*, a flexible plug-and-play sampler which adaptively sharpens or smoothens the sampling temperature per prompt; it mitigates the tax, improving accuracy and coverage simultaneously in multi-turn RL of LLM agents by increasing the chance of getting informative rollout.

2 PRELIMINARIES

Metrics. LLM reasoning boundaries are commonly probed through pass@k evaluation (Chen et al., 2025b; Zhao et al., 2025). For example, an LLM policy generates k independent rollouts for each task x_i, and success is evaluated across these trials. Let n_i be the total number of rollouts sampled for task $x_i \in \mathcal{D}$, of which $c_i \leq n_i$ succeed. Given a budget $k \leq n_i$, we consider three complementary metrics, each computed with the standard unbiased estimator (Chen et al., 2021) below.

- pass@$1_i = \frac{c_i}{n_i}$ **(Accuracy)**: Empirical pass rate of a single trial.
- pass@$k_i = 1 - \binom{n_i - c_i}{k} \Big/ \binom{n_i}{k}$ **(Coverage)**: Probability that at least one of k rollouts succeeds.
- pass$_i^k = \binom{c_i}{k} \Big/ \binom{n_i}{k}$ **(Consistency)**: Probability that all k rollouts succeed.

We report dataset-level metrics by averaging over all tasks, e.g., pass@$k = \frac{1}{|\mathcal{D}|} \sum_i$ pass@k_i. Intuitively, pass@1 captures the sampling efficiency of a policy, pass@k its solution coverage under a finite budget, and pass^k its success reliability across multiple attempts.

Datasets and environments. Most prior work evaluates pass@K reasoning boundary on math and coding benchmarks, which sometimes check only the final outcome. As a result, an incorrect reasoning trajectory that stumbles onto a lucky final answer still counts as a success, inflating pass@K (Wen et al., 2026). We instead evaluate on three agentic benchmarks that require multi-turn tool calling for final goal achievement: BFCL v4 multi-turn base split (Patil et al., 2025), WebShop (Yao et al., 2022), and ACEBench (Chen et al., 2025a). In these environments, intermediate actions and state transitions are checked by design, offering a faithful setup for stress testing of the agentic reasoning boundary. See Appendix C for more details.

Models. Our question is whether post-training extends the reasoning boundary of the base model. To explore this at scale, rather than developing the base and post-trained LLMs from scratch, we leverage open-source checkpoint pairs consisting of a pre-trained base model and its post-trained counterpart (e.g., `gemma-4-31B` vs. `gemma-4-31B-it`). Specifically, the evaluation spans 14 backbones from `Gemma-4` (Google, 2026), `Ministral-3` (Liu et al., 2026), `Qwen2.5` (Yang et al., 2024a), and `Qwen3.5` (Qwen, 2026) families in HuggingFace collections, ranging from 3B to 35B effective parameters (Tab. 3). The exact training recipes of these models are not fully disclosed, so we use the term loosely for RL post-trained models; since SFT and DPO (Rafailov et al., 2023) also sharpen the policy distribution (Huang et al., 2025), our analysis does not hinge on the exact recipe. Meanwhile, for fair comparison across different model backbones, we turn off thinking mode for all post-trained models that support the enforced explicit reasoning. See Appendix C.2 for justification and details.

3 THE ACCURACY-COVERAGE TENSION OF POST-TRAINED AGENT POLICY

Prior observations that base models have a higher performance upper bound than RL models come almost entirely from math and coding domains. In agentic domains, meanwhile, base models have likely seen far less data in those formats, and they are often assumed to lack the instruction-following and tool-calling skills that agentic tasks require (Zeng et al., 2024; Su et al., 2026). We fill this gap.

Base models in the harness handle agentic tasks, catch up to post-trained ones given test-time compute. Figure 2 presents our first observation, which holds consistently across model backbones and benchmarks. Pre-trained models equipped with a simple harness, perform agentic reasoning surprisingly well, catching up to their post-trained counterparts as the rollout budget K grows. This reminisces *large language monkeys* (Borel, 1913; Brown et al., 2024), but disciplined under harness.

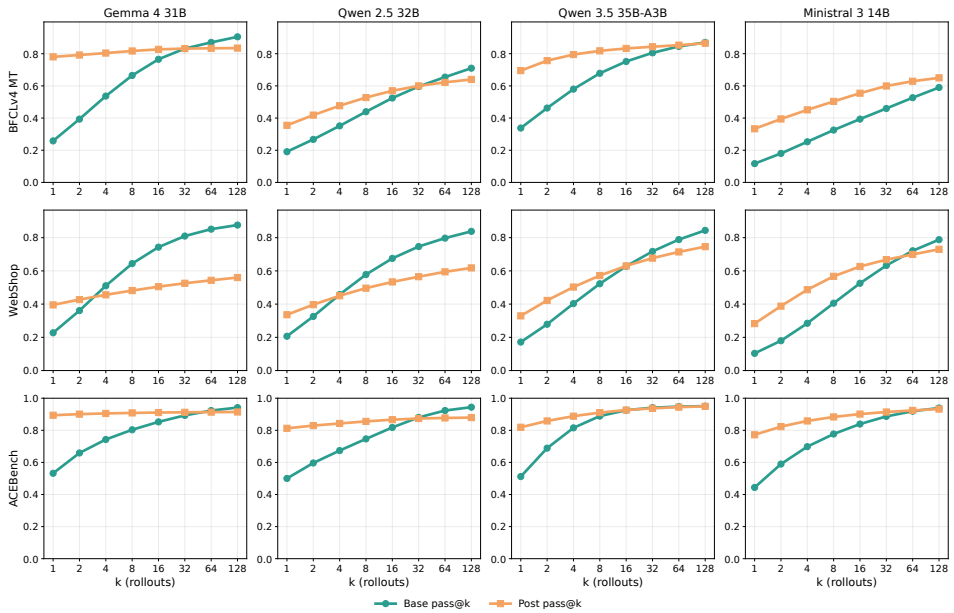


Figure 2: **Base models catch up to their RL post-trained counterparts on agentic tasks given sufficient compute**. We draw pass@K scaling curves (up to 128 rollouts per task) for four different models on three benchmarks. Post-trained models (orange) dominate at small K, but pre-trained base models equipped with an agentic harness (teal) scale more steeply and cross over in most settings.

By *harness*, we mean a dataset-independent, model-agnostic scaffolding that combines a simple system prompt with relaxed tool-calling and parsing interfaces similar to Yang et al. (2024b) or Wang et al. (2024). As shown in Table 1, harness dramatically improves both pass@1 and pass@32 on benchmarks that base models particularly struggle with, while leaving already-manageable tasks largely unchanged. Meanwhile, we observed that the same harness hurts the post-trained models' performance (Table 4), aligned with findings that instruction or harness can distract post-trained models (Wang et al., 2026a; Kim et al., 2026). Unless specified otherwise, we evaluate base models with the harness and post-trained models with the dataset-default scaffolding; see Appendix C.3 for details and full results.

We see a clear trade-off between the two policies. Post-trained models achieve higher single-sample accuracy (pass@1), but base models scale much more steeply in solution coverage (pass@K). As the budget grows to 128 rollouts per task, the curves of base models cross over and eventually surpass the post-trained ones across most benchmarks, e.g., over 85% pass@128 on WebShop vs. 56% for RL with `gemma-4-31B`. The natural next question is how post-training reshapes the policy to exhibit these consequences. A popular explanation is the ***sharpening hypothesis***: RL post-training amplifies a few high-reward behavior modes that the base model already contains, while suppressing the rest. We will dive deeper into this later in this section.

Table 1: **Harness ablation study of base models.** Average pass@1 and pass@32 of eight base models from the Gemma-4, Ministral-3, Qwen2.5, and Qwen3.5 families with (w/) and without (w/o) harness. Table 4 provides per-model results.

	pass@1		pass@32	
	w/o	w/	w/o	w/
BFCL	5.59	15.63	15.19	49.13
WebShop	6.32	9.73	39.48	49.15
ACEBench	44.24	42.63	89.25	87.69

Is sharpening a curse or a blessing? The scaling curves of base and post-trained models cross over, but when does the crossover happen? Figure 3 plots pass@K curves across four model scales of `Gemma-4` (4B, 12B, 26B, and 31B). We observe that larger models pull the crossover point k^* to fewer rollout budgets. In WebShop, for example, the crossover budget decreases from $k^* > 128$ for 4B to $k^* \approx 3$ for 31B. Appendix E.1 provides the full analysis results, including the similar conclusion from other model families.

This trend aligns with existing accounts of model capacity, saying larger models can store more behavior modes with less interference (Huang et al., 2026) and extract more structured information from the same data (Kaplan et al., 2020; Finzi et al., 2026), which helps them discover solutions within a small sampling budget (Wei et al., 2022). Therefore, sharpening cuts both ways. For smaller

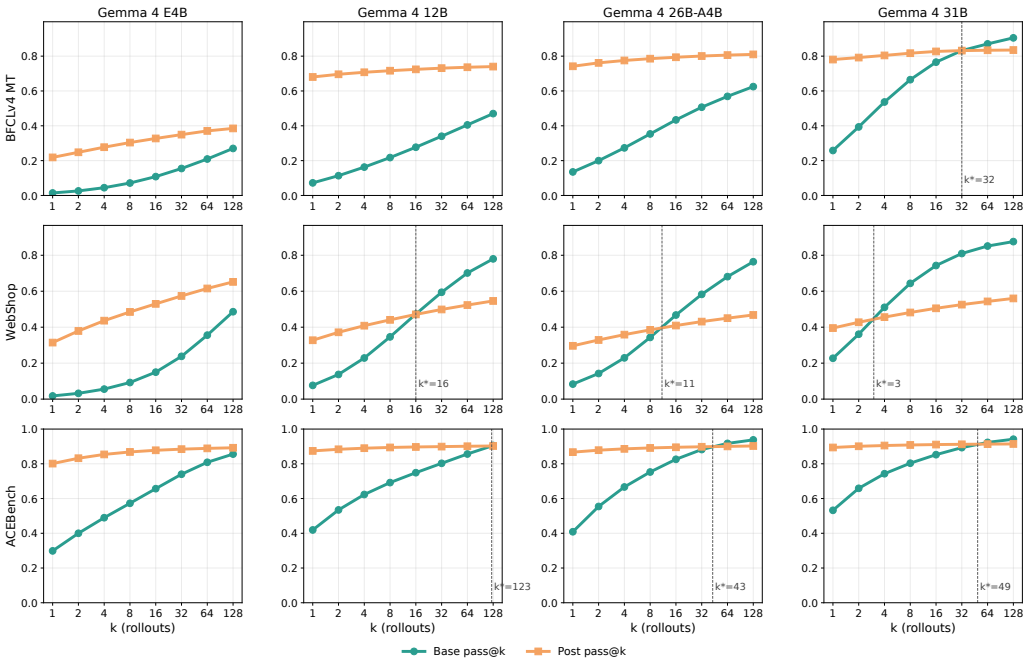


Figure 3: **Larger models pull the crossover points earlier.** Test-time scaling curves of pass@K for harness-equipped `gemma-4` base models (teal) vs. their post-trained counterparts (orange) with model size increasing from left to right. On every benchmark, the crossover budget at which the base model overtakes post-trained model generally shrinks as model scale grows.

models, it raises the performance floor (pass@1) with little coverage to lose; for larger models, it quickly lowers the performance ceiling (pass@K). Consequently, whether sharpening is harmful or beneficial should be interpreted along with model scale, budget, and application. For instance, a user who runs a large model with an abundant budget and only needs one promising solution among many rollouts is better served by the base model than by its post-trained counterpart. This is attractive whenever diversity and creativity matter, e.g., scientific discovery (Agarwal et al., 2025; Ghareeb et al., 2026; Yuksekgonul et al., 2026), automated research (Yamada et al., 2025; Lu et al., 2026), and long-standing open problems (Lee et al., 2025) given a scalable verifier (Kwok et al., 2026).

How does post-training shrink solution coverage? To look beyond test-time scaling curves, we categorize every task by its outcomes over 128 rollouts: *always pass* ($c_i = K$), *pass given compute* ($0 < c_i < K$), and *always fail* ($c_i = 0$). As shown in Figure 4 (and Table 6, Figure 13–16 in Appendix), post-training drastically reduces the middle category, *pass given compute* (e.g., from 87.6% to 30.0% on WebShop with `gemma-4-31B`), and pushes those tasks to both extremes, i.e., *always pass* grows (from 0.0% to 26.0%), but so does *always fail* (from 12.4% to 44.0%), which is aligned with the recent findings of Shen et al. (2026a) stating RL amplifies not only the correct modes of actions but also incorrect ones that base policy already prefers.

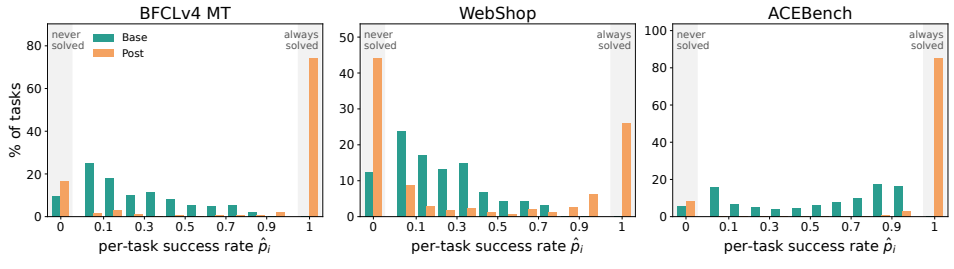


Figure 4: **Post-training bimodalizes per-task success rates** (`gemma-4-31B`, empirical pass rate over K=128 rollouts). The base model shows a spread distribution, whereas post-trained model polarizes it into a bimodal shape by pushing most of the intermediate *pass given compute* mass to the two extremes: always solved and never solved. See Appendix E.2 for full results.

In summary, from a distributional viewpoint over per-task empirical success rates, post-training show *bimodalization* effect. Base models spread most of their mass over the region of intermediate success rates, exactly where additional rollouts keep converting failures into successes, whereas the post-trained policies concentrate their mass onto two extrema, leaving little probability in between and hence little room to improve with more rollouts. Theorem 3 in Appendix D formalizes this.

Meanwhile, this bimodalization also makes the policy act more consistently, i.e., for a given prompt, the rollouts within a group show high agreement. Figure 5 confirms this by showing that, averaged over the 12 model-benchmark pairs (largest backbone per family $\times$ three benchmarks), the post-trained policy's coverage (pass@K) and consistency (pass^K) curves stay comparatively close, whereas the base model shows a remarkably wide gap, i.e., its consistency collapses to zero while its coverage eventually surpasses RL. In short, post-training buys sampling efficiency and consistency by paying with coverage. These phenomena replicate across all model families (See Appendix Figure 17– 20), with `Qwen3.5` exhibiting the softest sharpening and `Gemma-4` standing for the steepest sharpening model.

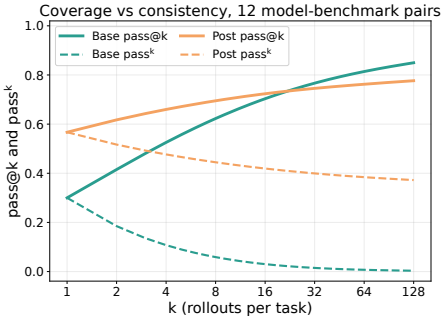


Figure 5: **Post-training trades coverage for consistency.** Mean pass@K and pass^K over 12 model-benchmark pairs.

4 Quantifying the Effect of Post-Training Sharpening

So far, we have examined how RL post-training reshapes test-time scaling dynamics by visualizing the scaling curve. However, manually inspecting full pass@K curves for every model and dataset quickly becomes impractical nor scalable. Meanwhile, evaluating models with only the endpoint metrics such as pass@K and pass^K at a specific K says little about the full landscape of the scaling behavior, e.g., whether success saturates within a few attempts or keeps rising. To this end, we propose **Sharpening Tax**, a diagnostic metric that summarizes the effect of post-training on test-time scalability in a single number. For an evaluation budget K, we define the following quantities.

- **Raw area scalability.** The cumulative performance recovered by additional compute (parallel rollouts) up to K, defined as the area beneath the pass@K ceiling:

$$A(K) = \sum_{k=1}^{K-1} \big[\text{pass@}K - \text{pass@}k\big] \tag{1}$$

  A larger value of $A(K)$ reflects a greater advantage from performing extra rollouts at test time.
- **Calibrated scalability.** We normalize $A(K)$ with the expected single-shot failure rate $(1-\text{pass@}1)$, i.e., accuracy headroom, to favor high average success rate while bounding $S(K) \in [0, 1]$:

$$S(K) = \frac{1}{K-1} \sum_{k=1}^{K-1} \frac{\text{pass@}K - \text{pass@}k}{1 - \text{pass@}1} = \frac{A(K)}{(K-1)(1-\text{pass@}1)} \tag{2}$$

- **Sharpening Tax.** For a given budget K, the deficit between the base and post-trained policies:

$$\text{Tax}_X(K) = X_{\text{Base}}(K) - X_{\text{Post}}(K), \quad X \in \{A, S\} \tag{3}$$

A positive $\text{Tax}_X(K)$ means post-training reduces test-time scalability relative to the base model, charging a "tax" on the coverage gains achievable through compute scaling up to K. Next, we show how broadly Sharpening Tax is charged across models and benchmarks (Fig. 6), and why it is a useful diagnostic (Fig. 7). Appendix D provides deeper theoretical insights for our tax formulation.

Sharpening Tax is charged broadly across modern post-training pipelines. Figure 6 reports the calibrated tax $\text{Tax}_S(k)$ of the largest and smallest backbone of each model family on the three benchmarks as a function of the rollout budget k (Figure 24 in Appendix F.1 reports both tax variants for all 14 pairs). For the largest backbones, the tax is positive at almost every budget and benchmark, and it is largest for the largest checkpoints such as `gemma-4-31B`. For the smallest backbones, the tax is often negative at small budgets ($k \leq 8$), most visibly for `Ministral-3-3B`, reflecting that sharpening does help when compute is severely constrained. As the budget scales toward $k = 128$, however, the tax rises in most of these cases, e.g., $\text{Tax}_S(128) > 0$ in 36 of the 42 combinations.

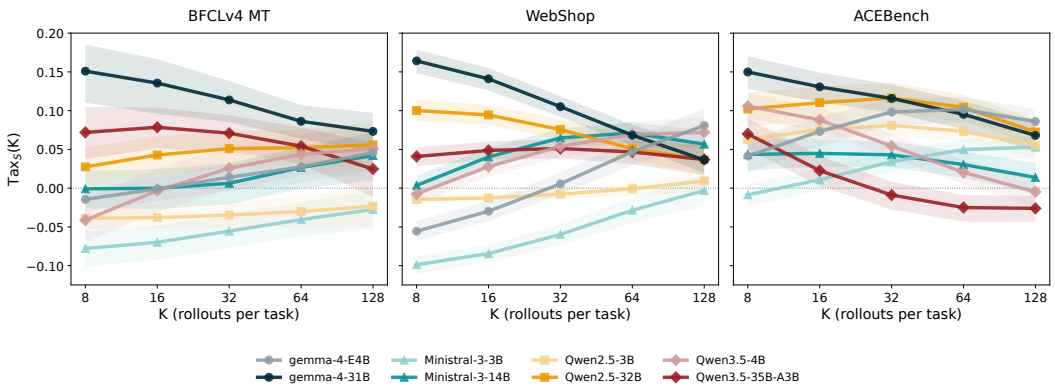


Figure 6: **Sharpening Tax across rollout budgets.** $\text{Tax}_S(k)$ of eight models (the largest and smallest backbone of each family) across three benchmarks; positive values indicate the base model's scaling advantage over RL. Shades represent 95% bootstrap confidence intervals. The tax is positive at almost every budget for the largest backbones, and for the smallest backbones it grows from negative values toward zero or above as the test-time budget scales. Refer to Figure 24 for full results.

The only cases where both tax variants turn negative are `Ministral-3-3B`, `Qwen2.5-3B`, and `Qwen2.5-7B` on BFCL, where post-training genuinely expands the boundary within the considered budget. In other words, the coverage trade-off we observed in the previous section is not specific to one model or benchmark; The modern post-training pipelines prioritize immediate sampling efficiency and consistency at the expense of solution coverage upper bound achievable by test-time scaling. The two variants also differ in shape. While both are positive in most cases, the raw tax (Tax_A; Figure 24) increases almost monotonically with the budget, whereas the calibrated tax (Tax_S) can either rise or fall with the budget, owing to its accuracy-headroom calibration. Moreover, a positive tax has a concrete probabilistic interpretation (Proposition 1 and Corollary 2 in Appendix D)—after post-training, successes rely less on retries, since failed first attempts are recovered less often (lost coverage) or sooner (faster saturation). Table 7 in Appendix separates the two effects.

Sharpening Tax is predictable and strongly correlates with other metrics. Beyond describing the observed tax values, we ask two practical questions: can tax be estimated cheaply without full budget evaluation, and what does it tell us about other evaluation metrics? Figure 7 provides answers.

1. **Early predictability**: In Figure 7 (Left), $\text{Tax}_S(8)$, estimated from 8 rollouts on half of the tasks, strongly predicts $\text{Tax}_S(32)$ on the held-out half of the tasks, achieving a remarkably high Spearman correlation $\rho = 0.85$. That is, a cheap 8-rollout estimate is a reliable early signal of how the tax will grow, indicating the practicality of the tax estimation. In Figure 31, we showcase an interesting application, routing base and post-trained models per task based on tax estimates.
2. **Correlation with the consistency gap**: As shown in Figure 7 (Middle), the tax computed at $K = 8$ exhibits a clear positive rank correlation with the consistency gap between the RL and base models, $\Delta\text{pass}^8 = \text{pass}^8_{\text{RL}} - \text{pass}^8_{\text{Base}}$. This supports our earlier finding in Figure 5: the more consistent the post-trained policy becomes, the larger the tax it pays in lost coverage.
3. **Comparison with other predictors**: Figure 7 (Right) compares early predictors using only 8 rollouts against three future targets from 32 held-out rollouts ($\text{Tax}_S(32)$, pass@32, and pass^{32}). The early tax $\text{Tax}_S(8)$ is the only one that predicts all three metrics with a strong statistical sign. This implies that the proposed Sharpening Tax metric is not only good for diagnosing the tradeoff in post-training, but also for evaluating the policy's potential in test-time scaling.

To summarize, Sharpening Tax is a practical diagnostic of a post-trained policy, which can be estimated from a handful of rollouts, extrapolates reliably to larger budgets, and is predictive of other metrics of interest, such as the consistency gap and future pass@K.

5 How to Balance Sampling Efficiency and Solution Coverage?

Previous sections established that post-training charges Sharpening Tax, collapsing the policy onto a few rewarding behaviors and compromising the rich coverage that the base model had. This diagnosis

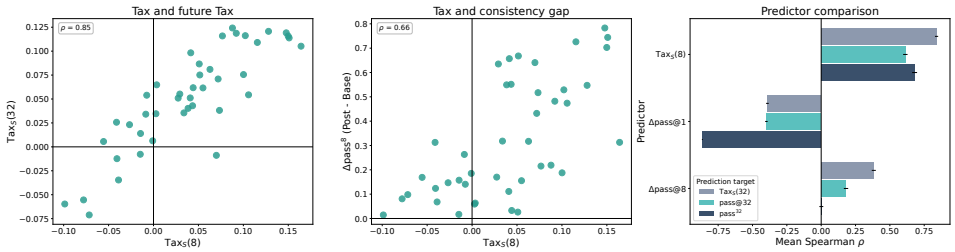


Figure 7: **Sharpening Tax extrapolates to larger budgets and predicts both the consistency gap and the value of additional test-time compute.** Panels span 42 model-benchmark combinations and report Spearman rank correlation ρ; estimation and validation use disjoint task halves. (**Left**) $\mathrm{Tax}_S(8)$ vs. future tax $\mathrm{Tax}_S(32)$. (**Middle**) $\mathrm{Tax}_S(8)$ estimated from 8 rollouts vs. consistency gap $\Delta\mathrm{pass}^8$. (**Right**) Mean Spearman ρ of candidate predictors against future performance targets.

immediately raises a question: *Can we keep the sampling efficiency that post-training buys while securing the coverage it takes away?* The most obvious remedy, increasing temperature helps to improve pass@K but monotonically degrades pass@1 (Figure 23 in Appendix E.4), since mastered prompts need no extra entropy while never-solved prompts need far more. To tackle this dilemma, we introduce a simple Bayesian sampler pursuing per-prompt adaptive smoothing during RL training.

Posterior-tempered group sampling. We propose *posterior-tempered group sampling (PTGS)*, estimating per-task difficulty online from the empirical success rate of the policy and letting this running estimate set the temperature of the next sampling across training progress. To be specific, whenever we observe n rollouts per prompt x, we track cumulative discounted success and failure counts $(\tilde{s}_x, \tilde{f}_x) \leftarrow (\gamma\tilde{s}_x + s_x,\ \gamma\tilde{f}_x + n - s_x)$ throughout training with a forgetting factor $0 \le \gamma < 1$. We set them as parameters of a Beta distribution combined with a target success rate $\tilde{p}$, forming the posterior $\mathrm{Beta}(a_x, b_x)$ via $a_x = 2\tilde{p} + \tilde{s}_x$ and $b_x = 2(1-\tilde{p}) + \tilde{f}_x$. Then, PTGS draws $\hat{p}_x \sim \mathrm{Beta}(a_x, b_x)$ by Thompson sampling and decodes the next group of n rollouts when facing the prompt x at the temperature T_x as follows:

$$T_x = \tau^{\,h(\hat{p}_x)},$$
$$h(p) = \begin{cases} \frac{\tilde{p}-p}{\tilde{p}}, & p \le \tilde{p} \quad \text{(hard prompt)}, \\ \frac{\tilde{p}-p}{1-\tilde{p}}, & p > \tilde{p} \quad \text{(easy prompt)}. \end{cases} \tag{4}$$

Algorithm 1 PTGS within an RL training loop

Require: task pool $\mathcal{D}$, group size n, target success rate $\tilde{p}$, base temperature $\tau > 1$, and forgetting factor $0 < \gamma \le 1$

$(\tilde{s}_x, \tilde{f}_x) \leftarrow (0, 0)$ for every $x \in \mathcal{D}$
for RL training step $t = 1, 2, \ldots$ **do**
  Sample a task batch $\mathcal{B}_t \subset \mathcal{D}$
  for each prompt $x \in \mathcal{B}_t$ **do**
    $(a_x, b_x) \leftarrow \big(2\tilde{p} + \tilde{s}_x,\ 2(1-\tilde{p}) + \tilde{f}_x\big)$
    Draw $\hat{p}_x \sim \mathrm{Beta}(a_x, b_x)$
    Set $T_x \leftarrow \tau^{\,h(\hat{p}_x)}$ using Eq. (4)
    Generate n rollouts at temperature T_x and count s_x successes
    $(\tilde{s}_x, \tilde{f}_x) \leftarrow \big(\gamma\tilde{s}_x + s_x,\ \gamma\tilde{f}_x + n - s_x\big)$
  end for
  Apply the chosen RL algorithm's policy update rule
end for

where the base temperature $\tau > 1$ defines the spread with a range $T_x \in [1/\tau, \tau]$. With this Bayesian sequential difficulty estimator, a prompt with a lower $\hat{p}_x$ is heated up to τ, to encourage exploratory rollouts, and a prompt with a higher $\hat{p}_x$ is cooled down to $1/\tau$, for exploitative rollouts. Although we drop this adaptive sampler at test time, intervening in the sampling distribution of training experience can improve the resulting policy's behavior after training (Schaul et al., 2015; Nath et al., 2025) by reinforcing the model's action sequence towards exploratory and exploitative trajectories depending on the prompt. Note that *PTGS does not require any changes to RL update itself nor bring computational overhead*, enabling its flexible application to any modern RL algorithms in practice. See Algorithm 1 for the procedure summary and Appendix C.5 for implementation details.

PTGS gets a tax break: sharper sampling without narrower coverage. Since PTGS changes rollout sampling during RL training, we evaluate it by fine-tuning our own agents. We thus study the sharpening from a general post-trained policy to a task-specific RL policy. Following RAGEN (Wang et al., 2025; 2026b), we fine-tune `Qwen2.5-7B-Instruct` on Sokoban and FrozenLake with two representative RL algorithms, PPO and GRPO, each with and without PTGS. All runs use 200 training steps and the StarPO-S signal-to-noise ratio reward-variance filter. We report means over five runs, using the final checkpoint for PPO and the checkpoint with the highest validation success in

each run for GRPO. We adopt $\tau \in [1.2, 1.5]$ and $\gamma = 0.95$, and gradually raise pivot success rate $\tilde{p}$ from 0.25 to 0.5 during training. Appendix C.5 details the training, PTGS settings, and checkpoint selection.

Table 2 shows that task-specific RL still inherits the accuracy-coverage tension. That is, both algorithms improve accuracy over the base model while losing solution coverage. **PTGS reduces the tax and improves both pass@1 and pass@128 for two representative RL methods, PPO and GRPO, in both environments.** These joint gains suggest that some of the coverage loss can be avoided by changing how training rollouts are sampled, while retaining the accuracy gains of RL.

Table 2: **PTGS improves accuracy and coverage while reducing tax.** RL training results of Qwen2.5-7B-Instruct, averaged over five random seed runs. †Best validation success checkpoint per run; PPO uses the final checkpoint.

	Method	pass@1 ↑	pass@128 ↑	$\text{Tax}_S(128)$ ↓
Sokoban	Base	20.7	76.6	-
	PPO	46.5	55.0	0.094
	PPO w/ PTGS	**61.1**	**69.7**	**0.081**
	GRPO†	36.5	55.3	0.081
	GRPO† w/ PTGS	**39.1**	**72.5**	**0.025**
FrozenLake	Base	26.3	89.1	-
	PPO	63.7	74.1	0.039
	PPO w/ PTGS	**65.0**	**80.0**	**0.020**
	GRPO†	63.4	77.8	0.029
	GRPO† w/ PTGS	**67.3**	**81.2**	**0.028**

Theorem 4 in Appendix D.4 offers an explanation through the training signal. If tempering raises a difficult prompt's success probability, PPO receives groups containing a success more often. Besides, rollout groups containing both successes and failures also become more likely, which is important for GRPO to induce nonzero group-relative advantages. Therefore, PTGS can keep difficult prompts contributing to learning under both algorithms, helping preserve coverage as accuracy improves. See Appendix G.1 and G.2 for further training dynamics and temperature ablations.

Next, Figure 8 shows the PPO training dynamics on FrozenLake. If we look at the validation success rate, the two methods look similar and reach the same value at the final checkpoint. However, the average entropy of the output token distribution evolves very differently. Under fixed-temperature PPO, the entropy decreases monotonically thereby the policy converges to a few action sequences. Under PTGS, in contrast, the entropy stays much higher throughout training and rises repeatedly, since PTGS heats the prompts that the policy keeps failing on. Therefore, the two runs end with the same accuracy but very different policies: PTGS keeps exploratory behavior for challenging prompts, which explains its higher coverage and smaller tax in Table 2. In short, **PTGS yields a policy that is as accurate as fixed-temperature PPO but has broader space of solvable problems**.

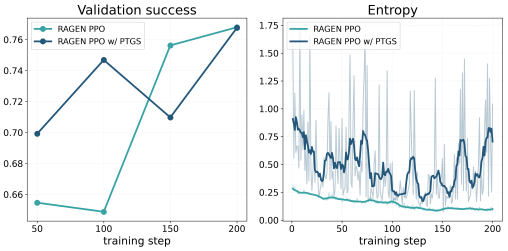


Figure 8: **Training dynamics on FrozenLake.** Validation success of intermediate checkpoints (left) and average entropy of the output token distribution during training for PPO and PPO with PTGS.

6 Conclusion

We explored whether RL post-training expands or merely sharpens the agentic reasoning boundary of its base model. Across fourteen open checkpoint pairs and three benchmarks, we found that pre-trained base models equipped with a lightweight harness are surprisingly capable agentic reasoners that, given enough test-time compute, match or surpass their post-trained counterparts in coverage. We then reveal that post-training buys sampling efficiency and consistency by pushing tasks toward the always-solved and never-solved extremes. To measure this tradeoff in a single number, we introduced the **Sharpening Tax** and showed that the tax is pervasive across 42 model-benchmark combinations, predictable from a handful of rollouts, tightly coupled with our performance metrics, and grounded in a theoretical account of what it measures and when sharpening must charge it. We then proposed **PTGS**, which sets the sampling temperature of each prompt according to its estimated difficulty, heating prompts the policy still fails on and cooling those it has already mastered. In multi-turn RL training, it improves accuracy and coverage simultaneously while paying a smaller tax.

AI Tool Usage Disclosure

We used large language model agents driven by Claude, GPT, Gemini, and Muse to assist with writing code for method implementation and experiments, running and managing experiments, and polishing the human-written manuscript draft and theoretical analyses. The lead author reviewed and fact-checked all AI-written content in the paper. The authors take responsibility for all the content of this work, including text, claims, code, and other artifacts produced with AI assistance.

Ethics Statement

We use publicly available model checkpoints and benchmarks and involves no human participants or collection of personal data. We do not see ethical concerns specific to this work.

Reproducibility Statement

The main text describes our methods and evaluation metrics, with detailed experimental settings provided in Appendix C. For full reproducibility, we share our code through an anonymous repository at https://anonymous.4open.science/r/sharpeningtax-anonymous-33AE/README.md

References

Dhruv Agarwal, Bodhisattwa Prasad Majumder, Reece Adamson, Megha Chakravorty, Satvika Reddy Gavireddy, Aditya Parashar, Harshit Surana, Bhavana Dalvi Mishra, Andrew McCallum, Ashish Sabharwal, and Peter Clark. Autodiscovery: Open-ended scientific discovery via bayesian surprise. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

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*, volume 2024, pp. 21246–21263, 2024.

Anthropic. Claude code. https://code.claude.com/docs/en/overview, 2025.

Aaron Blakeman, Aaron Grattafiori, Aarti Basant, Abhibha Gupta, Abhinav Khattar, Adi Renduchintala, Aditya Vavre, Akanksha Shukla, Akhiad Bercovich, Aleksander Ficek, et al. Nemotron 3 nano: Open, efficient mixture-of-experts hybrid mamba-transformer model for agentic reasoning. *arXiv preprint arXiv:2512.20848*, 2025.

Émile Borel. La mécanique statique et l'irréversibilité. *Journal de Physique Théorique et Appliquée*, 3(1):189–196, 1913. doi: 10.1051/jphystap:019130030018900.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Sungmin Cha and Kyunghyun Cho. Why knowledge distillation works in generative models: A minimal working explanation. *Advances in Neural Information Processing Systems*, 38:30017–30037, 2026.

Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Yuefeng Huang, et al. Acebench: Who wins the match point in tool usage? *arXiv preprint arXiv:2501.12851*, 2025a.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Zhaorun Chen, Zhuokai Zhao, Kai Zhang, Bo Liu, Qi Qi, Yifan Wu, Tarun Kalluri, Xuefei Cao, Yuanhao Xiong, Haibo Tong, et al. Scaling agent learning via experience synthesis. In *International Conference on Learning Representations*, volume 2026, pp. 121394–121420, 2026.

Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *Advances in Neural Information Processing Systems*, 38:57654–57689, 2025b.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

Xiangxiang Chu, Hailang Huang, Xiao Zhang, Fei Wei, and Yong Wang. Gpg: A simple and strong reinforcement learning baseline for model reasoning. In *International Conference on Learning Representations*, volume 2026, pp. 59637–59659, 2026.

Haoran Dang, Cuiling Lan, Hai Wan, Xibin Zhao, and Yan Lu. Temperature as a meta-policy: Adaptive temperature in LLM reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026.

Xingyu Dang, Christina Baek, Kaiyue Wen, Zico Kolter, and Aditi Raghunathan. Weight ensembling improves reasoning in language models. *arXiv preprint arXiv:2504.10478*, 2025.

Marius Dragoi, Ioana Pintilie, Florin Gogianu, and Florin Brad. Beyond pass@ k: Breadth-depth metrics for reasoning boundaries. *arXiv preprint arXiv:2510.08325*, 2025.

Marc Finzi, Shikai Qiu, Yiding Jiang, Pavel Izmailov, J Zico Kolter, and Andrew Gordon Wilson. From entropy to epiplexity: Rethinking information for computationally bounded intelligence. *arXiv preprint arXiv:2601.03220*, 2026.

Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, et al. Gaia2: Benchmarking llm agents on dynamic and asynchronous environments. In *International Conference on Learning Representations*, volume 2026, pp. 119758–119789, 2026.

Jingchu Gai, Guanning Zeng, Huaqing Zhang, and Aditi Raghunathan. Differential smoothing mitigates sharpening and improves llm reasoning. *arXiv preprint arXiv:2511.19942*, 2025.

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *International conference on machine learning*, pp. 10835–10866. PMLR, 2023.

Ali E. Ghareeb, Benjamin Chang, Ludovico Mitchener, Angela Yiu, Caralyn J. Szostkiewicz, Dmytro Shved, Gavin J. Gyimesi, Jon M. Laurent, Samantha M. Wright, Muhammed T. Razzak, Andrew D. White, Silvia C. Finnemann, Michaela M. Hinks, and Samuel G. Rodriques. A multi-agent system for automating scientific discovery. *Nature*, 655(8122):497–505, July 2026. ISSN 1476-4687. doi: 10.1038/s41586-026-10652-y.

Google. Introducing Google Antigravity, a new era in AI-assisted software development. Google Blog, 2025. URL https://antigravity.google/blog/introducing-google-antigravity.

DeepMind Google. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Minillm: On-policy distillation of large language models. *arXiv preprint arXiv:2306.08543*, 2023.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Andre Wang He, Daniel Fried, and Sean Welleck. Rewarding the unlikely: Lifting GRPO beyond distribution sharpening. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 25548–25560. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.1298.

Byeongho Heo, Jaehui Hwang, Sangdoo Yun, and Dongyoon Han. On-policy delta distillation. *arXiv preprint arXiv:2607.15161*, 2026.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Jingcheng Hu, Yinmin Zhang, Qi Han, Daxin Jiang, Xiangyu Zhang, and Heung-Yeung Shum. Open-reasoner-zero: An open source approach to scaling up reinforcement learning on the base model. *Advances in Neural Information Processing Systems*, 38:162239–162262, 2026.

Audrey Huang, Adam Block, Dylan Foster, Dhruv Rohatgi, Cyril Zhang, Max Simchowitz, Jordan Ash, and Akshay Krishnamurthy. Self-improvement in language models: The sharpening mechanism. In *International Conference on Learning Representations*, volume 2025, pp. 76687–76739, 2025.

Jing Huang, Daniel Wurgaft, Rachit Bansal, Laura Ruis, Naomi Saphra, David Alvarez-Melis, Andrew Kyle Lampinen, Christopher Potts, and Ekdeep Singh Lubana. Why larger models learn more: Effects of capacity, interference, and rare-task retention. *arXiv preprint arXiv:2605.29548*, 2026.

Ahmadreza Jeddi, Kimia Shaban, Negin Baghbanzadeh, Natasha Sharan, Abhishek Moturu, Elham Dolatabadi, and Babak Taati. When does rl help medical vlms? disentangling vision, sft, and rl gains. *arXiv preprint arXiv:2603.01301*, 2026.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Aayush Karan and Yilun Du. Reasoning with sampling: Your base model is smarter than you think. In *International Conference on Learning Representations*, volume 2026, pp. 476–494, 2026.

Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. VinePPO: Refining credit assignment in RL training of LLMs. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 29557–29590. PMLR, 13–19 Jul 2025.

Kyungmin Kim, Youngbin Choi, Seoyeon Lee, Suhyeon Jun, Dongwoo Kim, and Sangdon Park. The interplay of harness design and post-training in llm agents. *arXiv preprint arXiv:2606.25447*, 2026.

Jacky Kwok, Christopher Agia, Rohan Sinha, Matt Foutter, Shulu Li, Ion Stoica, Azalia Mirhoseini, and Marco Pavone. Robomonkey: Scaling test-time sampling and verification for vision-language-action models. In *Conference on Robot Learning*, 2025.

Jacky Kwok, Shulu Li, Pranav Atreya, Yuejiang Liu, Yixing Jiang, Chelsea Finn, Marco Pavone, Ion Stoica, and Azalia Mirhoseini. Llm-as-a-verifier: A general-purpose verification framework. *arXiv preprint arXiv:2607.05391*, 2026.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Sangyun Lee, Brandon Amos, and Giulia Fanti. Banel: Exploration posteriors for generative modeling using only negative rewards. *arXiv preprint arXiv:2510.09596*, 2025.

Haozhan Li, Yuxin Zuo, Jiale Yu, Yuhao Zhang, Yang Zhaohui, Kaiyan Zhang, Xuekai Zhu, Yuchen Zhang, Tianxing Chen, Ganqu Cui, et al. Simplevla-rl: Scaling vla training via reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 125846–125869, 2026a.

Junbo Li, Peng Zhou, Rui Meng, Meet P Vadera, Lihong Li, and Yang Li. Turn-ppo: Turn-level advantage estimation with ppo for improved multi-turn rl in agentic llms. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 6227–6243, 2026b.

Xiang Lisa Li, Ari Holtzman, Daniel Fried, Percy Liang, Jason Eisner, Tatsunori B Hashimoto, Luke Zettlemoyer, and Mike Lewis. Contrastive decoding: Open-ended text generation as optimization. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 12286–12312, 2023.

Mengqi Liao, Xiangyu Xi, Chen Ruinian, Jia Leng, Yangen Hu, Ke Zeng, Shuai Liu, and Huaiyu Wan. Enhancing efficiency and exploration in reinforcement learning for llms. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 1451–1463, 2025.

Alexander H Liu, Kartik Khandelwal, Sandeep Subramanian, Victor Jouault, Abhinav Rastogi, Adrien Sadé, Alan Jeffares, Albert Jiang, Alexandre Cahill, Alexandre Gavaudan, et al. Ministral 3. *arXiv preprint arXiv:2601.08584*, 2026.

Mingjie Liu, Shizhe Diao, Ximing Lu, Jian Hu, Xin Dong, Yejin Choi, Jan Kautz, and Yi Dong. ProRL: Prolonged reinforcement learning expands reasoning boundaries in large language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025a.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. In *Second Conference on Language Modeling*, 2025b.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Chris Lu, Cong Lu, Robert Tjarko Lange, Yutaro Yamada, Shengran Hu, Jakob Foerster, David Ha, and Jeff Clune. Towards end-to-end automation of ai research. *Nature*, 651(8107):914–919, 2026.

Kevin Lu. On-policy distillation. https://thinkingmachines.ai/blog/on-policy-distillation/, 2025. Thinking Machines Lab.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, et al. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*, 2024.

Meta Superintelligence Labs. Introducing Muse Spark 1.1. Meta AI Blog, July 2026. URL https://ai.meta.com/blog/introducing-muse-spark-meta-model-api/.

Microsoft AI Team. MAI-Thinking-1: Building a hill-climbing machine. Technical report, Microsoft AI, 2026. URL https://microsoft.ai/pdf/mai-thinking-1.pdf.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20286–20332, 2025.

Vaskar Nath, Elaine Lau, Anisha Gunjal, Manasi Sharma, Nikhil Baharte, and Sean Hendryx. Adaptive guidance accelerates reinforcement learning of reasoning models. *arXiv preprint arXiv:2506.13923*, 2025.

Sean O'Brien and Mike Lewis. Contrastive decoding improves reasoning in large language models. *arXiv preprint arXiv:2309.09117*, 2023.

Changdae Oh, Yixuan Li, Kyungwoo Song, Sangdoo Yun, and Dongyoon Han. Dawin: Training-free dynamic weight interpolation for robust adaptation. In *International Conference on Learning Representations*, volume 2025, pp. 11980–12008, 2025.

Changdae Oh, Wendi Li, Seongheon Park, Samuel Yeh, Tanwi Mallick, and Sharon Li. Neglected free lunch from post-training: Progress advantage for llm agents. *arXiv preprint arXiv:2606.26080*, 2026a.

Changdae Oh, Seongheon Park, To Eun Kim, Jiatong Li, Wendi Li, Samuel Yeh, Sean Du, Hamed Hassani, Paul Bogdan, Dawn Song, et al. Uncertainty quantification in llm agents: Foundations, emerging challenges, and opportunities. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16219–16250, 2026b.

Olmo Team. Olmo 3. *arXiv preprint arXiv:2512.13961*, 2025.

OpenAI. Codex. https://developers.openai.com/codex, 2025.

Alexander Pan, Erik Jones, Meena Jagadeesan, and Jacob Steinhardt. Feedback loops with language models drive in-context reward hacking. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 39154–39200. PMLR, 21–27 Jul 2024.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. *International Conference on Machine Learning*, 2025.

Team Qwen. Qwen3.5-omni technical report. *arXiv preprint arXiv:2604.15804*, 2026.

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. *Advances in neural information processing systems*, 36:53728–53741, 2023.

Salman Rahman, Jingyan Shen, Anna Mordvina, Hamid Palangi, Saadia Gabriel, and Pavel Izmailov. When can llms learn to reason with weak supervision? *arXiv preprint arXiv:2604.18574*, 2026.

Alexandre Ramé, Johan Ferret, Nino Vieillard, Robert Dadashi, Léonard Hussenot, Pierre-Louis Cedoz, Pier Giuseppe Sessa, Sertan Girgin, Arthur Douillard, and Olivier Bachem. Warp: On the benefits of weight averaged rewarded policies. *arXiv preprint arXiv:2406.16768*, 2024.

Rylan Schaeffer, Joshua Kazdan, John Hughes, Jordan Juravsky, Sara Price, Aengus Lynch, Erik Jones, Robert Kirk, Azalia Mirhoseini, and Sanmi Koyejo. How do large language monkeys get their power (laws)? In *Forty-second International Conference on Machine Learning*, 2025.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

Rulin Shao, Shuyue Stella Li, Rui Xin, Scott Geng, Yiping Wang, Sewoong Oh, Simon Shaolei Du, Nathan Lambert, Sewon Min, Ranjay Krishna, Yulia Tsvetkov, Hannaneh Hajishirzi, Pang Wei Koh, and Luke Zettlemoyer. Spurious rewards: Rethinking training signals in RLVR. In *Forty-third International Conference on Machine Learning*, 2026.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Jingyan Shen, Ang Li, Salman Rahman, Yifan Sun, Micah Goldblum, Matus Telgarsky, and Pavel Izmailov. Understanding reasoning from pretraining to post-training. *arXiv preprint arXiv:2607.16097*, 2026a.

Minghe Shen, Zhuo Zhi, Chonghan Liu, Shuo Xing, Zhengzhong Tu, and Che Liu. Does RLVR extend reasoning boundaries? investigating capability expansion in vision-language models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, San Diego, California, United States, July 2026b. Association for Computational Linguistics.

Joar Skalse, Nikolaus Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward gaming. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 9460–9471. Curran Associates, Inc., 2022. doi: 10.52202/068431-0687. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/3d719fee332caa23d5038b8a90e81796-Paper-Conference.pdf.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025.

Yuda Song, Julia Kempe, and Remi Munos. Outcome-based exploration for llm reasoning. *arXiv preprint arXiv:2509.06941*, 2025.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in neural information processing systems*, 33:3008–3021, 2020.

Liangcai Su, Zhen Zhang, Guangyu Li, Zhuo Chen, Chenxi Wang, Maojia Song, Xinyu Wang, Kuan Li, Jialong Wu, Xuanzhong Chen, et al. Scaling agents via continual pre-training. In *International Conference on Learning Representations*, volume 2026, pp. 22634–22659, 2026.

Yiyou Sun, Yuhan Cao, Pohao Huang, Haoyue Bai, Hanna Hajishirzi, Nouha Dziri, and Dawn Song. Rl grokking recipe: How does rl unlock and transfer new algorithms in llms? In *International Conference on Learning Representations*, volume 2026, pp. 93190–93220, 2026.

Zhiqing Sun, Sheng Shen, Shengcao Cao, Haotian Liu, Chunyuan Li, Yikang Shen, Chuang Gan, Liangyan Gui, Yu-Xiong Wang, Yiming Yang, et al. Aligning large multimodal models with factually augmented rlhf. In *Findings of the association for computational linguistics: ACL 2024*, pp. 13088–13110, 2024.

Fahim Tajwar, Guanning Zeng, Yueer Zhou, Yuda Song, Daman Arora, Yiding Jiang, Jeff Schneider, Ruslan Salakhutdinov, Haiwen Feng, and Andrea Zanette. Maximum likelihood reinforcement learning. *arXiv preprint arXiv:2602.02710*, 2026.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

Christian Walder and Deep Tejas Karkhanis. Pass@ k policy optimization: Solving harder reinforcement learning problems. *Advances in Neural Information Processing Systems*, 38:152416–152445, 2025.

Guoqing Wang, Zeyu Sun, Sixiang Ye, Zhihao Gong, Yizhou Chen, Yifan Zhao, Qingyuan Liang, and Dan Hao. Do advanced language models eliminate the need for prompt engineering in software engineering? *ACM Transactions on Software Engineering and Methodology*, 35(8):1–33, 2026a.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 50208–50232. PMLR, 21–27 Jul 2024.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Yu, Kefan Tang, Minh Nhat Liu, Anima Anandkumar, et al. RAGEN: Understanding self-evolution in LLM agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025.

Zihan Wang, Chi Gui, Xing Jin, Qineng Wang, Licheng Liu, Kangrui Wang, Shiqi Chen, Linjie Li, Zhengyuan Yang, Pingyue Zhang, et al. Ragen-2: Reasoning collapse in agentic rl. *arXiv preprint arXiv:2604.06268*, 2026b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Xumeng Wen, Zihan Liu, Shun Zheng, Shengyu Ye, Zhirong Wu, Yang Wang, Zhijian Xu, Xiao Liang, Junjie Li, Ziming Miao, et al. Reinforcement learning with verifiable rewards implicitly incentivizes correct reasoning in base llms. In *International Conference on Learning Representations*, volume 2026, pp. 49450–49483, 2026.

Mitchell Wortsman, Gabriel Ilharco, Jong Wook Kim, Mike Li, Simon Kornblith, Rebecca Roelofs, Raphael Gontijo Lopes, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, et al. Robust fine-tuning of zero-shot models. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7949–7961. IEEE, 2022.

Fang Wu, Weihao Xuan, Ximing Lu, Mingjie Liu, Yi Dong, Zaid Harchaoui, and Yejin Choi. The invisible leash: Why rlvr may or may not escape its origin. *arXiv preprint arXiv:2507.14843*, 2025.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*, 2024.

xAI. Introducing Grok Build. xAI News, 2026. URL https://x.ai/news/grok-build-cli.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv preprint arXiv:2504.08066*, 2025.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024a.

Chenghao Yang, Lin Gui, Chenxiao Yang, Victor Veitch, Lizhu Zhang, and Zhuokai Zhao. Let it calm: Exploratory annealed decoding for verifiable reinforcement learning. *arXiv preprint arXiv:2510.05251*, 2025.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024b.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244, 2025.

Jiayi Yuan, Hangoo Kang, James Jihao Liu, Yejin Choi, Vikram Iyer, Liwei Jiang, and Natasha Jaques. Forty shades of blue: Quality-diversity alignment via mode-conditioned reinforcement learning. *arXiv preprint arXiv:2609.14896*, 2026a.

Lifan Yuan, Wendi Li, Huayu Chen, Ganqu Cui, Ning Ding, Kaiyan Zhang, Bowen Zhou, Zhiyuan Liu, and Hao Peng. Free process rewards without process labels. *arXiv preprint arXiv:2412.01981*, 2024.

Lifan Yuan, Weize Chen, Yuchen Zhang, Ganqu Cui, Hanbin Wang, Ziming You, Ning Ding, Zhiyuan Liu, Maosong Sun, and Hao Peng. From f (x) and g (x) to f (g (x)): Llms learn new skills in rl by composing old ones. In *International Conference on Learning Representations*, volume 2026, pp. 147547–147574, 2026b.

Mert Yuksekgonul, Daniel Koceja, Xinhao Li, Federico Bianchi, Jed McCaleb, Xiaolong Wang, Jan Kautz, Yejin Choi, James Zou, Carlos Guestrin, and Yu Sun. Learning to discover at test time. In *Forty-third International Conference on Machine Learning*, 2026.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3053–3077, 2024.

Charlie Zhang, Graham Neubig, and Xiang Yue. On the interplay of pre-training, mid-training, and rl on reasoning language models. *arXiv preprint arXiv:2512.07783*, 2025.

Rosie Zhao, Alexandru Meterez, Sham Kakade, Cengiz Pehlevan, Samy Jelassi, and Eran Malach. Echo chamber: Rl post-training amplifies behaviors learned in pretraining. *arXiv preprint arXiv:2504.07912*, 2025.

Todd Zhou. When rlvr shrinks the reasoning boundary: Diagnosing pass@ k inversion. *arXiv preprint arXiv:2607.20543*, 2026.

Appendix

A Related Work **19**

B Limitations and Future Work **19**

C Experimental Details **20**

C.1 Benchmarks and Environments 20

C.2 Model Checkpoints and Serving 20

C.3 Harness Design for Base Models 22

C.4 Metrics and Uncertainty Estimation 23

C.5 PTGS Training and Evaluation Setup 23

C.6 Pseudocode 25

D Theoretical Analysis **26**

D.1 What the Sharpening Tax Measures 26

D.2 Ceiling and Saturation Decomposition of the Tax 27

D.3 Why Sharpening Charges a Tax 27

D.4 How PTGS Improves Group-Based RL 28

E Additional Results on the Accuracy–Coverage Tension **29**

E.1 Aggregate Results by Backbone Scale 30

E.2 Results by Model Family and Checkpoint Pair 31

E.3 Per-Task Scaling Curves and Their Density on WebShop 39

E.4 Global Temperature Scaling Cannot Resolve the Trade-off 40

F Additional Results on the Sharpening Tax **40**

F.1 Sharpening Tax for All Fourteen Checkpoint Pairs 41

F.2 Decomposition of the Tax by Task Movement 42

F.3 Which Tasks Pay the Tax? 45

F.4 Tax-Guided Routing between the Base and the Post-trained Policy 51

G Additional Results on PTGS **52**

G.1 Training Dynamics of PTGS 52

G.2 Ablation on the Temperature Spread τ of PTGS 52

G.3 Fixed Rollout-Temperature and Decoding Ablations of the PPO Baseline 54

G.4 Inference-Time PTGS Application 54

H Deeper Analysis of PTGS **55**

H.1 Accuracy of the Difficulty Estimate and Realized Temperatures 55

H.2 Does PTGS Induce a More Exploratory Policy? 56

H.3 Case Study: Tasks That PPO Loses and PTGS Recovers 57

A Related Work

Does RL post-training expand or sharpen the reasoning boundary? Whether RL post-training extends the reasoning boundary of the base model is under active debate. One line of work observes that RL-tuned models lose to their base counterparts in pass@K at large K, suggesting that post-training sharpens the policy around behaviors the base model already possesses (Chen et al., 2025b; Zhao et al., 2025; Wu et al., 2025). Another line argues that the boundary can genuinely expand, e.g., through prolonged RL training (Liu et al., 2025a) or grokking (Sun et al., 2026), while others attribute the outcome to properties of the training data and task distribution (Zhang et al., 2025; Shen et al., 2026a; Shao et al., 2026). However, the evidence on both sides comes almost entirely from mathematics and coding (Chen et al., 2025b; He et al., 2025), where the skills required for realistic agentic tasks are not assessed, and pass@K can be inflated by lucky final answers (Wen et al., 2026; Dragoi et al., 2025). We bring this debate to agentic environments and give it a quantitative diagnosis, Sharpening Tax, which turns manual inspection of test-time scaling visualizations into a measurable and predictable quantity.

Preserving diversity in post-training. Alongside with the investigation on the sharpening hypothesis of post-trained policy, preventing RL post-training from collapsing policy diversity has been a popular topic of research. Common objective-side solutions include exploration bonuses and mitigation of entropy-collapse (Yu et al., 2025; He et al., 2025; Song et al., 2025; Gai et al., 2025) and pass@k-aware learning objectives (Walder & Karkhanis, 2025; Tajwar et al., 2026). Another line is the sampling-side solutions adjusting the rollout distribution through dynamic temperature scheduling across training (Yang et al., 2025; Liao et al., 2025; Dang et al., 2026), or estimating power distribution (Karan & Du, 2026). While belonging to this sampling-side solution, our PTGS distinguishes itself from others by pursuing difficulty-guided adaptation of temperature for each prompt via a Beta posterior over its success rate; it requires no change to the existing RL algorithm, thereby applying to most of the modern RL pipelines on the fly.

Test-time parallel scaling via repeated sampling. Repeated sampling has been established as a reliable axis of test-time compute where solution coverage grows smoothly with the number of rollouts (Brown et al., 2024; Muennighoff et al., 2025), following predictable inference-time scaling laws (Wu et al., 2024; Schaeffer et al., 2025), and can be gathered with verifiers or best-of-N selection (Christiano et al., 2017; Stiennon et al., 2020; Gao et al., 2023; Luo et al., 2024; Snell et al., 2025; Kwok et al., 2026), where the standard unbiased estimators of pass@k (Chen et al., 2021) has been central these analyses. While this line of work treats repeated sampling as a way to gain performance, we instead repurpose it as a diagnostic tool, using the shape of the scaling curve itself rather than its endpoints to reveal what post-training did to the policy.

B Limitations and Future Work

Note that the representative open-source models that we extensively analyze in §3 do not disclose the exact data used in pre-training and post-training, so our analysis remains observational rather than interventional. Replicating this in a controlled setup (Zhao et al., 2025; Shen et al., 2026a) where both data mixes are fully known would allow a causal account of when and how sharpening emerges exactly. Second, although the Sharpening Tax does not depend on the exact post-training recipe, we did not compare how different types of post-training sharpen the policy. Investigating how RL, classic supervised fine-tuning based on conditional likelihood modeling, and on-policy distillation (Hinton et al., 2015; Gu et al., 2023; Agarwal et al., 2024; Lu, 2025; Heo et al., 2026) charge taxes differently is a natural next step. Third, our scope is text-based LLM agents. Whether the same accuracy-consistency-coverage trilemma holds and whether the benefits of PTGS carry over to post-training beyond pure LLMs, including multimodal LLMs and vision-language-action models, remains an open question (Sun et al., 2024; Kwok et al., 2025; Shen et al., 2026b; Jeddi et al., 2026; Li et al., 2026a). Moreover, Sharpening Tax relies on a binary success signal; extending it to open-ended generation, where alignment likewise collapses output diversity (Yuan et al., 2026a), would require replacing pass@k with quality-gated diversity measures. Finally, given the diverse applications of base and post-trained policy pairs spanning contrastive decoding (Li et al., 2023; O'Brien & Lewis, 2023), model merging (Wortsman et al., 2022; Ramé et al., 2024; Oh et al., 2025), and process reward modeling (Yuan et al., 2024; Oh et al., 2026a), characterizing how sharpening reshapes these signals and quantities would be valuable beyond test-time scaling diagnosis.

C Experimental Details

In this section, we describe the settings needed to reproduce our experiments. For the sharpening analysis in §3 and §4, we describe the benchmarks (§C.1), the model pairs and serving stack (§C.2), the harness for base models (§C.3), metric estimation and uncertainty quantification (§C.4), and the pseudocode of the tax computation (§C.6). For PTGS (§5), we describe the training and evaluation setup (§C.5) and provide its pseudocode (§C.6).

C.1 Benchmarks and Environments

Notably, all three benchmarks we consider are process-centric. That is, task success is determined by the environment state that the model reaches through its intermediate actions, not by matching a final answer string. Therefore, unlike some math and STEM benchmarks (Wen et al., 2026; Shao et al., 2026; Rahman et al., 2026), a flawed trajectory cannot get credit from a lucky final answer.

BFCL multi-turn. We use the `multi_turn_base` category of the Berkeley Function-Calling Leaderboard v4 (Patil et al., 2025). A rollout passes if and only if the native BFCL multi-turn checker returns `valid`. This checker compares the environment state produced by the tool calls of the model with the reference end state, so it also grades intermediate actions. Following BFCL, we allow at most 20 tool-call rounds per user turn. We report this single category rather than the full 800-task multi-turn suite with four categories, because the categories differ substantially in difficulty and the comparison between the base and post-trained models can flip from one category to another. Fixing one category avoids averaging over such different regimes.

WebShop. We evaluate on the first 500 goal instructions of WebShop (Yao et al., 2022) with a 1,000-product index, and each episode is limited to 30 environment actions. The model takes one action per turn, either `search[<keywords>]` or `click[<button>]`. Each observation is a text rendering of the page, with UI elements separated by `[SEP]` special token, followed by the list of available actions. Although WebShop environment produces a dense reward in $[0, 1]$ that also credits partial attribute matches, we count an episode as a success only when the final reward is 1.0, i.e., a fully matching purchase, following the common evaluation protocol (Chen et al., 2026).

ACEBench. We use the official English version of ACEBench (Chen et al., 2025a) with its per-category system prompts and default verifier. We use 770-task subset that combines single-turn general tool calling and multi-step problem solving (750 tasks) with multi-turn agentic tasks (20 tasks). Due to budget constraints, we exclude the tasks that require an LLM-based user simulator.

BFCL and WebShop are widely used benchmarks of basic agentic capabilities such as multi-turn tool calling and environment interaction. ACEBench tests similar skills but is more recent and less commonly reported in model development, so it is less likely to have been optimized for during post-training of the considered model families. We also chose benchmarks on which both base and post-trained models reach non-trivial success within our budget, since pass@k curves that stay near zero for both policies reveal little about sharpening.

C.2 Model Checkpoints and Serving

Checkpoint pairs. We use open-source base/post-trained checkpoint pairs (Table 3) from four model families, Gemma-4[1], Ministral-3[2], Qwen2.5[3], and Qwen3.5[4], available in huggingface collections.

On the term "RL post-trained model". The post-training recipes of these open checkpoints are not fully disclosed, and the `-it`/`-Instruct` models may combine SFT, preference optimization, and RL with verifiable rewards in unknown proportions. However, our analysis does not depend on the exact recipe. SFT and DPO also sharpen the policy distribution toward preferred behaviors (Huang et al., 2025); even knowledge distillation does the same (Cha & Cho, 2026). Here, Sharpening Tax measures the effect of post-training, i.e., the loss of test-time scalability relative to the base checkpoint, rather than its mechanism. Therefore, we use "RL post-trained model" as a shorthand

[1] https://huggingface.co/collections/google/gemma-4
[2] https://huggingface.co/collections/mistralai/ministral-3
[3] https://huggingface.co/collections/Qwen/qwen25
[4] https://huggingface.co/collections/Qwen/qwen35

Table 3: **The 14 base/post-trained checkpoint pairs used in our analysis**, ranging from 3B to 35B parameters. `gemma-4-26B-A4B` and `Qwen3.5-35B-A3B` are mixture-of-experts models with about 4B and 3B active parameters, respectively.

#	Base checkpoint	Post-trained checkpoint	Family
1	`google/gemma-4-E4B`	`google/gemma-4-E4B-it`	Gemma-4
2	`google/gemma-4-12B`	`google/gemma-4-12B-it`	Gemma-4
3	`google/gemma-4-26B-A4B`	`google/gemma-4-26B-A4B-it`	Gemma-4
4	`google/gemma-4-31B`	`google/gemma-4-31B-it`	Gemma-4
5	`mistralai/Ministral-3-3B-Base-2512`	`mistralai/Ministral-3-3B-Instruct-2512`	Ministral-3
6	`mistralai/Ministral-3-8B-Base-2512`	`mistralai/Ministral-3-8B-Instruct-2512`	Ministral-3
7	`mistralai/Ministral-3-14B-Base-2512`	`mistralai/Ministral-3-14B-Instruct-2512`	Ministral-3
8	`Qwen/Qwen2.5-3B`	`Qwen/Qwen2.5-3B-Instruct`	Qwen2.5
9	`Qwen/Qwen2.5-7B`	`Qwen/Qwen2.5-7B-Instruct`	Qwen2.5
10	`Qwen/Qwen2.5-14B`	`Qwen/Qwen2.5-14B-Instruct`	Qwen2.5
11	`Qwen/Qwen2.5-32B`	`Qwen/Qwen2.5-32B-Instruct`	Qwen2.5
12	`Qwen/Qwen3.5-4B-Base`	`Qwen/Qwen3.5-4B`	Qwen3.5
13	`Qwen/Qwen3.5-9B-Base`	`Qwen/Qwen3.5-9B`	Qwen3.5
14	`Qwen/Qwen3.5-35B-A3B-Base`	`Qwen/Qwen3.5-35B-A3B`	Qwen3.5

for all post-trained models, reflecting that RL has become the central and most compute-intensive stage of modern post-training pipelines (Guo et al., 2025; Olmo Team, 2025; Blakeman et al., 2025; Microsoft AI Team, 2026).

Inference configurations. All models are served with vLLM 0.19.0 (Kwon et al., 2023). Base models receive a raw text prompt without any chat template, whereas post-trained models are queried with the native chat template and tool-call parser of each family (e.g., `gemma4` for Gemma-4 and `mistral` for Ministral-3). Meanwhile, sampling parameters are fixed for each benchmark and shared by base and post-trained models, without per-model tuning (Table 5). We use temperature 0.7 for WebShop and ACEBench and 0.4 for BFCL, since the structured tool calls on BFCL were parsed less reliably at 0.7. Some post-trained models support an optional thinking mode, while others do not. For a fair comparison across model families, we turn it off for all post-trained models that support it by passing `chat_template_kwargs = {"enable_thinking": false}` in every chat-completion request. Another reason is that our budget counts rollouts, not tokens. With thinking mode, post-trained models would use extra test-time compute in each rollout, which would bias the coverage comparison against the base model. We also confirmed that this choice does not change our conclusion. As shown in Figure 9, although turning on the thinking mode improves pass@k of post-trained models, the overall trend stays the same, i.e., base models still catch up the post-trained ones at the end while paying a positive tax within the considered budget range.

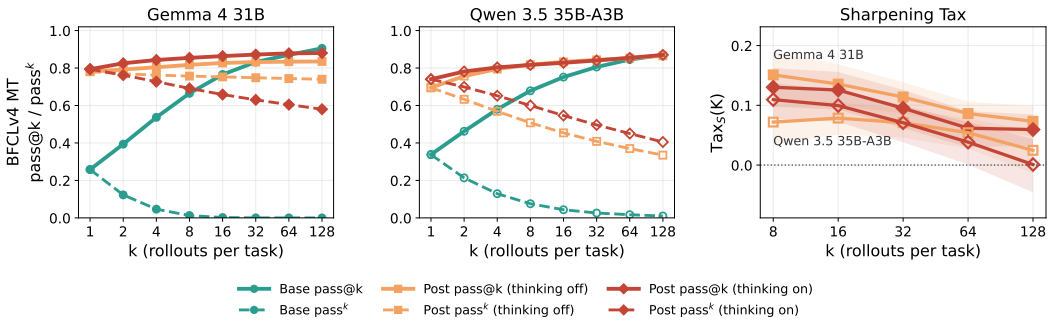


Figure 9: **Ablation on thinking mode.** Coverage (pass@k) and consistency (pass^k) curves of the base and post-trained models of `gemma-4-31B` (left) and `Qwen3.5-35B-A3B` (middle) on BFCLv4 MT, with the thinking mode of the post-trained model turned off (our default setup) and on, and the corresponding $\mathrm{Tax}_S(k)$ (right), as a function of the rollout budget k per task. Thinking mode improves the accuracy and coverage of the post-trained models, but it still doesn't change the bold conclusion—base models still catch up at large k, and the tax remains non-negative.

C.3 HARNESS DESIGN FOR BASE MODELS

````
You are a helpful assistant that can call tools.

# Available tools
```python
def get_weather(
    city: str, unit: Literal["celsius", "fahrenheit"] = "celsius"
) -> dict:
    """Get the current weather for a city.

    Args:
        city (str, required): City name.
        unit (Literal["celsius", "fahrenheit"], optional):
            Temperature unit. choices: ['celsius', 'fahrenheit']
    """
    ...
'''

# How to call tools
To call one or more tools, emit a fenced ```tool_call``` block
containing a JSON array of calls. [...] When the task is complete,
reply in plain text with no ```tool_call``` block.

### User
What's the weather in Palo Alto?

### Assistant
```tool_call
[{"name": "get_weather", "arguments": {"city": "Palo Alto"}}]
'''

### Tool Output
[get_weather] -> {"temp_c": 21, "sky": "sunny"}

### Assistant
````

A pre-trained base checkpoint has no chat template and no tool-call format, but the default benchmark runners expect the OpenAI-style tooling interface with `messages` and `tools` arguments. Our harness serves as an adapter between the two. It renders the conversation into a plain-text prompt for the completion endpoint and parses the free-text completion back into structured tool calls. Since our goal is not state-of-the-art performance of base models, we use **a simple, lightweight universal harness for every base model and every benchmark**, without per-model presets, per-family calibration, or few-shot demonstrations. The prompt explains the tool-call format with placeholder names but never shows a solved task, so the harness does not leak any task information.

Prompt construction. The prompt consists of five parts in the following order. (1) The system prompt of the benchmark. (2) The tool catalog, where each tool is written as a Python function signature with a short docstring that describes each parameter, its type, and whether it is required, similar to Yang et al. (2024b) and Wang et al. (2024). (3) An instruction on how to call tools, i.e., by writing a fenced `tool_call` block that contains a JSON array of calls. (4) The conversation so far, under the section markers `### User`, `### Assistant`, and `### Tool Output`. (5) A trailing `### Assistant` header, so that the completion of the model starts inside its own turn. We mix the two formats (Python and JSON) on purpose. In preliminary experiments, we found that base models read tool definitions best as Python function signatures but write JSON most reliably. Therefore, we let the model read Python and write JSON, and keep the two consistent by using the parameter names of the function signature as the argument keys of the JSON call. The colored box above shows the exact prompt text that the harness produces for a simple episode with one tool, one user request, and one completed tool-call round.

Stop sequences and episode termination. Unlike an instruction-tuned model, a base model has no special token that marks the end of its turn. Without a guard, it keeps writing and eventually makes up the response of the environment by itself. To prevent this, we use three stop sequences. The first two,

`"### Tool Output"` and `"### User"`, stop the model as soon as it starts writing a section that belongs to the environment or the user. The third, `"\n ”’ "`, stops the model right after it closes its first tool-call block. Well-behaved models already stop there by themselves, and this rule only prevents a rare failure in which a model repeats the same block until the token limit. An episode ends when the model replies without a `tool_call` block, and we take this reply as its final answer.

Asymmetric scaffolding. Throughout the paper, we evaluate base models with the harness and post-trained models with the default scaffolding of each benchmark, so that each checkpoint is used in the way it would be deployed. Table 4 gives the full ablation. For base models, the harness is essential on BFCL, where six of the eight base checkpoints score zero without it. The two Qwen3.5 base models are the exception, since they already produce parseable native tool calls without the harness. For them, the harness helps the 35B-A3B model but hurts the 4B model on BFCL. On ACEBench, the harness leaves the base models largely unchanged. For post-trained models, the harness lowers performance on average, especially pass@1, which is why we keep their default scaffolding.

Harness ablation protocol. Since a base checkpoint has no chat template, we define the "w/o harness" setting of Tables 1 and 4 in the following way. We serve the base checkpoint with the chat template of its post-trained counterpart, enable the native tool-call parser, and send the requests through the same chat-completion endpoint as for the post-trained models. In this ablation, only the prompt and parsing interface change, while the weights, tasks, scoring, seed, rollout budget ($N = 32$), and decoding parameters of each benchmark are kept fixed.

C.4 Metrics and Uncertainty Estimation

Estimation. We compute all metrics of §2 for each task from its success count c_i out of n_i rollouts with the unbiased estimators of Chen et al. (2021), and then average them over tasks with equal weight. For numerical stability, we implement the estimators with cumulative products instead of binomial coefficients. Each model uses all of its own N rollouts.

Bootstrap confidence intervals. The 95% bands in Figure 6 are percentile intervals from a paired task bootstrap with 1,000 resamples. In each resample, we draw tasks with replacement and use the same draw for both models, so that the interval of the paired difference accounts for the correlation of task difficulty between the two models. We resample tasks rather than rollouts because we want to know whether the tax would remain under a different set of evaluation tasks. Note that the estimate of pass@K at $K = N$ reduces to a single per-task indicator of whether any rollout succeeded, without averaging over subsets, so the last point of each scaling curve is a noisier estimate.

C.5 PTGS Training and Evaluation Setup

Model and environments. All PTGS experiments fine-tune `Qwen2.5-7B-Instruct` in the RAGEN framework (Wang et al., 2025; 2026b) on two interactive environments. FrozenLake uses the slippery variant (each move goes in the intended direction with probability 0.8 and otherwise slips to one of the two perpendicular directions with probability 0.1 each) with a binary success reward. Sokoban uses a 6×6 grid with one box with a shaped reward, and an episode counts as a success once the box is on its target. Each episode allows at most 5 turns with up to 2 actions per turn.

Training setup. We train PPO and GRPO, each with and without PTGS, for 200 StarPO steps (Wang et al., 2025). Each step samples 8 tasks from a fixed pool of 200 tasks and generates 16 rollouts per task, i.e., 128 trajectories per step. All methods use AdamW (Loshchilov & Hutter, 2017) with a constant actor learning rate of 10^{-6}, DAPO-style (Yu et al., 2025) asymmetric clip ratios of 0.2 (lower) and 0.28 (upper), token-mean loss aggregation, no KL penalty, and a minibatch size of 32 with one epoch per step. The StarPO-S verision 2 reward-variance filter (Wang et al., 2026b) keeps the top 90% of prompt groups ranked by reward variance. Both baselines sample rollouts ancestrally at a fixed temperature $T_{\text{ref}} = 1.0$, and PTGS replaces this fixed temperature with a per-prompt temperature.

PPO uses generalized advantage estimation (GAE; Schulman et al. (2015)) with both the discount factor and the GAE parameter set to 1, a critic learning rate of 10^{-5}, and an entropy coefficient of 0.001. GRPO (Shao et al., 2024) uses group-relative advantages without a critic and an entropy coefficient of 0. Within each algorithm, the baseline and PTGS use the same recipe except for the rollout temperature and the matching temperature used to compute the policy log-probabilities.

Table 4: **Harness ablation for base and post-trained models.** pass@1 and pass@32 (%) of eight backbones with (w/) and without (w/o) the harness, using $N = 32$ rollouts per task and the same tasks, seed, temperature, and step budget in all four settings. Δp@32 is the pass@32 gain from the harness (w/ minus w/o), and the larger value of each w/o–w/ pair is in bold. For base models, "w/o harness" serves the checkpoint with the chat template and native tool-call parser of its post-trained counterpart. For post-trained models, "w/ harness" sends the plain-text harness prompt through the completion endpoint, without the chat template or the native parser.

		Base model					Post-trained model				
		w/o harness		w/ harness			w/o harness		w/ harness		
Backbone	Benchmark	p@1	p@32	p@1	p@32	Δp@32	p@1	p@32	p@1	p@32	Δp@32
`gemma-4-E4B`	BFCL	0.0	0.0	**2.2**	**21.0**	+21.0	**23.7**	**38.0**	12.0	21.0	-17.0
	WebShop	**1.4**	**16.8**	0.9	10.2	-6.6	**32.4**	**56.2**	4.1	22.8	-33.4
	ACEBench	23.8	77.6	**30.8**	**77.7**	+0.1	**81.4**	**89.1**	34.0	68.1	-20.9
`gemma-4-31B`	BFCL	0.0	0.0	**32.8**	**86.5**	+86.5	**78.0**	**83.0**	7.5	21.5	-61.5
	WebShop	3.3	38.8	**18.2**	**78.4**	+39.6	**40.3**	52.0	20.2	**54.4**	+2.4
	ACEBench	35.8	**96.9**	**54.8**	92.0	-4.9	**89.7**	**91.5**	57.0	87.1	-4.4
`Ministral-3-8B`	BFCL	0.0	0.0	**8.3**	**40.0**	+40.0	**31.7**	**59.0**	7.8	26.0	-33.0
	WebShop	1.5	27.6	**6.1**	**51.4**	+23.8	**14.7**	**44.8**	11.9	31.4	-13.4
	ACEBench	**47.0**	**93.9**	38.7	88.0	-5.9	**74.9**	**90.0**	56.8	89.1	-0.9
`Ministral-3-14B`	BFCL	0.0	0.0	**13.2**	**47.0**	+47.0	**33.6**	**61.5**	17.2	40.5	-21.0
	WebShop	4.8	45.4	**8.4**	**53.4**	+8.0	28.5	66.2	**30.1**	**67.0**	+0.8
	ACEBench	**56.3**	**94.0**	46.1	92.5	-1.5	**78.9**	92.4	70.1	**92.9**	+0.5
`Qwen2.5-3B`	BFCL	0.0	0.0	**0.8**	**6.3**	+6.3	**11.5**	**27.5**	2.6	8.5	-19.0
	WebShop	0.0	1.2	**0.1**	**2.5**	+1.3	3.1	9.4	**5.9**	**21.8**	+12.4
	ACEBench	15.7	70.4	**21.3**	**70.7**	+0.3	**48.4**	**67.5**	37.1	65.6	-1.9
`Qwen2.5-32B`	BFCL	0.0	0.0	**19.1**	**59.6**	+59.6	**35.5**	**59.0**	13.4	22.0	-37.0
	WebShop	8.9	61.4	**20.6**	**74.7**	+13.3	33.8	54.2	**34.6**	**57.8**	+3.6
	ACEBench	47.0	**95.2**	**51.3**	90.4	-4.8	81.8	87.6	**83.2**	**93.7**	+6.1
`Qwen3.5-4B`	BFCL	**25.9**	**60.5**	14.9	52.0	-8.5	**58.9**	**83.5**	21.7	69.0	-14.5
	WebShop	**7.1**	50.6	6.3	**51.0**	+0.4	**32.1**	61.6	21.7	**64.0**	+2.4
	ACEBench	**58.3**	88.5	45.4	**93.5**	+5.0	**71.6**	86.9	45.3	**90.5**	+3.6
`Qwen3.5-35B-A3B`	BFCL	18.9	61.0	**33.8**	**80.5**	+19.5	**69.8**	84.0	48.5	**88.0**	+4.0
	WebShop	**23.5**	**74.0**	17.2	71.7	-2.3	**32.9**	**69.6**	17.6	66.2	-3.4
	ACEBench	**69.9**	**97.5**	52.6	96.6	-0.8	**83.1**	**94.4**	42.8	87.7	-6.7
Average (8)	BFCL	5.6	15.2	**15.6**	**49.1**	+33.9	**42.8**	**61.9**	16.3	37.1	-24.9
	WebShop	6.3	39.5	**9.7**	**49.2**	+9.7	**27.2**	**51.8**	18.3	48.2	-3.6
	ACEBench	**44.2**	**89.2**	42.6	87.7	-1.6	**76.2**	**87.4**	53.3	84.4	-3.1

Table 5: **Decoding configuration for each benchmark**, shared by the base and post-trained models of all 14 checkpoint pairs. Every model-benchmark combination is evaluated with $N = 128$ rollouts per task.

	BFCLv4 multi-turn	WebShop	ACEBench
temperature	0.4	0.7	0.7
top_p	0.95	0.95	0.95
max tokens per generation	1024	256	1200
context length	16384 / 32768	16384	16384 / 32768
rollouts per task (N)	128	128	128
reported K grid of pass@K	$1, 2, 4, 8, 16, 32, 64, 128$		

PTGS configuration. For PPO, we use the same PTGS setting in both environments, with $\tau = 1.5$ (so $T_x \in [0.667, 1.5]$) and a pivot $\tilde{p}$ that grows from 0.25 to 0.5 over the 200 steps by a constant factor per step. The final pivot of 0.5 is the success probability at which a rollout group is most likely to contain both successes and failures, i.e., where the within-group reward variance $p(1-p)$ peaks (Theorem 4). Starting from a lower pivot heats fewer prompts early in training, when most prompts still have low success rates. Appendix G.2 ablates $\tau \in \{1.2, 1.3, 1.4, 1.5\}$ for PPO. For GRPO, we

select the PTGS parameter based on peak validation success rate for each environment, which gives $\tau = 1.4$ with a fixed pivot of 0.5 on Sokoban and $\tau = 1.2$ with a pivot growing from 0.25 to 0.5 on FrozenLake. In all PTGS runs, the forgetting factor is $\gamma = 0.95$, the Beta prior has total mass 2 and is centered on the current pivot, and $\hat{p}_x$ is drawn from the posterior of each task by Thompson sampling. A rollout counts as a success when its trajectory return exceeds 0.5. We draw one temperature per prompt group and use it for every turn of every rollout in that group.

Implementation details. We denote T_{ref} as a reference temperature for an environment. It is 1.0 by default, so we omit it in §5. In some cases, the environment decoding temperature departs from 1.0, one can set T_{ref} to that environment-default temperature (e.g., §G.4). The tempering rule in Eq. (4) passes through three points, $T(0) = \tau T_{\text{ref}}$, $T(\tilde{p}) = T_{\text{ref}}$, and $T(1) = T_{\text{ref}}/\tau$, and it reduces to the symmetric rule $T = T_{\text{ref}}\,\tau^{\,1-2\hat{p}_x}$ when $\tilde{p} = 1/2$.

Checkpoint selection. For PPO, Table 2 reports the final checkpoint at step 200. For GRPO, whose validation success peaks before step 200 and then sometimes collapses (which is common in multi-turn RL setups Li et al. (2026b); Wang et al. (2025; 2026b)), both the baseline and PTGS use the checkpoint with the highest validation success during training (marked with † in the table).

Offline evaluation and repeated runs. In all the evaluations in the main paper, we evaluate each checkpoint on 64 unseen tasks with 128 ancestral rollouts per task at $T = 0.5$, without PTGS at test time (except in §G.4 where we introduce the inference-time PTGS variant). We compute pass@k for $k \leq 128$ with the estimators of §2, and $\text{Tax}_S(128)$ for all RL-tuned models against a single shared evaluation of the base model in each environment. Table 2 reports the mean over five runs of each RL setting.

C.6 Pseudocode

Listing 1: Computing the Sharpening Tax metrics of §4.

```
def pass_at_k(c, n, k):
    return 1 - comb(n - c, k) / comb(n, k)

def scalability(counts, K):  # counts: [(c_i, n_i) for each task in D]
    curve = [mean([pass_at_k(c, n, k) for (c, n) in counts]) for k in
    range(1, K + 1)]
    A = sum(curve[-1] - curve[k] for k in range(K - 1))
    S = A / ((K - 1) * (1 - curve[0]))
    return A, S

def sharpening_tax(counts_base, counts_rl, K):
    A_b, S_b = scalability(counts_base, K)
    A_r, S_r = scalability(counts_rl, K)
    return A_b - A_r, S_b - S_r
```

Listing 1 shows how we compute the Sharpening Tax of §4. Given the per-task success counts of a policy, we first compute the dataset-level pass@k curve for $k = 1, \ldots, K$. We then sum the gaps between pass@K and pass@k to obtain $A(K)$ (Eq. (1)) and normalize it by the budget and the single-attempt failure rate with total computation budget to obtain $S(K)$ (Eq. (2)). Finally, the differences between the base and post-trained policies on the same task set give $\text{Tax}_A(K)$ and $\text{Tax}_S(K)$ (Eq. (3)). For clarity, the listing omits the cumulative-product implementation of the estimator (§C.4) and the paired task bootstrap for confidence intervals.

PTGS. Listing 2 gives the pseudocode of PTGS (§5). For each prompt x, we keep discounted success and failure counts $(\tilde{s}_x, \tilde{f}_x)$. With a prior of total mass 2 centered on the pivot $\tilde{p}$, these counts define a Beta posterior over the current success rate of the prompt. Before each rollout group, we draw $\hat{p}_x$ from this posterior by Thompson sampling, map it to a temperature $T_x \in [1/\tau, \tau]$ with Eq. (4), generate the group at T_x, and add the observed successes and failures to the counts with the forgetting factor γ. The log-probabilities in the policy-gradient ratio are computed at the same temperature, so the update remains on-policy. Algorithm 1 in the main text summarizes the same procedure.

Listing 2: PTGS within an RL training loop (Algorithm 1).

```
s, f = zeros(len(D)), zeros(len(D))

for step in range(num_rl_steps):
    batch = sample_tasks(D)
    for x in batch:
        a, b = 2 * pivot + s[x], 2 * (1 - pivot) + f[x]
        p = Beta(a, b).sample()
        h = (pivot - p) / (pivot if p <= pivot else 1 - pivot)
        T[x] = tau ** h

        group = policy.generate(x, n=group_size, temperature=T[x])
        s[x] = gamma * s[x] + num_success(group)
        f[x] = gamma * f[x] + group_size - num_success(group)

    rl_update(batch, logprobs=policy.log_probs(batch, temperature=T))
```

D Theoretical Analysis

To explain the empirical findings of §3–§5 and to support our method, we present a few simple theoretical results, each followed by its proof. First, §D.1 provides a probabilistic interpretation of what the tax measures (Proposition 1), and §D.2 decomposes the tax into a coverage-ceiling gap and a saturation gap (Corollary 2). Building on these, §D.3 shows that sharpening charges a tax when it collapses tasks to the two extremes (Theorem 3), and §D.4 explains how the adaptive smoothing of PTGS improves RL training when multiple rollouts are sampled per prompt (Theorem 4). Throughout this section, we abbreviate the subscript Base and Post of the main text as B and P, respectively.

D.1 What the Sharpening Tax Measures

The raw scalability $A(K)$ of Eq. (1) has a simple probabilistic meaning. It counts, in expectation, the failed attempts that are followed by a success within the budget.

Proposition 1 (Scalability as expected failures before the first success)**.** *Suppose we randomly draw a task from the dataset and let H be the index of the first successful rollout ($H = \infty$ if no rollout succeeds), so that $\mathbb{P}(H \le k) = \text{pass@}k$. Then, given the test-time rollout budget K,*

$$A(K) = \mathbb{E}[(H-1)\,\mathbf{1}\{H \le K\}], \qquad S(K) = \mathbb{E}\left[\tfrac{H-1}{K-1}\,\mathbf{1}\{H \le K\} \;\middle|\; H > 1\right] \in [0,1], \quad (5)$$

where the second identity holds whenever pass@$1 < 1$.

In other words, a task solved on the first try adds nothing to $A(K)$, a task first solved on the h-th try adds the $h-1$ failures before it, and a task never solved within the budget adds nothing. Thus, $A(K)$ is the expected number of failed attempts that precede a success within the budget, and $S(K)$ is the same quantity, normalized by $K-1$ and restricted to the tasks that fail on the first try. Both measure how much of a policy's success is obtained *through retries*, and a late recovery counts more than an early one. A positive tax therefore means that the post-trained policy's successes rely less on retries, either because its failed first attempts are recovered less often within the budget (lost coverage) or because they are recovered after fewer retries (faster saturation). The two effects differ, i.e., a policy that improves pass@k at every k can still pay a positive tax if it recovers its failures sooner. Corollary 2 below separates them. This interpretation is also used in Theorem 3 below, since a task pushed to always solved ($H = 1$) or never solved ($H = \infty$) adds zero to $A(K)$.

Proof of Proposition 1. By definition of H, the event $\{H \le k\}$ is that at least one of the first k rollouts succeeds, so we know $\mathbb{P}(H \le k) = \text{pass@}k$ for every $k \ge 1$. Hence, for each $1 \le k \le K-1$,

$$\text{pass@}K - \text{pass@}k = \mathbb{P}(k < H \le K) = \sum_{h=k+1}^{K} \mathbb{P}(H = h).$$

By summing over k and exchanging the two finite sums, we have

$$A(K) = \sum_{k=1}^{K-1} \sum_{h=k+1}^{K} \mathbb{P}(H = h) = \sum_{h=2}^{K} \mathbb{P}(H = h) \sum_{k=1}^{h-1} 1$$
$$= \sum_{h=2}^{K} (h-1)\, \mathbb{P}(H = h) = \mathbb{E}[(H-1)\, \mathbf{1}\{H \le K\}]\,,$$

where the $h = 1$ term is cancelled out due to zero contribution, proves the first identity of Eq. (5).

For the second identity, note that $1 - \text{pass@1} = \mathbb{P}(H > 1) > 0$ by assumption, and that the random variable $(H-1)\,\mathbf{1}\{H \le K\}$ zero out on the event $\{H = 1\}$. Therefore,

$$S(K) = \frac{\mathbb{E}[(H-1)\,\mathbf{1}\{H \le K\}]}{(K-1)\,\mathbb{P}(H > 1)} = \frac{\mathbb{E}[(H-1)\,\mathbf{1}\{H \le K\}\,\mathbf{1}\{H > 1\}]}{(K-1)\,\mathbb{P}(H > 1)}$$
$$= \mathbb{E}\Big[\tfrac{H-1}{K-1}\,\mathbf{1}\{H \le K\} \;\Big|\; H > 1\Big]\,.$$

Finally, $S(K) \in [0, 1]$ because $\frac{H-1}{K-1}\,\mathbf{1}\{H \le K\}$ takes values in $[0, 1]$. It equals $(H-1)/(K-1) \in (0, 1]$ on $\{1 < H \le K\}$ and 0 otherwise. □

D.2 Ceiling and Saturation Decomposition of the Tax

The raw scalability $A(K)$ and the tax $\text{Tax}_A(K)$ can also be decomposed by using the coverage ceiling and the coverage accumulated below it.

Corollary 2 (Ceiling and saturation decomposition)**.** *Denote $C(K) = \text{pass@}K$ as the coverage ceiling and $\overline{C}(K) = \frac{1}{K}\sum_{k=1}^{K} \text{pass@}k$ for the budget-averaged coverage of a policy. Then, for each policy and every $K \ge 2$ test-time rollout budget,*

$$A(K) = K\,\big[C(K) - \overline{C}(K)\big], \qquad \textit{Tax}_A(K) = K\,\big[C_{\mathrm{B}}(K) - C_{\mathrm{P}}(K)\big] \;-\; K\,\big[\overline{C}_{\mathrm{B}}(K) - \overline{C}_{\mathrm{P}}(K)\big]. \tag{6}$$

In other words, the raw scalability of a policy is K times the gap between its ceiling and its budget-averaged coverage, which we call the *saturation gap*. The tax is then K times the difference in the saturation gaps of the two policies, which splits into a ceiling gap and a gap in budget-averaged coverage as in Eq. (6).

Proof of Corollary 2. For the first identity,

$$A(K) = \sum_{k=1}^{K-1} \big[C(K) - \text{pass@}k\big] = (K-1)\,C(K) - \Big(\sum_{k=1}^{K} \text{pass@}k - C(K)\Big)$$
$$= K\,C(K) - \sum_{k=1}^{K} \text{pass@}k = K\,\big[C(K) - \overline{C}(K)\big].$$

The second follows by applying the first identity to each policy in $\text{Tax}_A(K) = A_{\mathrm{B}}(K) - A_{\mathrm{P}}(K)$. □

D.3 Why Sharpening Charges a Tax

We empirically observed that post-training pushes per-task success rates toward the two extremes, always solved and never solved (§3), and a task at either extreme gains nothing from additional rollouts (Proposition 1). We now model the limiting case of this effect, in which sharpened tasks collapse fully to the extremes, and ask how much test-time scalability it removes.

Theorem 3 (Collapse to the extremes charges a proportional tax)**.** *For each task x, let $p_x \in [0, 1]$ be the base model's probability of success on one rollout, and assume that repeated rollouts are conditionally independent given x. Suppose that post-training sharpens each task with probability $\lambda \in [0, 1]$, independently of p_x, in which case its success probability is replaced by either 0 or 1; otherwise, p_x remains unchanged. Then, for every budget $K \ge 2$, the raw tax equals the sharpened proportion of the base model's scalability,*

$$\textit{Tax}_A(K) = \lambda\, A_{\mathrm{Base}}(K) \;\ge\; 0, \qquad A_{\mathrm{P}}(K) = (1-\lambda) A_{\mathrm{Base}}(K). \tag{7}$$

Theorem 3 has two implications. First, sharpening reduces scalability regardless of which extreme a task moves to. A task moved to $p = 1$ succeeds on its first attempt, and a task moved to $p = 0$ fails on every attempt. In both cases, retries can no longer turn the policy's failures into successes, so the task contributes nothing in computing $A(K)$. That is, the raw tax depends only on the sharpened fraction λ, and it is strictly positive whenever $\lambda > 0$ and $A_{\text{Base}}(K) > 0$. Second, a higher first-attempt accuracy does not remove the calibrated tax. Suppose that $A_{\text{Base}}(K) > 0$. If a nonzero proportion of tasks is sharpened to $p_x = 0$, then $\text{Tax}_S(K) > 0$, while pass@1 can still improve as long as a sufficient number of the other sharpened tasks move to $p_x = 1$ (see the proof below). Thus, a policy can look better at K=1 and still pay the tax. Overall, Theorem 3 describes how the bimodalization in §3 produces the tax, and the empirical decomposition in Table 8 and Figure 25 is consistent with this mechanism.

Proof of Theorem 3. Each task x has a per-rollout success probability p_x (population-level exact probability), and rollouts are conditionally independent given the task, so pass@k for task x equals $1 - (1 - p_x)^k$ and dataset-level metrics are expectations over x. Define the per-task raw scalability

$$a_K(p) = \sum_{k=1}^{K-1} \Big[\big(1 - (1-p)^K\big) - \big(1 - (1-p)^k\big) \Big] = \sum_{k=1}^{K-1} \Big[(1-p)^k - (1-p)^K \Big],$$
$$A(K) = \mathbb{E}_x\big[a_K(p_x)\big].$$

Each term $(1-p)^k - (1-p)^K$ vanishes at $p = 0$ and $p = 1$, so $a_K(0) = a_K(1) = 0$. A task that always fails or always succeeds gains nothing from retries. Under the sharpening model, an independently chosen λ-fraction of tasks has p_x replaced by a value in $\{0, 1\}$ while the remaining $(1 - \lambda)$-fraction is unchanged. Taking expectations,

$$A_{\text{P}}(K) = (1-\lambda)\,\mathbb{E}_x\big[a_K(p_x)\big] + \lambda \cdot 0 = (1-\lambda) A_{\text{B}}(K),$$
$$\text{Tax}_A(K) = A_{\text{B}}(K) - A_{\text{P}}(K) = \lambda A_{\text{B}}(K),$$

which is Eq. (7); Since pass@k is nondecreasing in k, we have $A_{\text{B}}(K) \geq 0$; hence $\text{Tax}_A(K) = \lambda A_{\text{B}}(K) \geq 0$, with strict inequality whenever $\lambda > 0$ and $A_{\text{B}}(K) > 0$, regardless of which extreme the sharpened tasks reach.

Meanwhile, for the calibrated tax, write $\epsilon_{\text{B}} = 1 - \text{pass@}1_{\text{B}}$ and $\epsilon_{\text{P}} = 1 - \text{pass@}1_{\text{P}}$ for the single-attempt failure rates, and let $\lambda_0 \leq \lambda$ be the proportion of tasks sharpened to $p = 0$. Unsharpened tasks keep their failure probability and tasks sharpened to $p = 0$ always fail, so

$$\epsilon_{\text{P}} = (1-\lambda)\,\epsilon_{\text{B}} + \lambda_0.$$

Assume $A_{\text{B}}(K) > 0$, which implies $\epsilon_{\text{B}} > 0$, and $\epsilon_{\text{P}} > 0$. Since $S(K) = A(K)/\big((K-1)\,\epsilon\big)$ and $A_{\text{P}}(K) = (1-\lambda)A_{\text{B}}(K)$,

$$\frac{S_{\text{P}}(K)}{S_{\text{B}}(K)} = \frac{(1-\lambda)\,\epsilon_{\text{B}}}{(1-\lambda)\,\epsilon_{\text{B}} + \lambda_0},$$

which is below 1 when $\lambda_0 > 0$. This implies that post-training keeps only a $(1 - \lambda)$ proportion of the base model's scalability but more than a $(1 - \lambda)$ proportion of its failures, because tasks sharpened to $p = 0$ still fail. Meanwhile, first-attempt accuracy improves exactly when $\epsilon_{\text{P}} < \epsilon_{\text{B}}$, i.e., $\lambda_0 < \lambda\,\epsilon_{\text{B}}$. Thus, when $0 < \lambda_0 < \lambda\,\epsilon_{\text{B}}$, first-attempt accuracy improves while the calibrated tax remains strictly positive. □

D.4 How PTGS Improves Group-Based RL

If the tax arises because tasks drift to the two extremes, a natural remedy is to keep tasks at intermediate success rates, where rollouts still carry learning signal. PTGS does this by heating the prompts that the policy keeps failing and cooling the prompts that it has mastered. We now show that this rule improves the rollout groups from which RL learns. Sampling a group of $n \geq 2$ rollouts per prompt is standard in modern RL pipelines. Group-based methods such as GRPO and its variants (Shao et al., 2024; Yu et al., 2025; Liu et al., 2025b; Chu et al., 2026) need multiple rollouts by design, and even PPO, which in principle needs only one rollout per prompt, often uses multiple rollouts in practice for cheaper and lower-variance advantage estimates (Kazemnejad et al., 2025; Wang et al., 2025; 2026b; Hu et al., 2026). Here, whether a group gives a useful learning signal

depends on two events: (1) A value-based learner such as PPO needs at least one success in the group; (2) A group-contrast learner such as GRPO gets a nonzero advantage only from a mixed group, i.e., a group with both a success and a failure. Theorem 4 shows that heating a hard prompt with PTGS makes both events more likely.

Theorem 4 (PTGS improves the groups that RL learns from)**.** *Consider a prompt whose true success probability changes from p to p' under the temperature selected by PTGS, and let each group consist of $n \geq 2$ conditionally independent rollouts. Denote $I_n(q) = 1-(1-q)^n$ as the probability that the group contains at least one success and $J_n(q) = 1-q^n-(1-q)^n$ as the probability that it is a mixed group of successes and failures. If $p' > p$, then*

$$\begin{aligned} &\text{(i)} && I_n(p') > I_n(p) && \text{for all } 0 \leq p < p' \leq 1, \\ &\text{(ii)} && J_n(p') > J_n(p) && \text{if } p < p' \leq \tfrac{1}{2}, \\ &\text{(iii)} && J_n(p') < J_n(p), \text{ with } J_n(p) - J_n(p') \leq (p')^n - p^n && \text{if } \tfrac{1}{2} \leq p < p'. \end{aligned} \tag{8}$$

Suppose that heating raises the success probability of a prompt from p to p'. Part (i) states that groups with at least one success become more frequent at every difficulty level, so a value-based learner receives a reward signal more often. Part (ii) states that mixed groups also become more frequent as long as the prompt remains hard ($p' \leq \frac{1}{2}$), so a group-contrast learner receives nonzero advantages more often as well. Part (iii) covers prompts that the policy has mostly mastered ($p \geq \frac{1}{2}$). In this case, mixed groups become rarer, but so do all-failure groups, and the bound in Eq. (8) ensures that the lost mixed groups turn into all-success groups rather than all-failure groups. These group events are closely tied to our evaluation metrics, since averaging over prompts gives pass@$n = \mathbb{E}_x[I_n(p_x)]$ and pass@n − pass^n $= \mathbb{E}_x[J_n(p_x)]$. In these terms, Theorem 4 says that PTGS gains coverage while a prompt is still being explored and converts that coverage into consistency once the prompt is mastered, which is consistent with the improvement we observed in §5.

Proof of Theorem 4. Within a group, the n rollouts are conditionally independent given the prompt, each succeeding with the prompt's true probability. Hence, for a prompt with success probability q, the probability that at least one rollout succeeds is $1-(1-q)^n = I_n(q)$, the probability that all succeed is q^n, and the probability of a mixed group, at least one success and at least one failure, is $1-q^n-(1-q)^n = J_n(q)$. Averaging the first quantity over prompts gives pass@$n = \mathbb{E}_x[I_n(p_x)]$, and subtracting the all-success probability gives pass@n − pass^n $= \mathbb{E}_x[J_n(p_x)]$, the identities stated after Theorem 4.

(i) $I_n'(q) = n(1-q)^{n-1} > 0$ on $(0,1)$, so I_n is strictly increasing and $p' > p$ implies $I_n(p') > I_n(p)$.

(ii) $J_n'(q) = n\big[(1-q)^{n-1} - q^{n-1}\big]$, which is positive if and only if $q < \frac{1}{2}$ for $n \geq 2$. Thus J_n is strictly increasing on $[0, \frac{1}{2}]$ and maximized at $q = \frac{1}{2}$, so $p < p' \leq \frac{1}{2}$ implies $J_n(p') > J_n(p)$.

(iii) By the same derivative computation, J_n is strictly decreasing on $[\frac{1}{2}, 1]$, so $\frac{1}{2} \leq p < p'$ implies $J_n(p') < J_n(p)$. Moreover, from $J_n(q) = 1-q^n-(1-q)^n$,

$$J_n(p) - J_n(p') = \big[(p')^n - p^n\big] - \big[(1-p)^n - (1-p')^n\big] \leq (p')^n - p^n,$$

since $q \mapsto (1-q)^n$ is strictly decreasing on $[0,1]$. That is, the decrease in the probability of a mixed group is fully absorbed by the increase in the all-success probability q^n, which is itself strictly increasing in q. □

E Additional Results on the Accuracy–Coverage Tension

This section gives the full results behind §3 for each backbone scale, model family, and checkpoint pair. The following three main findings of §3 hold in every family. First, post-training bimodalizes the per-task success rates (Figures 13–16). Second, the coverage and consistency curves of the base model diverge much more than those of the post-trained model (Figures 17–20). Third, the post-trained model stands out in both sampling efficiency and coverage at a small budget, but at a large budget the base model frequently reaches higher coverage, mostly for the larger backbones (Figure 21). Below, we first compare the largest and smallest backbones (§E.1), then show the results for each family and checkpoint pair (§E.2), and look at the individual task curves on WebShop (§E.3). Finally, we show that a global temperature cannot resolve the trade-off (§E.4).

E.1 Aggregate Results by Backbone Scale

We group the model-benchmark combinations by backbone scale. The *largest-backbone* set contains the largest backbone of each family (`gemma-4-31B`, `Ministral-3-14B`, `Qwen2.5-32B`, and `Qwen3.5-35B-A3B`) on the three benchmarks, and the *smallest-backbone* set contains the smallest backbone of each family (`gemma-4-E4B`, `Ministral-3-3B`, `Qwen2.5-3B`, and `Qwen3.5-4B`), giving 12 combinations per set. The aggregate curves are a simple average over the 12 combinations, and the shaded bands show one standard deviation across combinations, i.e., the spread between models and benchmarks (not sampling error).

Figures 10–12 show that the two sets behave in qualitatively different ways. For the smallest backbones (3–4B), post-training mainly supplies basic skills that the base model lacks, e.g., well-formed tool calls, and thereby expands coverage. For the largest backbones, post-training mainly polarizes the per-task success rates and thereby loses coverage. Accordingly, the mean base curve overtakes the mean post-trained curve at around k=22 for the largest backbones but never does so for the smallest ones. At the task level, post-training loses more tasks than it gains for the largest backbones, while the opposite holds for the smallest ones. The same contrast appears in Table 6, where post-training reduces the proportion of always-fail tasks on BFCL for every smallest backbone but increases it for three of the four largest backbones.

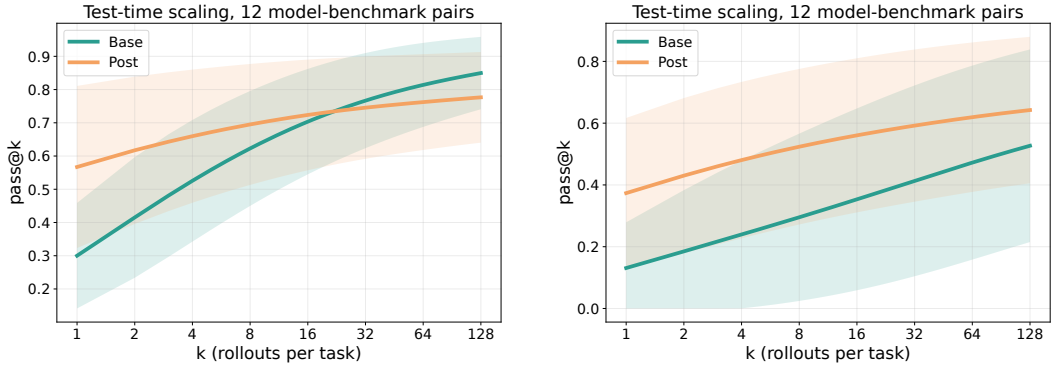


Figure 10: **Aggregate test-time scaling curves of the base and post-trained models for the largest (left) and smallest (right) backbones.** Each curve is the mean pass@k over 12 model-benchmark combinations (one backbone per family on each of the three benchmarks), and the bands show one standard deviation across combinations. For the largest backbones, the mean base curve overtakes the mean post-trained curve at $k \approx 22$ and ends clearly ahead at k=128. For the smallest backbones, the base curve never catches up within the budget.

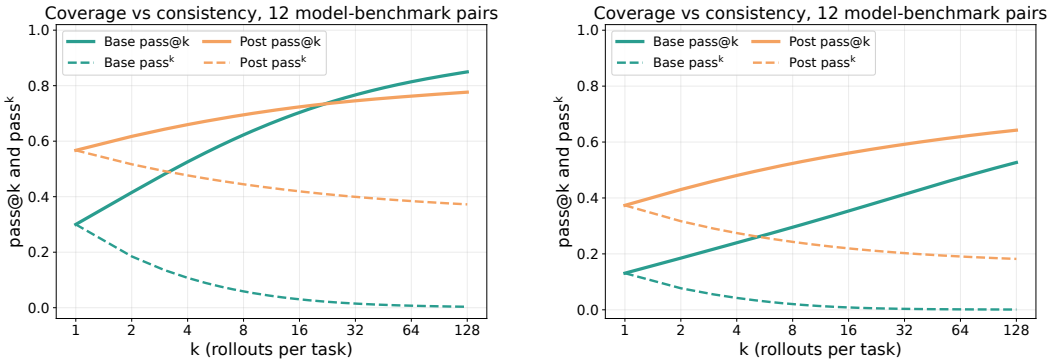


Figure 11: **Aggregate coverage (pass@k) and consistency (pass^k) curves of the base and post-trained models for the largest (left) and smallest (right) backbones**, averaged over 12 model-benchmark combinations (one backbone per family on each of the three benchmarks). The consistency of the base model drops to nearly zero as k grows, but that of the post-trained model stays much higher at both scales. For the largest backbones, more samples let the base model overtake the post-trained model in coverage, but not in consistency.

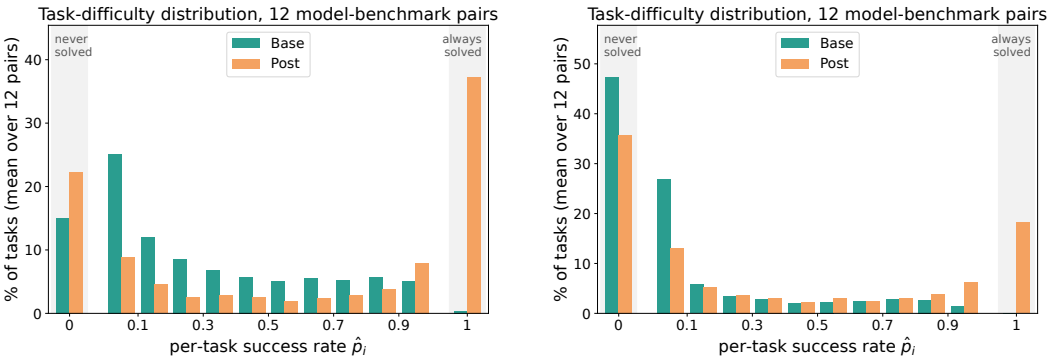


Figure 12: **Aggregate per-task success-rate distributions of the base and post-trained models for the largest (left) and smallest (right) backbones**, averaged over 12 model-benchmark combinations (one backbone per family on each of the three benchmarks). For the largest backbones, post-training increases both the never-solved share and the always-solved share (the latter from nearly 0% to 37%), i.e., it loses some solvable tasks and fully solves others. For the smallest backbones, post-training instead reduces the never-solved share by repairing the base model.

E.2 Results by Model Family and Checkpoint Pair

Table 6 and Figures 13–21 show the results for every family and checkpoint pair, including the task categorization per benchmark at K=128, the per-task success-rate distributions, the coverage and consistency curves, and the comparison between the coverage lost and the sampling efficiency gained by post-training.

Table 6: **Task categories of all 14 base/post-trained pairs at K=128.** Each task is classified by its outcomes over 128 rollouts as *always pass* (every rollout succeeds), *pass given compute* (some but not all rollouts succeed), or *always fail* (no rollout succeeds). Numbers are percentages of tasks in each benchmark. Teal (↑) and purple (↓) mark an increase and a decrease after post-training. For the larger backbones of every family, the middle category shrinks and its tasks move to the two extremes. For the smallest backbones on BFCL, post-training can instead enlarge the middle category by increasing tasks out of always fail, as it improves the tool-call formatting capability of the base model. This is the setting in which the tax can become negative.

		BFCLv4 MT		WebShop		ACEBench	
Backbone	Task category	Base	Post	Base	Post	Base	Post
`gemma-4-E4B`	Always pass (any k)	0.0	12.5 (↑)	0.0	11.4 (↑)	0.0	66.4 (↑)
	Pass given compute	27.0	26.0 (↓)	48.6	53.8 (↑)	85.6	22.9 (↓)
	Always fail (any k)	73.0	61.5 (↓)	51.4	34.8 (↓)	14.4	10.8 (↓)
`gemma-4-12B`	Always pass (any k)	0.0	60.5 (↑)	0.0	14.0 (↑)	0.5	83.2 (↑)
	Pass given compute	47.0	13.5 (↓)	78.0	40.6 (↓)	90.0	7.0 (↓)
	Always fail (any k)	53.0	26.0 (↓)	22.0	45.4 (↑)	9.5	9.7 (↑)
`gemma-4-26B-A4B`	Always pass (any k)	0.0	60.5 (↑)	0.0	17.2 (↑)	0.0	82.5 (↑)
	Pass given compute	62.5	20.5 (↓)	76.4	29.6 (↓)	93.8	7.8 (↓)
	Always fail (any k)	37.5	19.0 (↓)	23.6	53.2 (↑)	6.2	9.7 (↑)
`gemma-4-31B`	Always pass (any k)	0.0	74.0 (↑)	0.0	26.0 (↑)	0.4	84.9 (↑)
	Pass given compute	90.5	9.5 (↓)	87.6	30.0 (↓)	93.8	6.5 (↓)
	Always fail (any k)	9.5	16.5 (↑)	12.4	44.0 (↑)	5.8	8.6 (↑)
`Ministral-3-3B`	Always pass (any k)	0.0	4.5 (↑)	0.0	0.2 (↑)	0.0	16.4 (↑)
	Pass given compute	24.0	57.0 (↑)	32.6	68.6 (↑)	85.7	69.5 (↓)
	Always fail (any k)	76.0	38.5 (↓)	67.4	31.2 (↓)	14.3	14.2 (↓)
`Ministral-3-8B`	Always pass (any k)	0.0	9.0 (↑)	0.0	0.6 (↑)	0.0	40.6 (↑)
	Pass given compute	54.0	55.5 (↑)	78.8	54.0 (↓)	93.2	51.2 (↓)
	Always fail (any k)	46.0	35.5 (↓)	21.2	45.4 (↑)	6.8	8.2 (↑)
`Ministral-3-14B`	Always pass (any k)	0.0	11.5 (↑)	0.0	0.8 (↑)	0.0	50.4 (↑)
	Pass given compute	59.0	53.5 (↓)	78.8	72.2 (↓)	93.9	42.9 (↓)
	Always fail (any k)	41.0	35.0 (↓)	21.2	27.0 (↑)	6.1	6.8 (↑)
`Qwen2.5-3B`	Always pass (any k)	0.0	5.5 (↑)	0.0	1.2 (↑)	0.0	26.8 (↑)
	Pass given compute	12.0	31.0 (↑)	7.6	12.0 (↑)	80.0	42.6 (↓)
	Always fail (any k)	88.0	63.5 (↓)	92.4	86.8 (↓)	20.0	30.6 (↑)
`Qwen2.5-7B`	Always pass (any k)	0.5	8.0 (↑)	0.0	10.6 (↑)	0.0	51.0 (↑)
	Pass given compute	21.5	52.0 (↑)	71.2	46.2 (↓)	89.2	24.0 (↓)
	Always fail (any k)	78.0	40.0 (↓)	28.8	43.2 (↑)	10.8	24.9 (↑)
`Qwen2.5-14B`	Always pass (any k)	0.5	7.5 (↑)	0.0	1.4 (↑)	1.6	59.2 (↑)
	Pass given compute	58.5	67.5 (↑)	74.0	53.8 (↓)	89.2	21.3 (↓)
	Always fail (any k)	41.0	25.0 (↓)	26.0	44.8 (↑)	9.2	19.5 (↑)
`Qwen2.5-32B`	Always pass (any k)	1.0	12.5 (↑)	0.0	11.8 (↑)	1.7	74.0 (↑)
	Pass given compute	70.0	51.5 (↓)	83.8	50.0 (↓)	92.7	13.9 (↓)
	Always fail (any k)	29.0	36.0 (↑)	16.2	38.2 (↑)	5.6	12.1 (↑)
`Qwen3.5-4B`	Always pass (any k)	1.0	19.5 (↑)	0.0	5.6 (↑)	0.1	48.6 (↑)
	Pass given compute	62.5	65.0 (↑)	71.6	65.0 (↓)	93.9	39.1 (↓)
	Always fail (any k)	36.5	15.5 (↓)	28.4	29.4 (↑)	6.0	12.3 (↑)
`Qwen3.5-9B`	Always pass (any k)	0.0	19.5 (↑)	0.0	1.0 (↑)	0.0	45.6 (↑)
	Pass given compute	84.0	65.0 (↓)	73.4	70.6 (↓)	95.1	46.6 (↓)
	Always fail (any k)	16.0	15.5 (↓)	26.6	28.4 (↑)	4.9	7.8 (↑)
`Qwen3.5-35B-A3B`	Always pass (any k)	1.0	33.5 (↑)	0.0	3.2 (↑)	0.0	64.2 (↑)
	Pass given compute	86.0	53.0 (↓)	84.4	71.4 (↓)	95.1	30.8 (↓)
	Always fail (any k)	13.0	13.5 (↑)	15.6	25.4 (↑)	4.9	5.1 (↑)

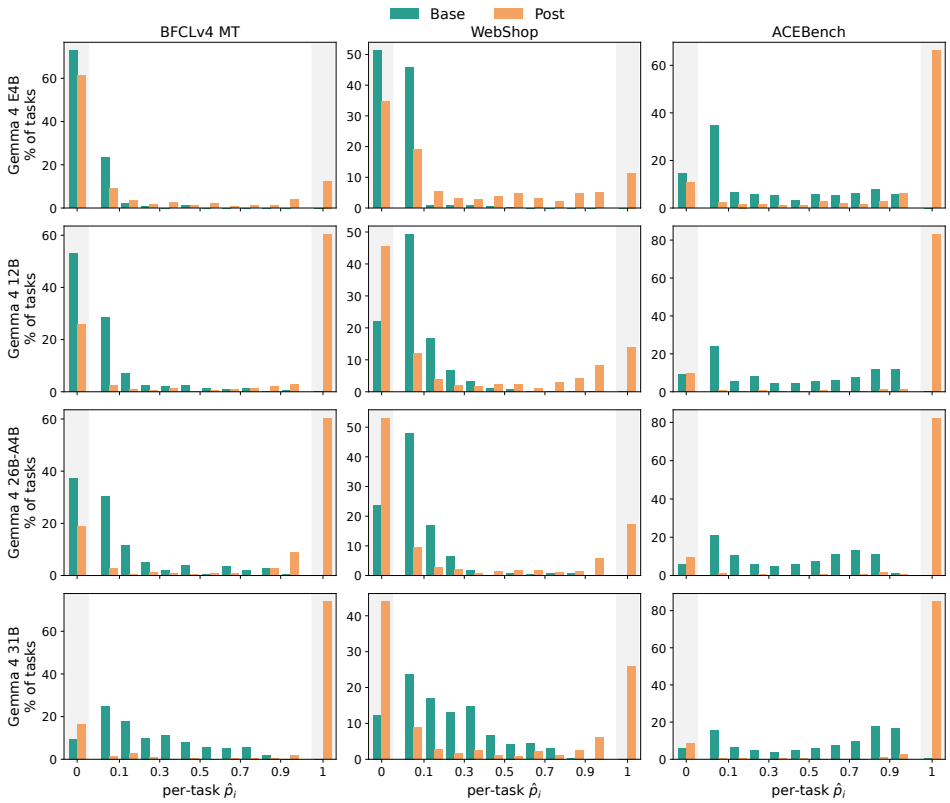


Figure 13: **Per-task success-rate distributions of the Gemma-4 base and post-trained models** (empirical success rate over 128 rollouts per task) for every backbone and benchmark. Post-training moves most of the intermediate mass to the two extremes at every model size and on every benchmark. Among the four families, this polarization is the strongest for Gemma-4, which also has the largest mean tax.

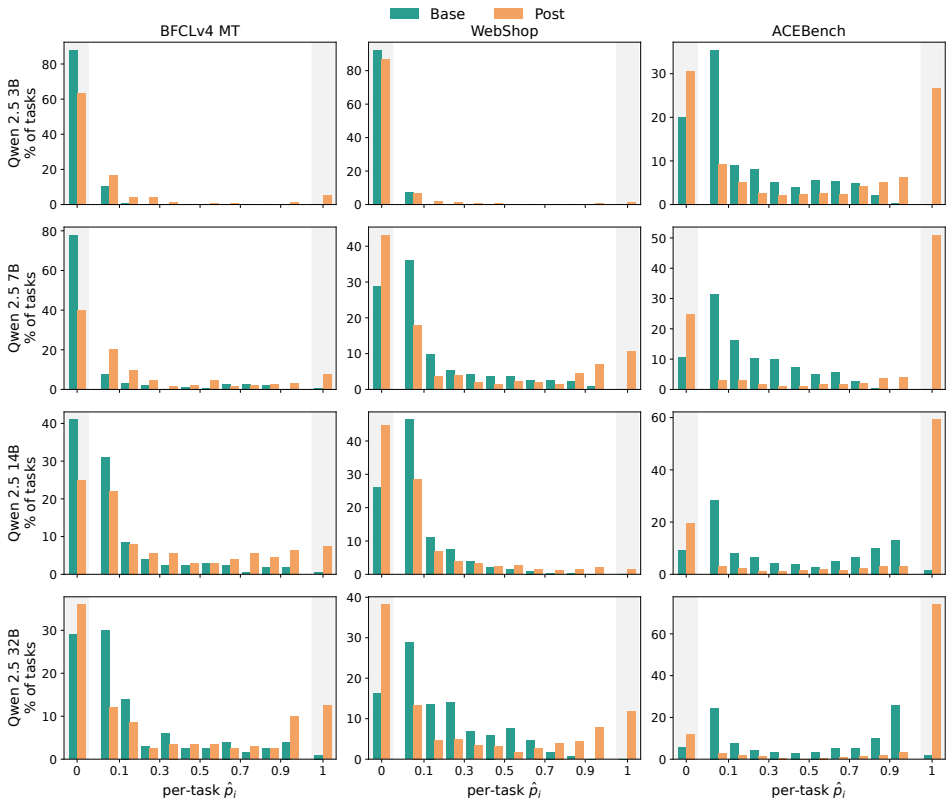


Figure 14: **Per-task success-rate distributions of the Qwen2.5 base and post-trained models** (empirical success rate over 128 rollouts per task) for every backbone and benchmark. Post-training moves the intermediate mass to the two extremes, as in the other families.

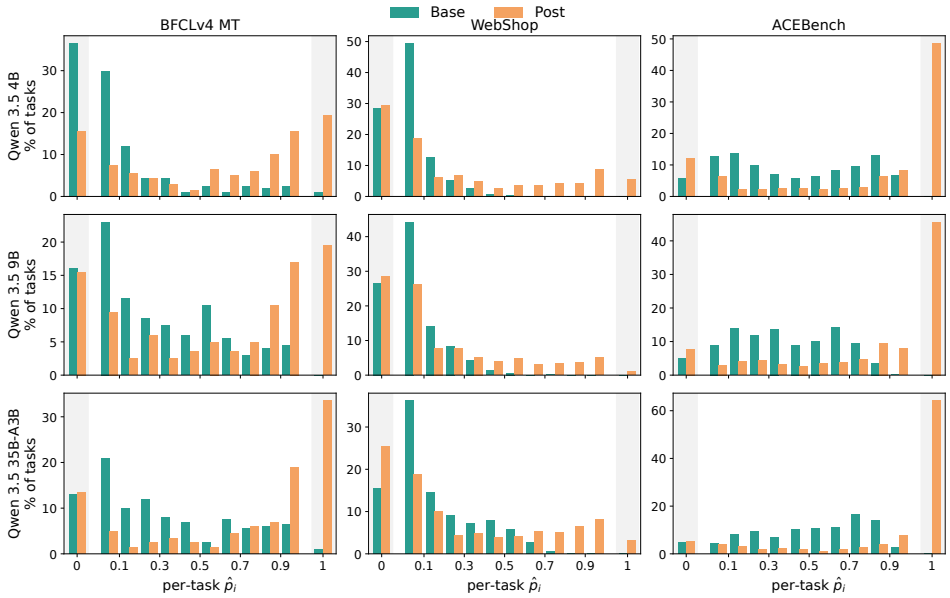


Figure 15: **Per-task success-rate distributions of the Qwen3.5 base and post-trained models** (empirical success rate over 128 rollouts per task) for every backbone and benchmark. Qwen3.5 shows the softest sharpening among the four families.

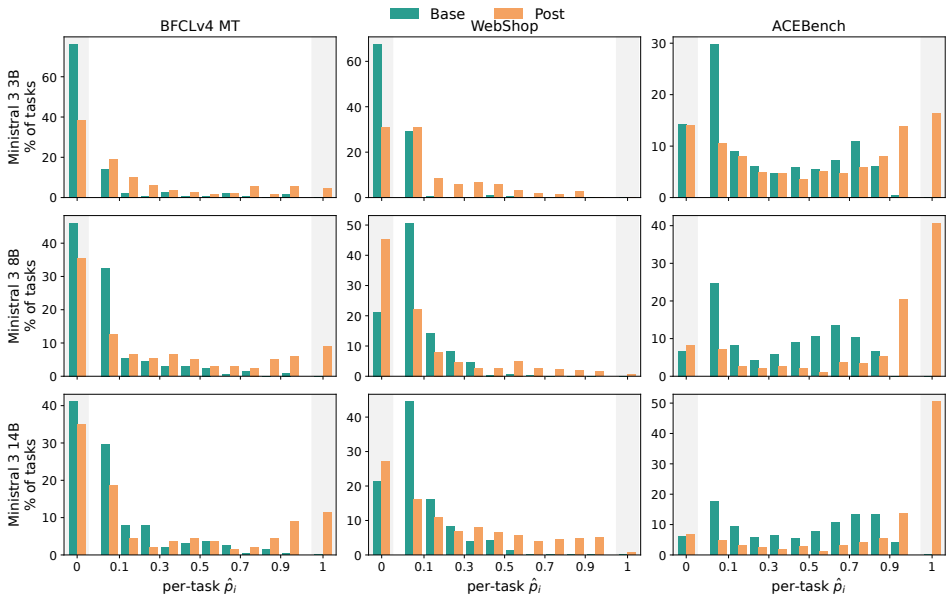


Figure 16: **Per-task success-rate distributions of the Ministral-3 base and post-trained models** (empirical success rate over 128 rollouts per task) for every backbone and benchmark. The polarization grows with model scale. For the 3B backbone, the base model rarely succeeds on BFCL and WebShop and post-training largely improve it, while the 8B and 14B backbones show the shift of mass to the two extremes as we observed in other backbones.

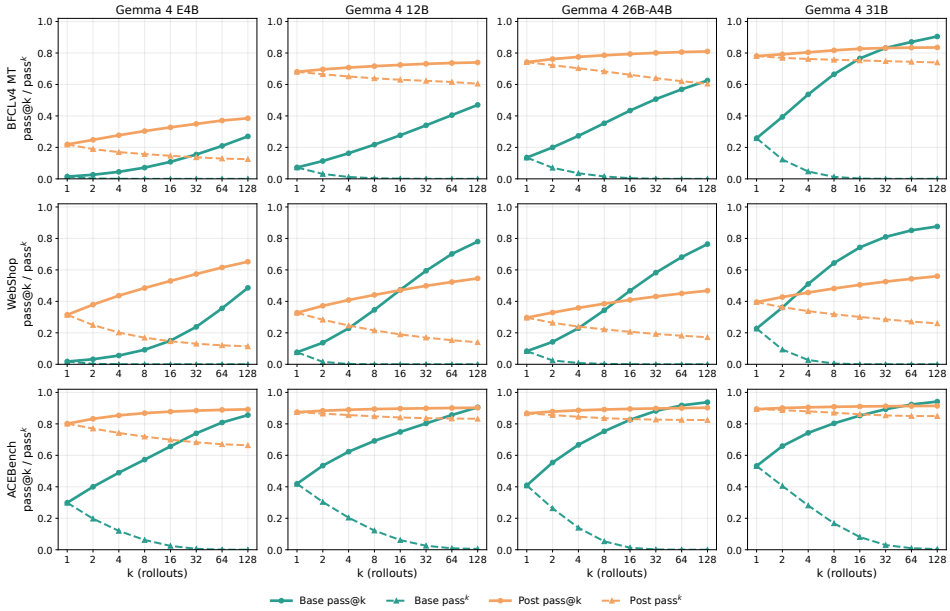


Figure 17: **Coverage (pass@k) and consistency (pass^k) curves of the Gemma-4 base and post-trained models** for every backbone and benchmark. The two curves of the post-trained model stay close together, as expected from a bimodalized policy. In contrast, the consistency of the base model drops toward zero while its coverage keeps rising. The base model overtakes the post-trained model in coverage at a smaller budget for larger backbones.

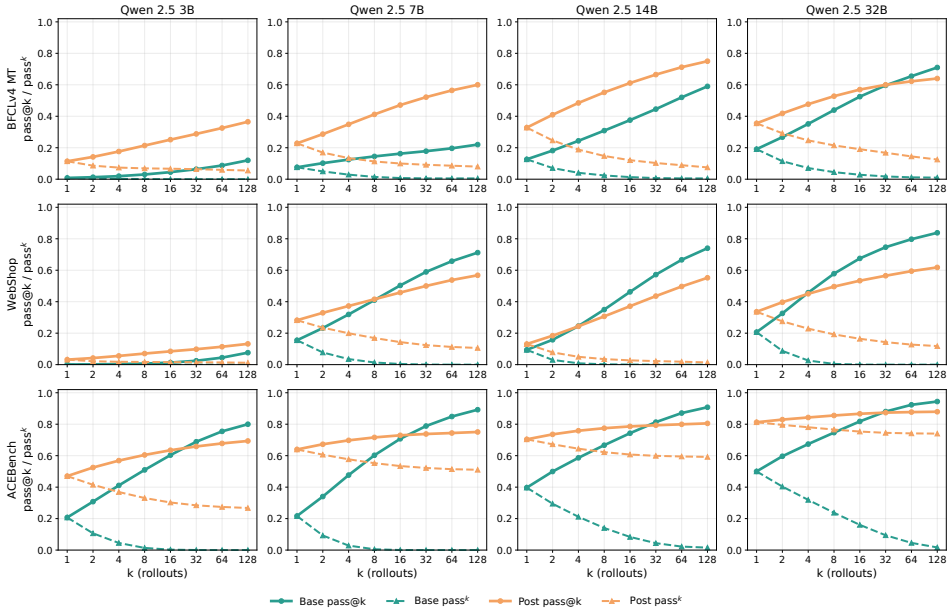


Figure 18: **Coverage (pass@k) and consistency (pass^k) curves of the Qwen2.5 base and post-trained models** for every backbone and benchmark. The consistency of the base model drops toward zero while its coverage keeps rising, and the two curves of the post-trained model stay much closer together.

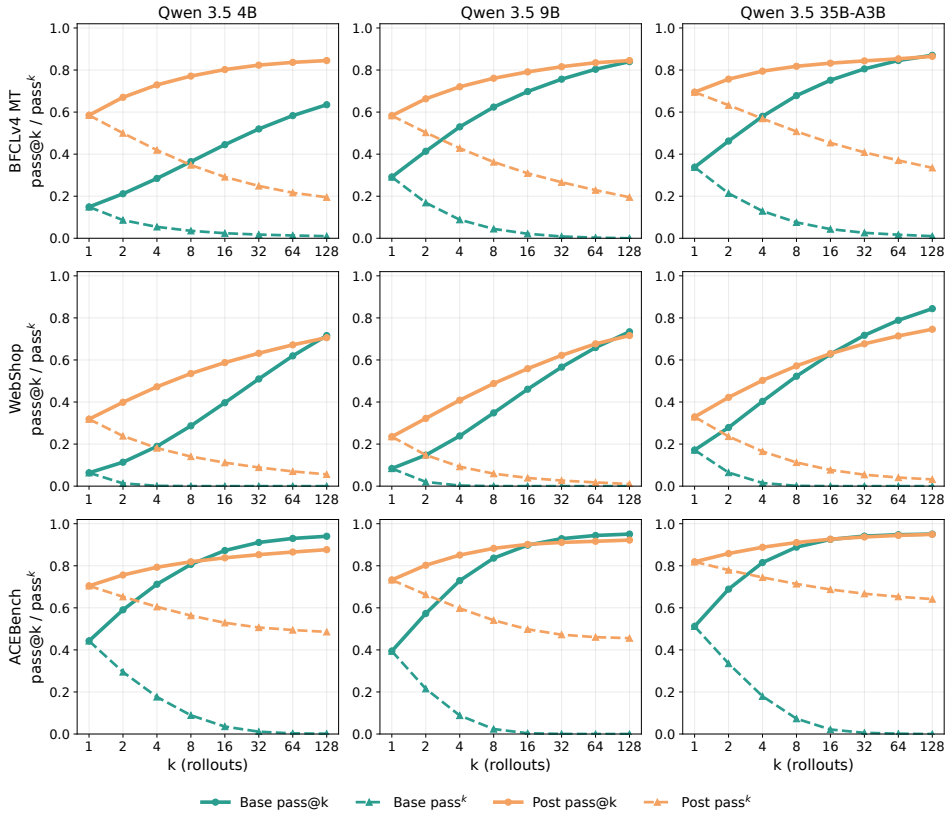


Figure 19: **Coverage (pass@k) and consistency (pass^k) curves of the Qwen3.5 base and post-trained models** for every backbone and benchmark. Since Qwen3.5 has the softest sharpening among the four families, the consistency curves of its post-trained models decay slightly instead of staying flat, and the coverage gaps between the base and post-trained models at k=128 are the smallest among the four families.

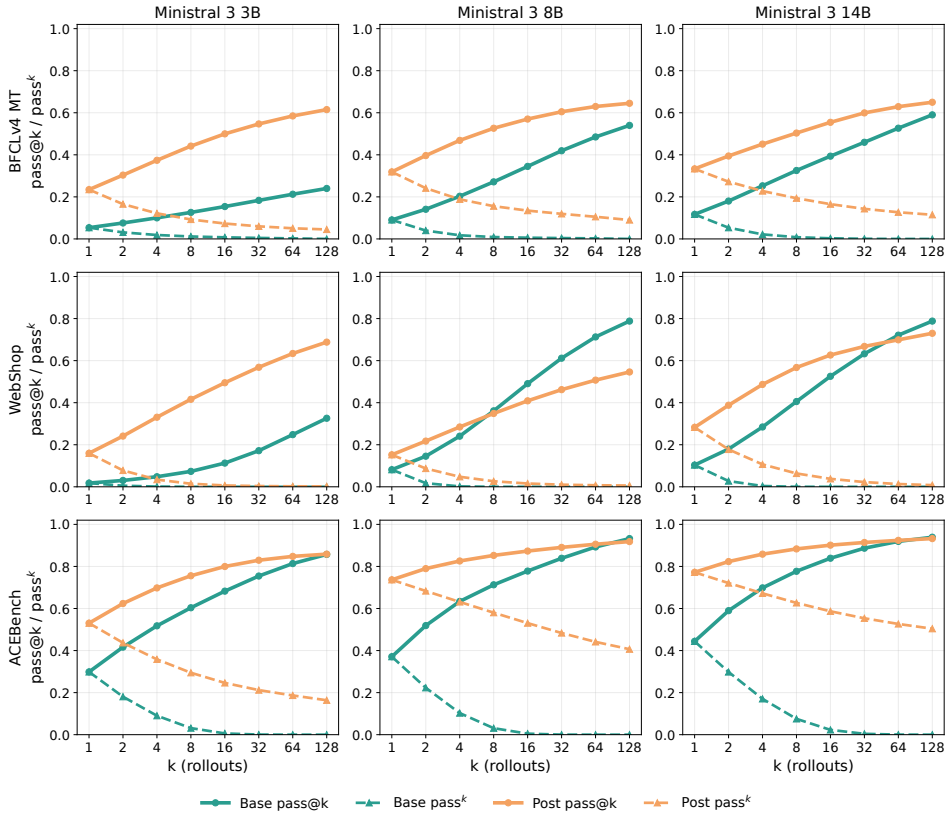


Figure 20: **Coverage (pass@k) and consistency (pass^k) curves of the Ministral-3 base and post-trained models** for every backbone and benchmark. The consistency of the base model drops toward zero while its coverage keeps rising, and the two curves of the post-trained model stay much closer together.

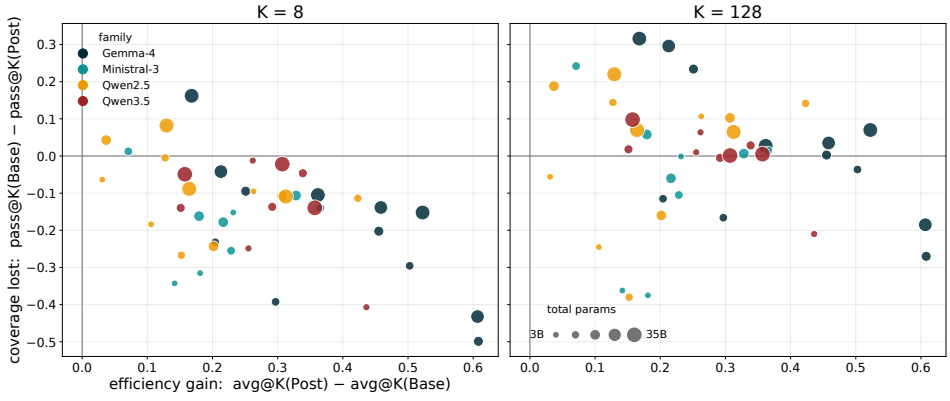


Figure 21: **Coverage lost versus sampling efficiency gained by post-training** at $K{=}8$ (left) and $K{=}128$ (right). The y-axis is the coverage lost, pass@K_{base} − pass@K_{RL}, and the x-axis is the efficiency gained, avg@K_{RL} − avg@K_{base}, where avg@K is the mean success rate over K rollouts. Each point is one model–benchmark combination, colored by family, and the marker size grows with the total number of parameters. At $K{=}8$, most points lie below zero, i.e., post-training is beneficial in both coverage and efficiency when compute is scarce. At $K{=}128$, many points, mostly from the larger backbones, cross above zero, implying that post-training still gains efficiency, but now pays for it with lost coverage.

E.3 Per-Task Scaling Curves and Their Density on WebShop

Figure 22 shows the scaling curves of individual WebShop tasks (500 tasks, 128 rollouts per policy) for the smallest and largest backbones of Gemma-4, the family with the steepest sharpening. The top row of each block shows example tasks of each policy at the harder, intermediate, and easier difficulty ranks, and the bottom row shows the density of all 500 per-task curves together with their mean.

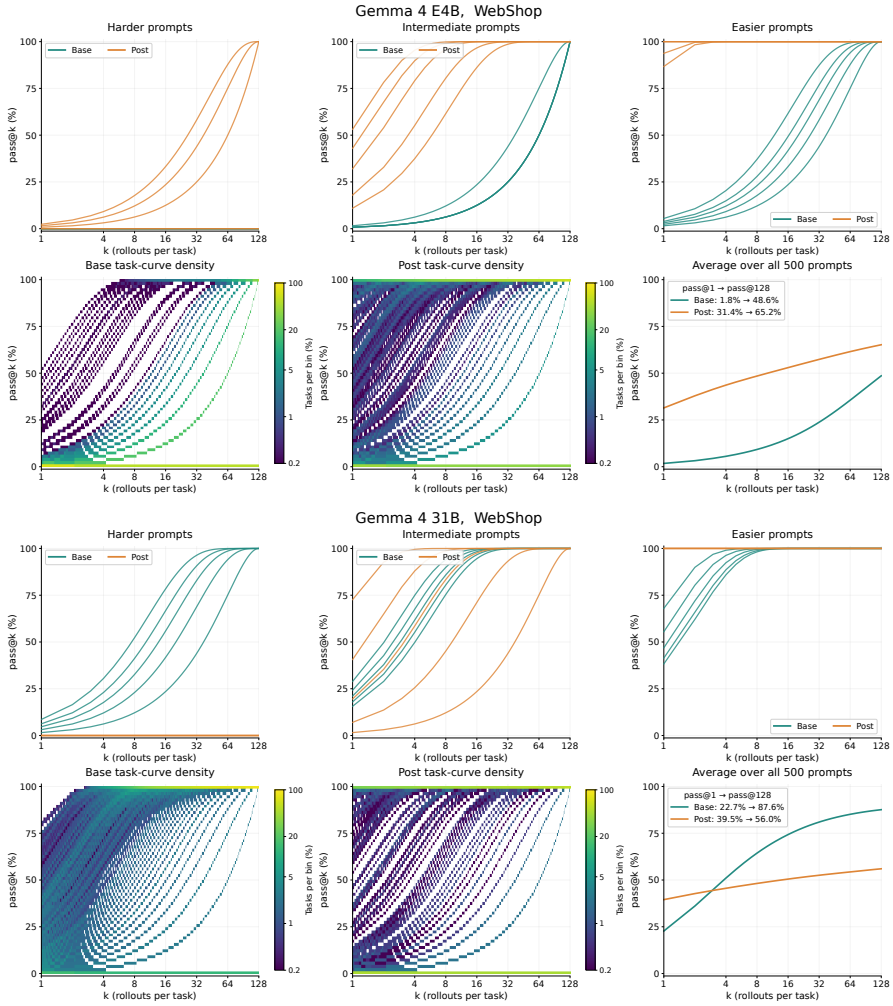


Figure 22: **Per-task scaling curves on WebShop for the smallest and largest Gemma-4 backbones** (`gemma-4-E4B`, top two rows; `gemma-4-31B`, bottom two rows). In each block, the top row shows five base (teal) and five post-trained (orange) task curves from the harder (5–25%), intermediate (40–60%), and easier (75–95%) rank quantiles. Tasks are ranked separately for each policy by their success counts, among the tasks that at least one policy solves at least once. The bottom row shows the density of all 500 per-task curves of each policy and their mean pass@k, with pass@1 and pass@128 given in the legend. All curves use the unbiased estimator from 128 rollouts, and a flat curve at zero means no observed success.

Bimodalization across the budget axis. The density panels in Figure 22 show the mechanism of §3 at every budget. Under the post-trained model, the mass stays at the two edges for all k, i.e., always-solved tasks at the top and never-solved tasks at the bottom, with a thin band in between. Under the base model of `gemma-4-31B`, the mass starts low and moves upward through the intermediate region as k grows, and reaches the top only at large k. These are the tasks that retries turn into successes, and they explain why the mean base curve ends well above the mean post-trained curve. The difficulty-matched examples show the same pattern for individual tasks. That is, as the harder base tasks reach 100% within the budget while the harder post-trained tasks stay at zero. The smallest

backbone, `gemma-4-E4B`, behaves in the opposite way. Much of its base density stays at the bottom edge, and the post-trained model remains ahead at k=128. The other three families we omitted here also show the same trend.

E.4 Global Temperature Scaling Cannot Resolve the Trade-off

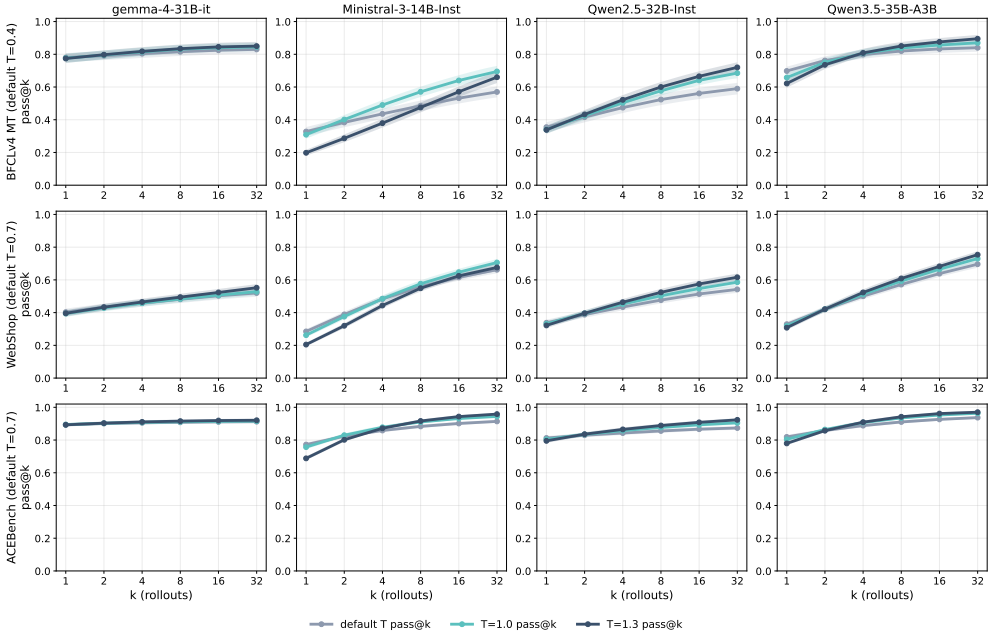


Figure 23: **Global temperature scaling trades single-sample accuracy for coverage rather than improving both.** pass@k of the post-trained model of the largest backbone in each family (columns) on BFCLv4 MT, WebShop, and ACEBench (rows), at the default temperature of each benchmark (0.4 for BFCL and 0.7 for the others) and at $T \in \{1.0, 1.3\}$. Shaded bands show ± 1 bootstrap standard error over tasks (200 resamples). A higher temperature usually improves pass@k at larger budgets, but pass@1 drops in some cases.

A simple way to recover the lost coverage is to flatten the post-trained policy with a higher sampling temperature at inference, i.e., to heat every prompt with higher temperature. Figure 23 increases the temperature of the post-trained model of the largest backbone in each family from the default of each benchmark up to T=1.3. A higher temperature usually improves pass@k at larger budgets, which confirms that part of the lost coverage can be recovered by widening the sampling distribution (Dang et al., 2025). However, pass@1 does not improve in any of the twelve panels and even drops clearly in some cases, e.g., `Ministral-3-14B-Instruct-2512`. The coverage gains are also neither uniform nor monotone in T. `gemma-4-31B-it`, the most sharply post-trained policy, hardly responds to heating, and for `Ministral-3-14B-Instruct-2512`, T=1.0 gives a higher pass@32 than T=1.3 on BFCL and WebShop. In summary, a single global temperature therefore moves the policy along the accuracy–coverage trade-off and cannot improve both. This suggests that **a prompt-adaptive temperature is needed to push the accuracy–coverage frontier upward**, which motivates PTGS in §5.

F Additional Results on the Sharpening Tax

This section gives additional results for §4. We report both tax variants for all 14 checkpoint pairs together with the full numerical results (§F.1), decompose the tax by how tasks move after post-training (§F.2), show the scaling curves of the tasks that contribute most to the tax (§F.3), and use a cheap tax estimate to route each task to the base or the post-trained policy (§F.4).

F.1 Sharpening Tax for All Fourteen Checkpoint Pairs

Figure 24 shows $\mathrm{Tax}_A(k)$ and $\mathrm{Tax}_S(k)$ as a function of the rollout budget for all 14 base/post-trained pairs, extending Figure 6 of the main text. Table 7 summarizes all 42 model-benchmark evaluations. Every evaluation uses $N = 128$ rollouts per task for both models and the same task set for each benchmark (200, 500, and 770 tasks for BFCL, WebShop, and ACEBench). For each combination, we report the coverage $C = \mathrm{pass@128}$ of both models, the coverage gap $\Delta C = C_B - C_P$ with a 95% paired task-bootstrap confidence interval, the budget-averaged coverage $\overline{C}(128)$ of both models, and the two tax metrics. By Corollary 2, the raw scalability of each model is $A(128) = 128\,(C - \overline{C}(128))$.

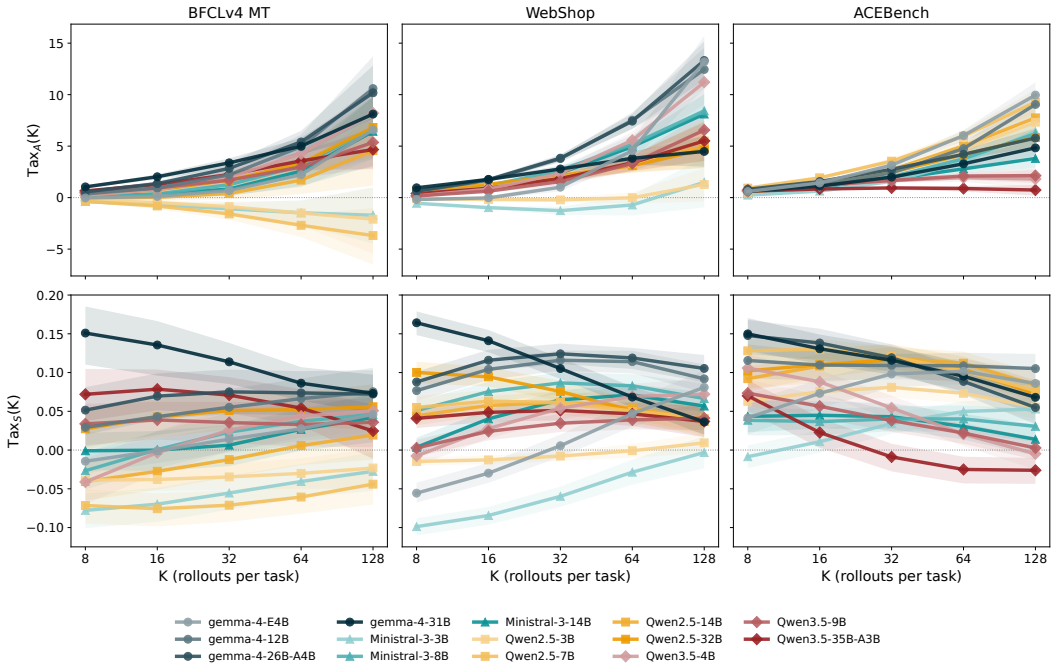


Figure 24: **Sharpening Tax as a function of the rollout budget for all 14 base/RL pairs.** $\mathrm{Tax}_A(k)$ (top) and $\mathrm{Tax}_S(k)$ (bottom) on BFCLv4 MT, WebShop, and ACEBench, with one curve per checkpoint pair and shaded 95% bootstrap confidence intervals. Positive values mean that post-training reduces test-time scalability relative to the base model. The raw tax starts near zero and grows with the budget for almost every pair, while the calibrated tax can either rise or fall with the budget. Both variants stay negative at k=128 only for `Ministral-3-3B`, `Qwen2.5-3B`, and `Qwen2.5-7B` on BFCL, where post-training repairs the tool-call formatting of the base model.

Table 7: **Summary of the 42 model–benchmark evaluations at K=128.** C_B and C_P are the coverage (pass@128) of the base and post-trained (RL) models, $\Delta C = C_B - C_P$ is the coverage gap with a 95% paired task-bootstrap confidence interval, and $\overline{C}_B$ and $\overline{C}_P$ are the budget-averaged coverage $\overline{C}(128) = \frac{1}{128}\sum_{k=1}^{128} \text{pass@}k$ of the two models. The raw scalability of each model is $A(128) = 128\,(C - \overline{C}(128))$, and the last two columns are the raw and calibrated Sharpening Tax at K=128. In every combination, the post-trained model reaches a larger fraction of its ceiling ($\overline{C}/C$) than the base model.

Benchmark	Pair	C_B	C_P	ΔC [95% CI]	$\overline{C}_B$	$\overline{C}_P$	Tax_A	Tax_S
BFCLv4 MT	Gemma 4 E4B	0.270	0.385	-0.115 [-0.190,-0.050]	0.194	0.360	+6.59	+0.046
BFCLv4 MT	Gemma 4 12B	0.470	0.740	-0.270 [-0.345,-0.200]	0.380	0.733	+10.59	+0.075
BFCLv4 MT	Gemma 4 26B-A4B	0.625	0.810	-0.185 [-0.255,-0.120]	0.538	0.803	+10.18	+0.072
BFCLv4 MT	Gemma 4 31B	0.905	0.835	+0.070 [+0.015,+0.120]	0.838	0.831	+8.11	+0.073
BFCLv4 MT	Ministral 3 3B	0.240	0.615	-0.375 [-0.445,-0.300]	0.201	0.562	-1.71	-0.027
BFCLv4 MT	Ministral 3 8B	0.540	0.645	-0.105 [-0.190,-0.025]	0.454	0.611	+6.76	+0.046
BFCLv4 MT	Ministral 3 14B	0.590	0.650	-0.060 [-0.140,+0.030]	0.499	0.609	+6.44	+0.042
BFCLv4 MT	Qwen 2.5 3B	0.120	0.365	-0.245 [-0.310,-0.180]	0.082	0.311	-2.09	-0.023
BFCLv4 MT	Qwen 2.5 7B	0.220	0.600	-0.380 [-0.460,-0.305]	0.190	0.542	-3.69	-0.044
BFCLv4 MT	Qwen 2.5 14B	0.590	0.750	-0.160 [-0.230,-0.085]	0.491	0.687	+4.56	+0.019
BFCLv4 MT	Qwen 2.5 32B	0.710	0.640	+0.070 [-0.005,+0.140]	0.624	0.607	+6.81	+0.056
BFCLv4 MT	Qwen 3.5 4B	0.635	0.845	-0.210 [-0.270,-0.155]	0.551	0.825	+8.21	+0.051
BFCLv4 MT	Qwen 3.5 9B	0.840	0.845	-0.005 [-0.055,+0.045]	0.774	0.821	+5.36	+0.036
BFCLv4 MT	Qwen 3.5 35B-A3B	0.870	0.865	+0.005 [-0.035,+0.045]	0.816	0.848	+4.67	+0.025
WebShop	Gemma 4 E4B	0.486	0.652	-0.166 [-0.212,-0.116]	0.327	0.596	+13.19	+0.081
WebShop	Gemma 4 12B	0.780	0.546	+0.234 [+0.188,+0.280]	0.648	0.511	+12.45	+0.092
WebShop	Gemma 4 26B-A4B	0.764	0.468	+0.296 [+0.246,+0.346]	0.633	0.441	+13.32	+0.105
WebShop	Gemma 4 31B	0.876	0.560	+0.316 [+0.272,+0.360]	0.816	0.534	+4.49	+0.036
WebShop	Ministral 3 3B	0.326	0.688	-0.362 [-0.410,-0.316]	0.228	0.601	+1.49	-0.003
WebShop	Ministral 3 8B	0.788	0.546	+0.242 [+0.194,+0.290]	0.660	0.484	+8.42	+0.066
WebShop	Ministral 3 14B	0.788	0.730	+0.058 [+0.022,+0.096]	0.673	0.678	+8.11	+0.057
WebShop	Qwen 2.5 3B	0.076	0.132	-0.056 [-0.090,-0.022]	0.043	0.109	+1.28	+0.009
WebShop	Qwen 2.5 7B	0.712	0.568	+0.144 [+0.100,+0.192]	0.620	0.519	+5.50	+0.041
WebShop	Qwen 2.5 14B	0.740	0.552	+0.188 [+0.138,+0.236]	0.619	0.470	+4.97	+0.039
WebShop	Qwen 2.5 32B	0.838	0.618	+0.220 [+0.184,+0.260]	0.763	0.579	+4.69	+0.037
WebShop	Qwen 3.5 4B	0.716	0.706	+0.010 [-0.030,+0.050]	0.574	0.651	+11.21	+0.072
WebShop	Qwen 3.5 9B	0.734	0.716	+0.018 [-0.012,+0.048]	0.614	0.647	+6.56	+0.041
WebShop	Qwen 3.5 35B-A3B	0.844	0.746	+0.098 [+0.064,+0.130]	0.747	0.692	+5.51	+0.037
ACEBench	Gemma 4 E4B	0.856	0.892	-0.036 [-0.062,-0.012]	0.771	0.885	+9.96	+0.086
ACEBench	Gemma 4 12B	0.905	0.903	+0.003 [-0.023,+0.030]	0.832	0.900	+9.05	+0.105
ACEBench	Gemma 4 26B-A4B	0.938	0.903	+0.035 [+0.010,+0.060]	0.889	0.899	+5.78	+0.055
ACEBench	Gemma 4 31B	0.942	0.914	+0.027 [+0.005,+0.048]	0.902	0.912	+4.83	+0.068
ACEBench	Ministral 3 3B	0.857	0.858	-0.001 [-0.027,+0.025]	0.780	0.831	+6.42	+0.053
ACEBench	Ministral 3 8B	0.932	0.918	+0.014 [-0.006,+0.035]	0.862	0.897	+6.21	+0.031
ACEBench	Ministral 3 14B	0.939	0.932	+0.006 [-0.010,+0.026]	0.894	0.917	+3.80	+0.014
ACEBench	Qwen 2.5 3B	0.800	0.694	+0.106 [+0.071,+0.140]	0.715	0.667	+7.32	+0.055
ACEBench	Qwen 2.5 7B	0.892	0.751	+0.142 [+0.110,+0.169]	0.808	0.740	+9.26	+0.076
ACEBench	Qwen 2.5 14B	0.908	0.805	+0.103 [+0.074,+0.131]	0.837	0.795	+7.74	+0.084
ACEBench	Qwen 2.5 32B	0.944	0.879	+0.065 [+0.039,+0.088]	0.893	0.873	+5.83	+0.072
ACEBench	Qwen 3.5 4B	0.940	0.877	+0.064 [+0.043,+0.084]	0.907	0.858	+1.81	-0.005
ACEBench	Qwen 3.5 9B	0.951	0.922	+0.029 [+0.012,+0.045]	0.922	0.910	+2.14	+0.003
ACEBench	Qwen 3.5 35B-A3B	0.951	0.949	+0.001 [-0.016,+0.018]	0.935	0.939	+0.74	-0.026

F.2 Decomposition of the Tax by Task Movement

Table 8 splits $\text{Tax}_A(128)$ into three parts, the contribution of tasks solved only by the base model (lost), that of tasks solved only by the post-trained model (gained), and the difference in curve shape on tasks solved by both (shape). This is an empirical, finite-K counterpart of the sharpening model in Theorem 3. The lost term is positive in every combination, whereas a large gained term appears mainly for the small backbones on BFCL and WebShop, where post-training repairs the base model. Figure 25 shows how individual tasks move for the largest backbone of each family.

Table 8: **Decomposition of $\mathrm{Tax}_A(128)$ by task movement for all 42 model–benchmark combinations.** Each task is assigned to a category by which model solves it at least once in 128 rollouts, i.e., *Lost* (only the base model), *Gained* (only the post-trained model), *Both*, or *Neither*. The left block reports the proportion of tasks in each category. The right block splits $\mathrm{Tax}_A(128)$ into the retry value lost on Lost tasks (positive), the retry value recovered on Gained tasks (negative), and the difference in curve shape on tasks solved by both models. The three contributions sum to the total in the last column.

		Task categories (% of tasks)				Contributions to $\mathrm{Tax}_A(128)$			
Benchmark	Pair	Lost	Gained	Both	Neither	Lost (+)	Gained (−)	Shape	$\mathrm{Tax}_A(128)$
BFCLv4 MT	Gemma 4 E4B	8.0	19.5	19.0	53.5	+3.88	-2.25	+4.97	+6.59
BFCLv4 MT	Gemma 4 12B	4.0	31.0	43.0	22.0	+1.73	-0.21	+9.07	+10.59
BFCLv4 MT	Gemma 4 26B-A4B	4.5	23.0	58.0	14.5	+0.82	-0.81	+10.17	+10.18
BFCLv4 MT	Gemma 4 31B	10.5	3.5	80.0	6.0	+2.43	-0.05	+5.73	+8.11
BFCLv4 MT	Ministral 3 3B	5.0	42.5	19.0	33.5	+1.73	-5.41	+1.96	-1.71
BFCLv4 MT	Ministral 3 8B	12.5	23.0	41.5	23.0	+3.44	-1.78	+5.09	+6.76
BFCLv4 MT	Ministral 3 14B	16.0	22.0	43.0	19.0	+4.26	-2.37	+4.54	+6.44
BFCLv4 MT	Qwen 2.5 3B	1.5	26.0	10.5	62.0	+0.74	-6.18	+3.35	-2.09
BFCLv4 MT	Qwen 2.5 7B	3.0	41.0	19.0	37.0	+1.06	-6.49	+1.74	-3.69
BFCLv4 MT	Qwen 2.5 14B	6.5	22.5	52.5	18.5	+1.34	-2.91	+6.13	+4.56
BFCLv4 MT	Qwen 2.5 32B	17.5	10.5	53.5	18.5	+3.08	-0.98	+4.71	+6.81
BFCLv4 MT	Qwen 3.5 4B	1.5	22.5	62.0	14.0	+0.52	-1.38	+9.08	+8.21
BFCLv4 MT	Qwen 3.5 9B	6.0	6.5	78.0	9.5	+2.00	-0.85	+4.21	+5.36
BFCLv4 MT	Qwen 3.5 35B-A3B	4.0	3.5	83.0	9.5	+0.81	-0.04	+3.90	+4.67
WebShop	Gemma 4 E4B	10.0	26.6	38.6	24.8	+4.73	-4.00	+12.46	+13.19
WebShop	Gemma 4 12B	29.2	5.8	48.8	16.2	+8.32	-0.51	+4.64	+12.45
WebShop	Gemma 4 26B-A4B	34.2	4.6	42.2	19.0	+9.20	-0.42	+4.54	+13.32
WebShop	Gemma 4 31B	33.0	1.4	54.6	11.0	+4.38	-0.38	+0.48	+4.49
WebShop	Ministral 3 3B	3.0	39.2	29.6	28.2	+1.52	-8.76	+8.73	+1.49
WebShop	Ministral 3 8B	28.8	4.6	50.0	16.6	+9.04	-0.58	-0.04	+8.42
WebShop	Ministral 3 14B	11.8	6.0	67.0	15.2	+4.73	-1.12	+4.51	+8.11
WebShop	Qwen 2.5 3B	5.4	11.0	2.2	81.4	+3.16	-2.48	+0.59	+1.28
WebShop	Qwen 2.5 7B	21.4	7.0	49.8	21.8	+5.61	-1.78	+1.67	+5.50
WebShop	Qwen 2.5 14B	26.2	7.4	47.8	18.6	+6.80	-1.67	-0.17	+4.97
WebShop	Qwen 2.5 32B	23.0	1.0	60.8	15.2	+5.62	-0.25	-0.69	+4.69
WebShop	Qwen 3.5 4B	11.0	10.0	60.6	18.4	+5.13	-2.31	+8.40	+11.21
WebShop	Qwen 3.5 9B	7.0	5.2	66.4	21.4	+2.91	-1.42	+5.07	+6.56
WebShop	Qwen 3.5 35B-A3B	12.2	2.4	72.2	13.2	+4.43	-0.90	+1.98	+5.51
ACEBench	Gemma 4 E4B	5.1	8.7	80.5	5.7	+1.10	-0.12	+8.98	+9.96
ACEBench	Gemma 4 12B	7.3	7.0	83.2	2.5	+1.33	-0.11	+7.82	+9.05
ACEBench	Gemma 4 26B-A4B	8.1	4.5	85.7	1.7	+1.34	-0.03	+4.47	+5.78
ACEBench	Gemma 4 31B	6.5	3.8	87.7	2.1	+0.72	-0.11	+4.22	+4.83
ACEBench	Ministral 3 3B	7.1	7.3	78.6	7.0	+1.83	-0.80	+5.39	+6.42
ACEBench	Ministral 3 8B	5.3	3.9	87.9	2.9	+1.60	-0.42	+5.03	+6.21
ACEBench	Ministral 3 14B	3.6	3.0	90.3	3.1	+0.81	-0.37	+3.37	+3.80
ACEBench	Qwen 2.5 3B	16.9	6.2	63.1	13.8	+4.42	-0.88	+3.78	+7.32
ACEBench	Qwen 2.5 7B	17.3	3.1	71.9	7.7	+4.25	-0.23	+5.24	+9.26
ACEBench	Qwen 2.5 14B	13.0	2.7	77.8	6.5	+3.01	-0.22	+4.95	+7.74
ACEBench	Qwen 2.5 32B	9.6	3.1	84.8	2.5	+1.66	-0.11	+4.28	+5.83
ACEBench	Qwen 3.5 4B	8.4	2.1	85.6	3.9	+1.25	-0.33	+0.89	+1.81
ACEBench	Qwen 3.5 9B	4.5	1.7	90.5	3.2	+0.62	-0.35	+1.87	+2.14
ACEBench	Qwen 3.5 35B-A3B	2.6	2.5	92.5	2.5	+0.39	-0.25	+0.59	+0.74

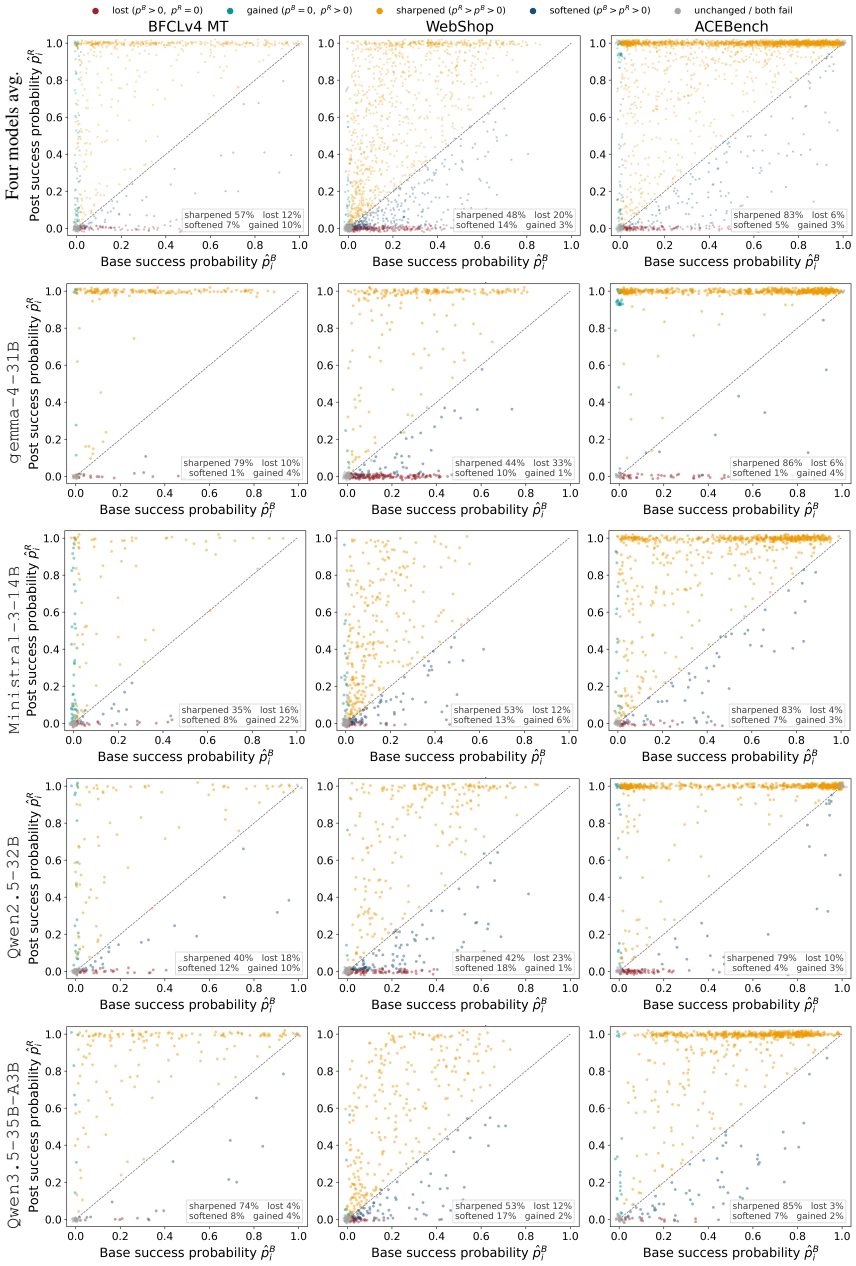


Figure 25: **Per-task success-rate movement from the base to the post-trained model** for the largest backbone of each family on BFCLv4 MT, WebShop, and ACEBench. Each point is one task, placed by its empirical success rate over 128 rollouts under the base model ($\hat{p}_i^B$, x-axis) and the post-trained model ($\hat{p}_i^R$, y-axis). Colors mark how each task moves, i.e., lost ($p^B > 0$, $p^R = 0$), gained ($p^B = 0$, $p^R > 0$), sharpened ($p^R > p^B > 0$), softened ($p^B > p^R > 0$), or unchanged, and the inset of each panel reports the proportion of each type among all tasks. The top row pools the four models, and each remaining row shows one model. Most tasks move far from the diagonal. Many sharpened tasks reach the top edge, while the lost tasks along the bottom edge show the coverage lost by post-training.

F.3 Which Tasks Pay the Tax?

The calibrated tax of a benchmark-backbone pair is an average of per-task contributions. Let $a_i(K) = \sum_{k=1}^{K-1} \left[\text{pass@}K_i - \text{pass@}k_i\right]$ be the raw scalability of task i. Then, Eq. (2) gives

$$\text{Tax}_S(K) = \frac{1}{N} \sum_{i=1}^{N} \left[\frac{a_i^B(K)}{(K-1)\,(1 - \text{pass@}1_B)} - \frac{a_i^P(K)}{(K-1)\,(1 - \text{pass@}1_P)} \right], \tag{9}$$

where each policy is normalized by its benchmark-level accuracy headroom. To see on which tasks the tax is charged, we rank the tasks of each benchmark-backbone pair by their signed contribution in Eq. (9) at K=128 and plot the scaling curves of the five highest and five lowest tasks under both policies. Figure 26 averages these curves over all 14 backbones, and Figures 27–30 show the individual tasks for each family.

The two extremes look alike across families and scales. On the highest-tax tasks, the post-trained policy has become deterministic, so its curve is flat at 100% or at 0% for every budget and gains nothing from retries. The base model instead succeeds only rarely, often once in 128 rollouts, so its curve stays low for small k and rises to 100% only near the full budget (Figure 26, top). This is Proposition 1 at the level of a single task, since the tax is charged where post-training removes a rare success that retries could recover. The lowest-tax tasks show the opposite pattern (Figure 26, bottom). There, the post-trained policy holds the rare success, and the base model either never succeeds or saturates within a few rollouts. The per-family figures show how this depends on scale somehow.

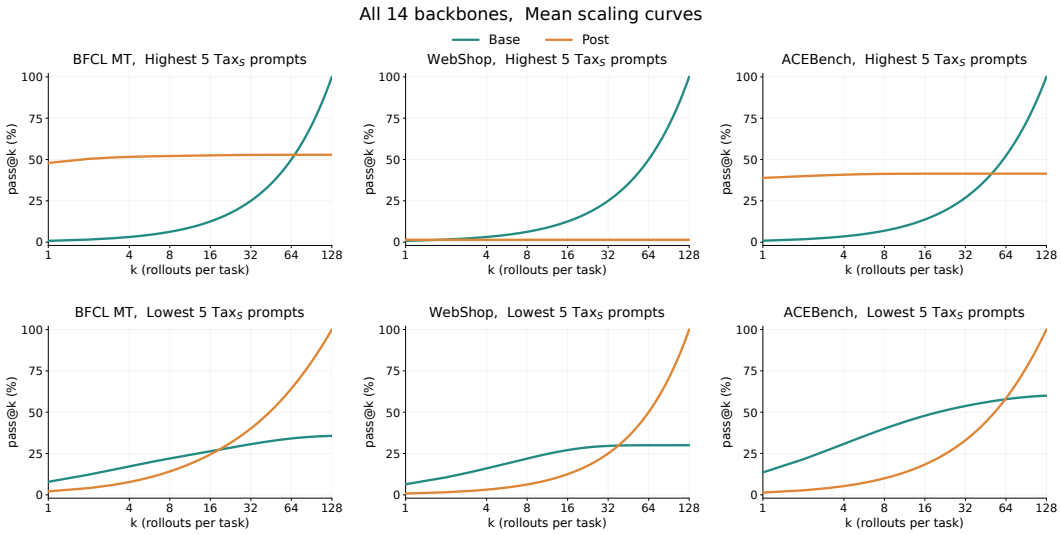


Figure 26: **Mean scaling curves of the tasks with the highest and lowest tax contributions, averaged over all 14 backbones.** For each benchmark and backbone, we select the five tasks with the highest (top row) and lowest (bottom row) signed contributions to $\mathrm{Tax}_S(128)$ in Eq. (9). Each panel shows the mean pass@k curve of the base (teal) and post-trained (orange) models over the selected tasks of all 14 backbones (70 tasks per panel).

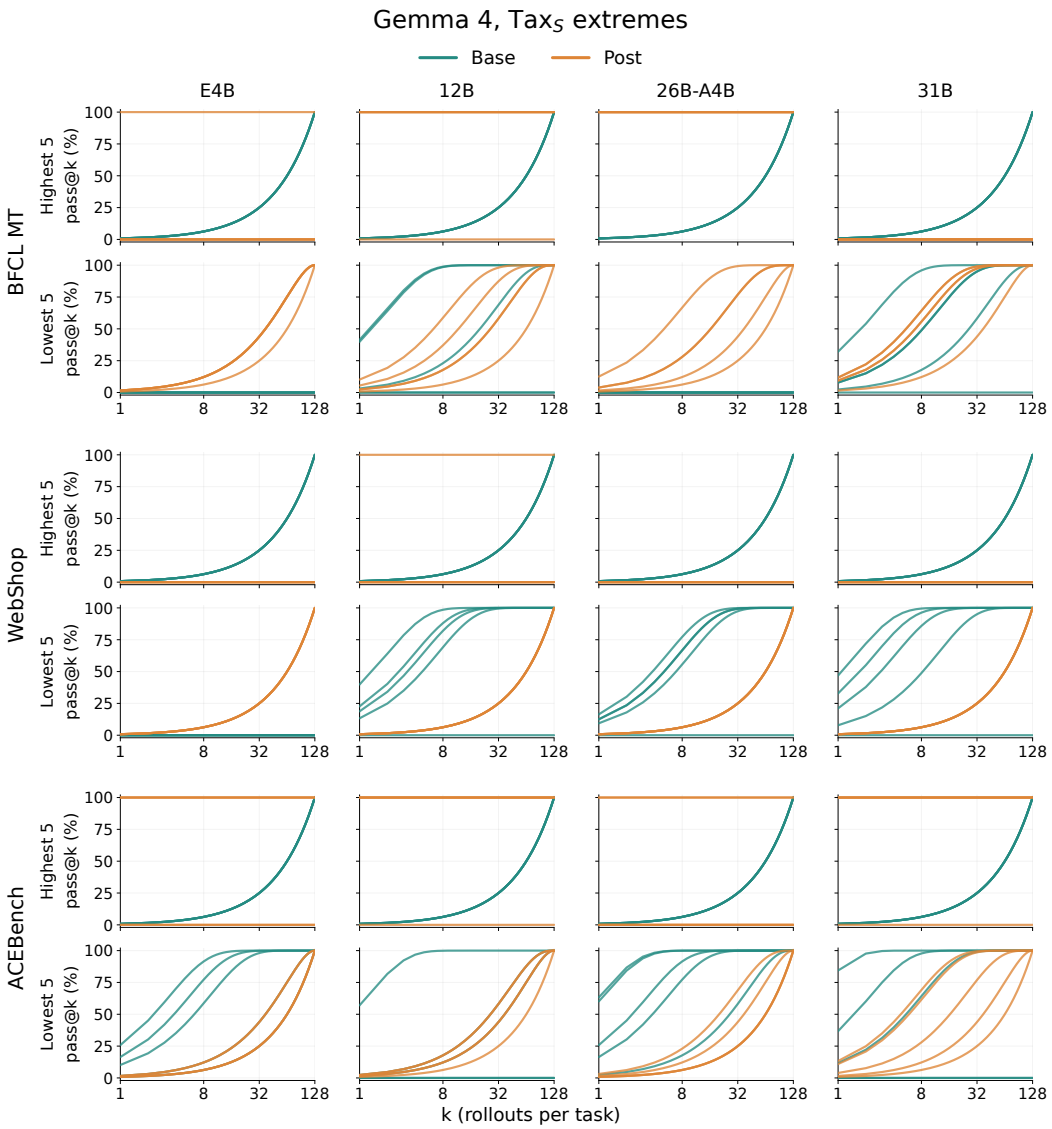


Figure 27: **Scaling curves of the tasks with the highest and lowest tax contributions for Gemma-4** (E4B, 12B, 26B-A4B, and 31B from left to right). For each benchmark (BFCL MT, WebShop, and ACEBench from top to bottom), the upper row shows the five tasks with the highest signed contributions to $\mathrm{Tax}_S(128)$ (Eq. (9)) and the lower row the five with the lowest, selected separately for each backbone. Each panel shows the pass@k curves of the base (teal) and post-trained (orange) models on the same five tasks, computed from 128 rollouts per task without averaging.

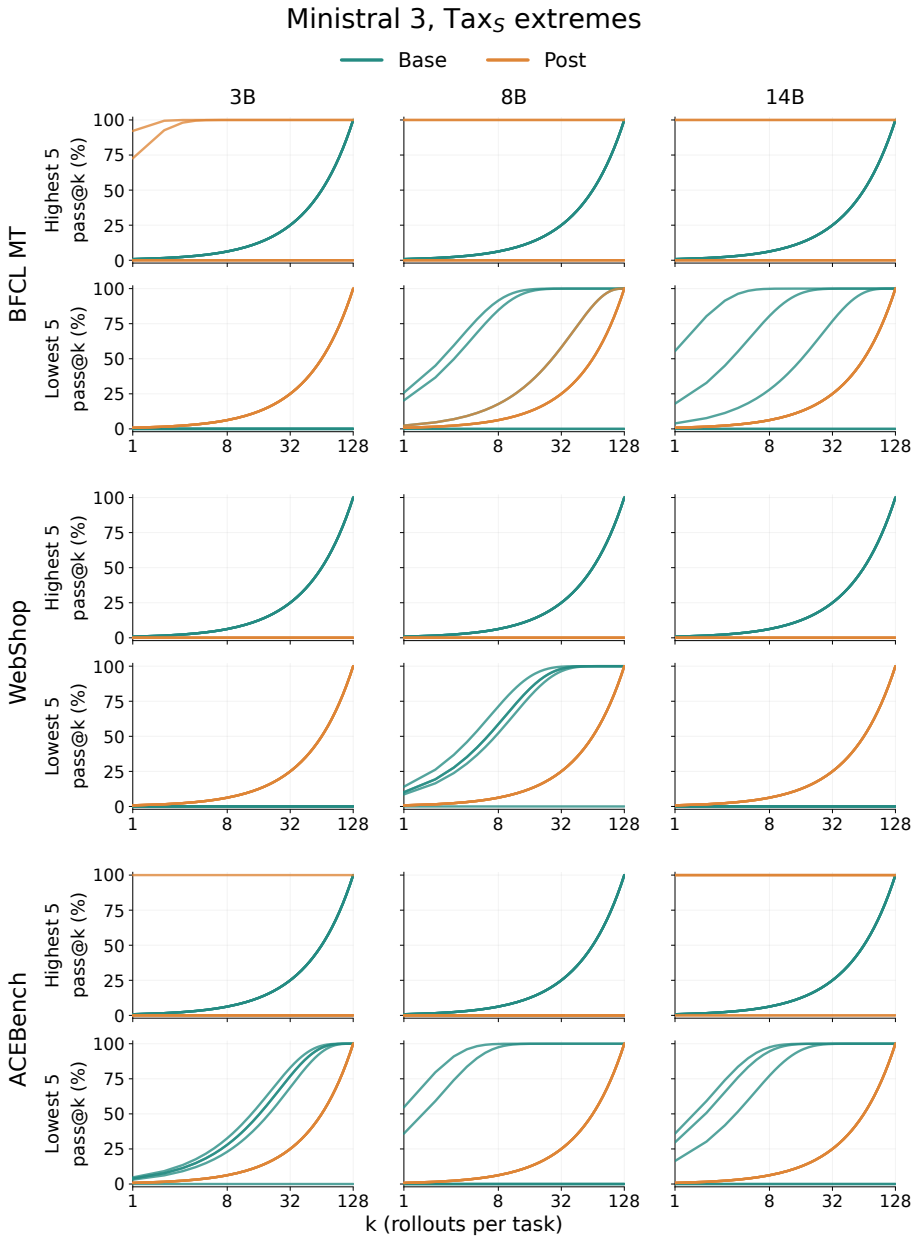


Figure 28: **Scaling curves of the tasks with the highest and lowest tax contributions for Ministral-3** (3B, 8B, and 14B from left to right). For each benchmark (BFCL MT, WebShop, and ACEBench from top to bottom), the upper row shows the five tasks with the highest signed contributions to $\mathrm{Tax}_S(128)$ (Eq. (9)) and the lower row the five with the lowest, selected separately for each backbone. Each panel shows the pass@k curves of the base (teal) and post-trained (orange) models on the same five tasks, computed from 128 rollouts per task.

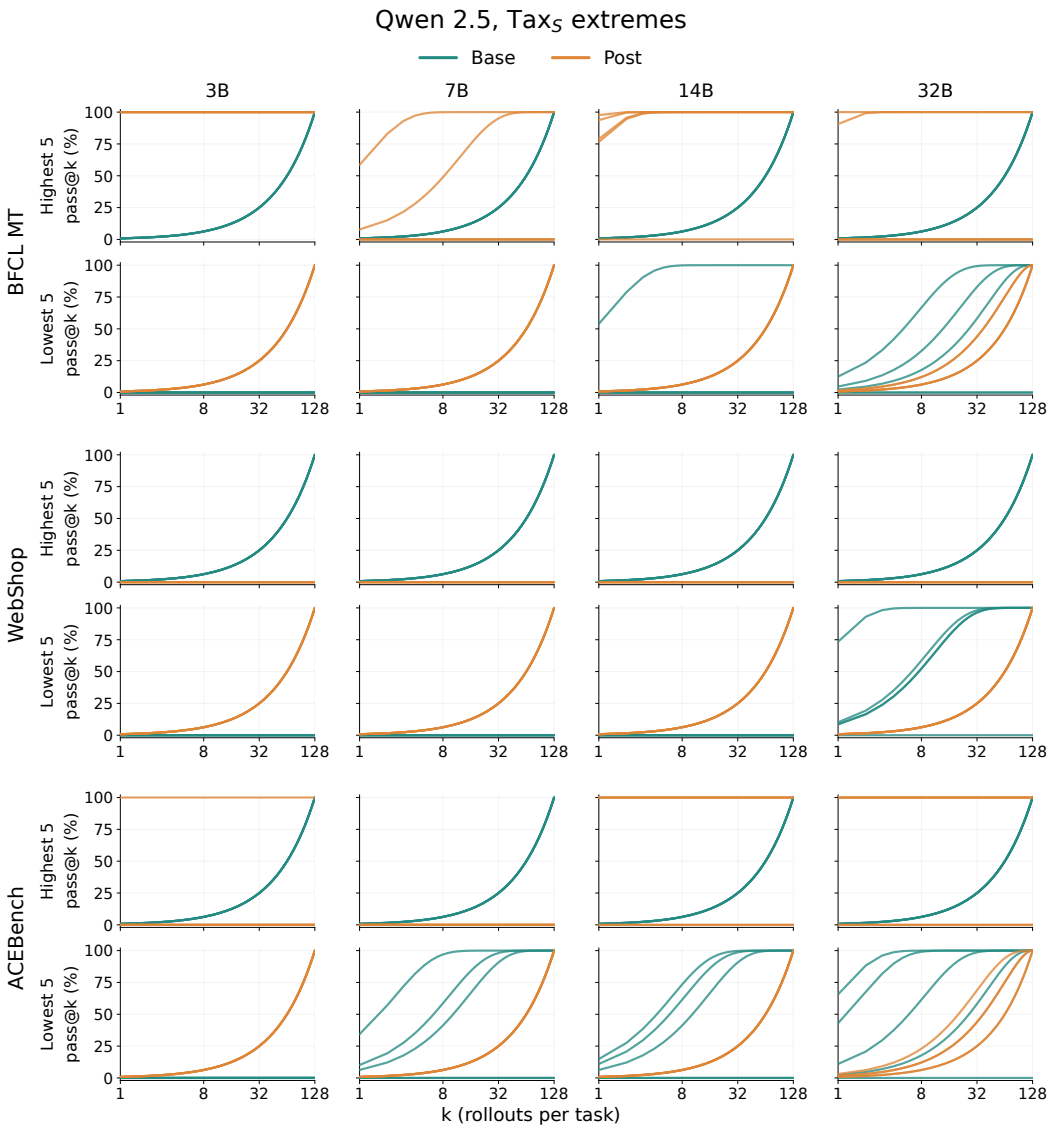


Figure 29: **Scaling curves of the tasks with the highest and lowest tax contributions for Qwen2.5** (3B, 7B, 14B, and 32B from left to right). For each benchmark (BFCL MT, WebShop, and ACEBench from top to bottom), the upper row shows the five tasks with the highest signed contributions to $\mathrm{Tax}_S(128)$ (Eq. (9)) and the lower row the five with the lowest, selected separately for each backbone. Each panel shows the pass@k curves of the base (teal) and post-trained (orange) models on the same five tasks, computed from 128 rollouts per task.

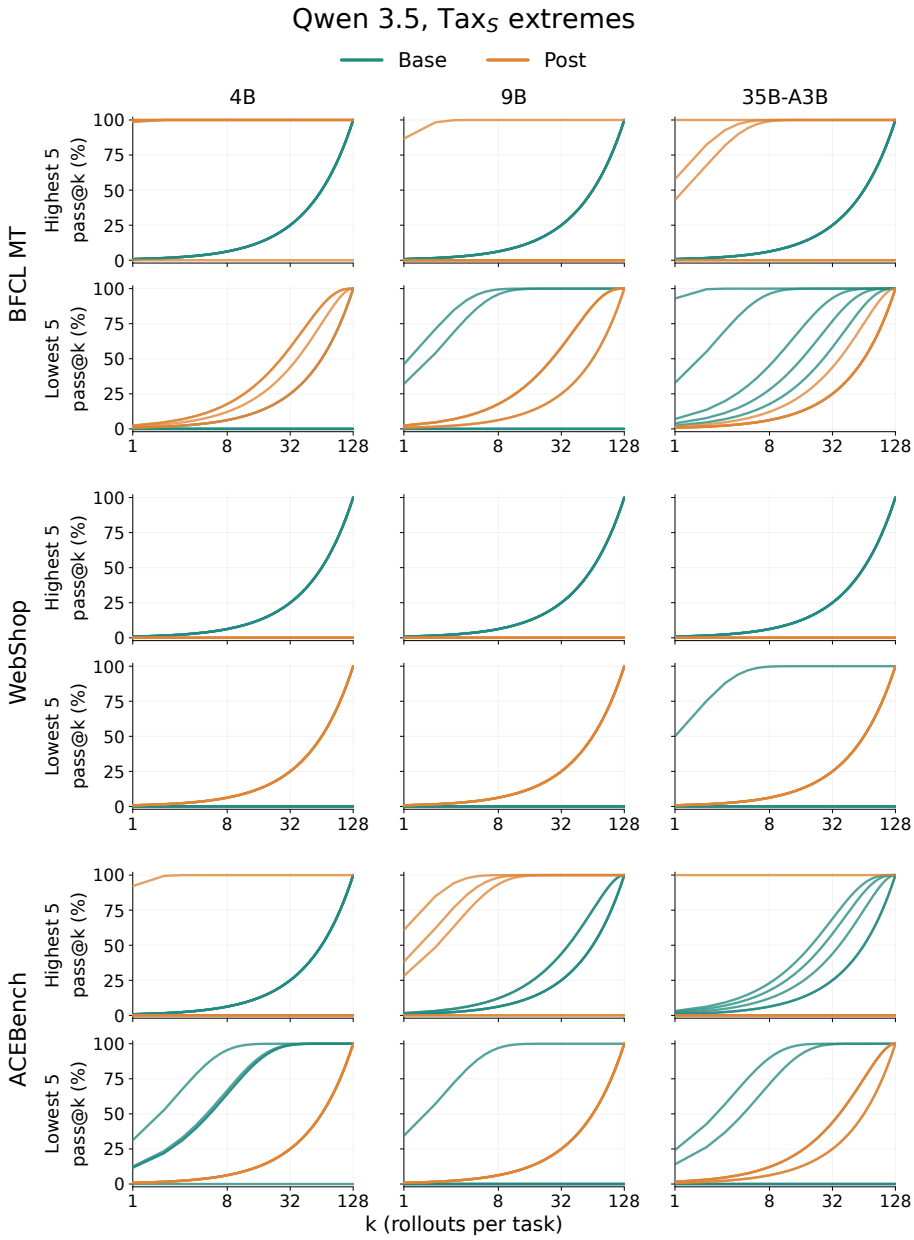


Figure 30: **Scaling curves of the tasks with the highest and lowest tax contributions for Qwen3.5** (4B, 9B, and 35B-A3B from left to right). For each benchmark (BFCL MT, WebShop, and ACEBench from top to bottom), the upper row shows the five tasks with the highest signed contributions to $\mathrm{Tax}_S(128)$ (Eq. (9)) and the lower row the five with the lowest, selected separately for each backbone. Each panel shows the pass@k curves of the base (teal) and post-trained (orange) models on the same five tasks, computed from 128 rollouts per task.

F.4 Tax-Guided Routing between the Base and the Post-trained Policy

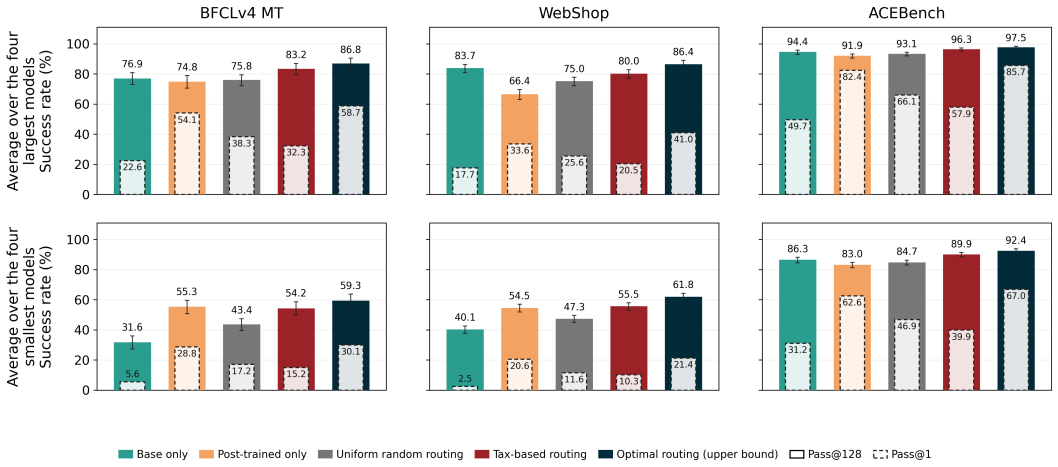


Figure 31: **Tax-guided routing between the base and post-trained policies.** Success rates averaged over the four largest (top) and the four smallest (bottom) backbones on BFCLv4 MT, WebShop, and ACEBench. Each task is served by one policy chosen by one of five rules, i.e., base only (teal), post-trained only (orange), uniform random routing (gray), tax-based routing (red), and optimal routing (navy). Tax-based routing chooses the policy for each task based on its $\mathrm{Tax}_S(32)$ estimated from an early pilot 32-rollout. Optimal routing is an oracle selector that assigns each task to the better policy on the full 128 rollouts, which upper-bounds any router. Filled bars show pass@128 and dashed inset bars show pass@1 under the same assignments, both computed from all 128 rollouts of the selected policy including the pilot rollouts. Error bars show the bootstrap uncertainty of pass@128 (2,000 paired task resamples).

Setup. If the tax tells us on which tasks retries pay off, a cheap estimate of it could also decide which of the two policies should receive the test-time budget for each task. For each base/post-trained pair and benchmark, we compare five ways of assigning each task to one policy. Besides the two single policies, we consider uniform random routing, which assigns each task to either policy with equal probability, tax-based routing, which chooses the policy for each task based on its calibrated tax $\mathrm{Tax}_S(32)$ estimated from the early pilot 32-rollout, and optimal routing, a hindsight oracle that assigns each task to the better policy on the full 128 rollouts. The reported pass@128 uses all 128 rollouts of the selected policy including its pilot rollouts, so the pilot rollouts spent on the other policy are the overhead of routing. Figure 31 averages the results over the four largest and the four smallest backbones.

Results. Tax-based routing combines the strengths of the two policies. Its pass@128 exceeds that of the post-trained model in five of the six panels, by up to 13.6 points (WebShop, largest backbones), and the only exception (BFCL, smallest backbones) is within 1.1 points. At the same time, its pass@1 exceeds that of the base model in every panel. In four of the six panels, it reaches a higher pass@128 than both single policies, and in the other two, where one policy is far better than the other, it stays within a few points of the better one. It also beats uniform random routing in every panel, so the gain comes from the tax signal rather than from simply mixing the two policies, and it stays close to the performance of oracle routing. **A tax estimate from a small pilot thus recovers most of the coverage that post-training gives up, and often more, without falling back to the low single-shot accuracy of the base model.** Since its pass@1 lies between those of the two single policies, tax-based routing is best suited to large test-time budgets, in line with the budget dependence discussed in §3.

G ADDITIONAL RESULTS ON PTGS

This section gives additional results for §5. Here, we show the training dynamics behind Table 2 (§G.1), an ablation on the temperature spread τ (§G.2), a comparison with fixed rollout temperatures and truncated decoding rules (§G.3), and an inference-time variant of PTGS (§G.4).

G.1 TRAINING DYNAMICS OF PTGS

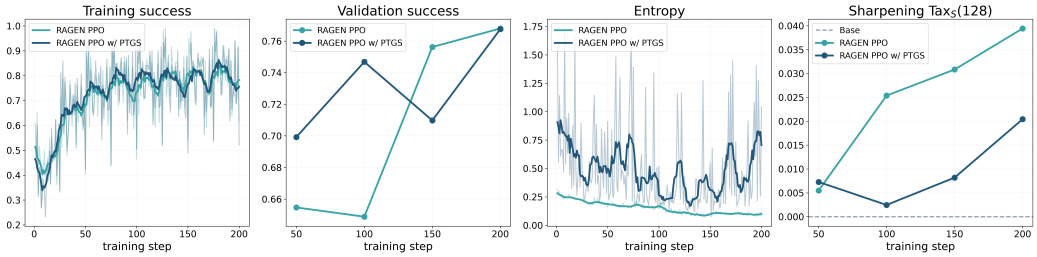


Figure 32: **Training dynamics of PPO and PPO with PTGS on FrozenLake.** The panels show, from left to right, training success, validation success, average entropy of the output token distribution (thin lines for raw values and thick lines for the moving average), and $\mathrm{Tax}_S(128)$ of the checkpoints at steps 50, 100, 150, and 200. The two methods have nearly identical training success and end at the same validation success, but PTGS keeps a much higher entropy throughout training and pays a smaller tax at every checkpoint after step 50.

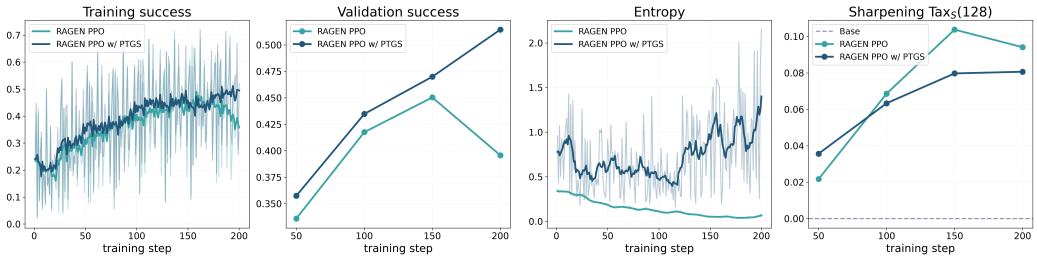


Figure 33: **Training dynamics of PPO and PPO with PTGS on Sokoban.** The panels show, from left to right, training success, validation success, average entropy of the output token distribution (thin lines for raw values and thick lines for the moving average), and $\mathrm{Tax}_S(128)$ of the checkpoints at steps 50, 100, 150, and 200. PTGS reaches higher validation success at every checkpoint and keeps improving until step 200, while PPO drops after step 150. The entropy of PPO decays toward zero while that of PTGS stays high, and PTGS pays a smaller tax than PPO from step 100 on.

Training dynamics of PTGS. Figures 32 and 33 show the training progress of PPO and PPO with PTGS behind the final results in Table 2. In both environments, the two methods have almost the same training success, so PTGS does not make the training tasks easier or harder on average. The other panels, however, show clear differences. First, the entropy of PPO decreases steadily toward zero, which is the usual entropy collapse of fixed-temperature RL, whereas PTGS keeps the entropy several times higher throughout training. The entropy of PTGS also rises repeatedly, which is consistent with the sampler heating the prompts that the policy keeps failing. Second, the tax of PPO grows over training in both environments, but PTGS pays a smaller tax at most checkpoints, and on Sokoban its tax stops growing after step 150. Third, PTGS matches PPO in final validation success on FrozenLake and clearly outperforms it on Sokoban, where PPO degrades after step 150 while PTGS keeps improving. Overall, **the per-prompt adaptive temperature of PTGS stabilizes training and yields a final policy that is at least as accurate, more exploratory, and less taxed**.

G.2 ABLATION ON THE TEMPERATURE SPREAD τ OF PTGS

The spread τ sets the temperature range $[1/\tau, \tau]$ of PTGS. As Figure 34 shows, a larger τ keeps a higher entropy throughout training while barely affecting training success. The spread needed depends on the environment: on FrozenLake, even $\tau = 1.2$ improves pass@128 and reduces the

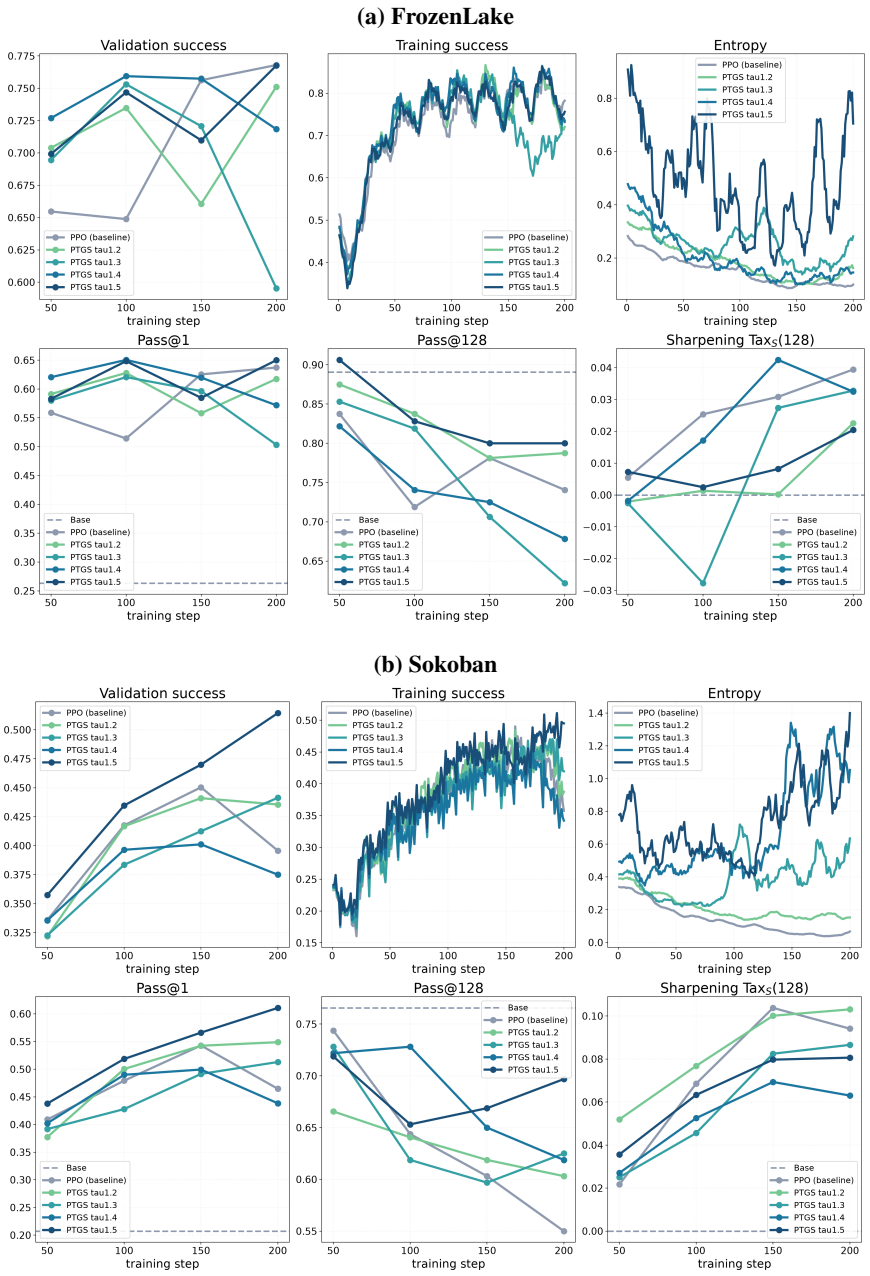


Figure 34: **Ablation on the temperature spread τ of PTGS on FrozenLake (a) and Sokoban (b).** We compare PPO with PTGS for $\tau \in \{1.2, 1.3, 1.4, 1.5\}$ against the PPO baseline. In each block, the top row shows validation success, training success, and average entropy of the output token distribution over training, and the bottom row shows pass@1, pass@128, and Tax$_S(128)$ of the intermediate checkpoints, with dashed lines for the base model. A larger τ keeps a higher entropy in both environments. $\tau = 1.5$ achieves the best final pass@1 and pass@128 in both environments and the smallest tax on FrozenLake. On Sokoban, the smallest temperature $\tau = 1.2$ only slightly raises the entropy above PPO and pays a similar tax.

tax over PPO, whereas on the harder Sokoban, $\tau = 1.2$ barely raises the entropy and only $\tau \geq 1.3$ improves pass@128. We use $\tau = 1.5$ for PPO in §5, since it gives the best accuracy and coverage in both environments.

G.3 Fixed Rollout-Temperature and Decoding Ablations of the PPO Baseline

Table 9: **Fixed rollout-temperature ablation of PPO, compared with PTGS**. Each PPO row samples its training rollouts at a fixed temperature T, where T=1.0 (shaded) is the PPO baseline of the main text. Values are the mean over five runs ± the 95% confidence interval.

	Sokoban				FrozenLake			
Method	pass@1	pass@128	pass^{128}	Tax$_S$(128)	pass@1	pass@128	pass^{128}	Tax$_S$(128)
Qwen2.5-7B-Instruct (base)	20.7	76.6	0.0	–	26.3	89.1	0.0	–
PPO, T = 0.5	45.6 ± 7.0	62.8 ± 10.7	26.2 ± 8.6	**0.073 ± 0.025**	58.7 ± 15.0	76.9 ± 12.1	41.2 ± 26.5	**0.004 ± 0.053**
PPO, T = 0.7	40.2 ± 26.8	51.9 ± 33.6	30.9 ± 23.1	0.075 ± 0.049	63.7 ± 7.1	70.3 ± 4.9	**54.7 ± 11.4**	0.046 ± 0.032
PPO, T = 1.0 (default)	46.5 ± 14.7	55.0 ± 11.3	36.2 ± 13.1	0.094 ± 0.016	63.7 ± 5.9	74.1 ± 7.2	50.9 ± 10.1	0.039 ± 0.026
PPO, T = 1.3	53.6 ± 7.6	58.8 ± 10.6	47.2 ± 4.8	0.100 ± 0.022	60.9 ± 12.1	70.3 ± 12.0	47.5 ± 17.5	0.042 ± 0.026
PPO, T = 1.5	57.4 ± 11.6	64.4 ± 11.1	**52.2 ± 7.8**	0.102 ± 0.017	62.8 ± 6.8	68.4 ± 6.9	**54.7 ± 10.0**	0.055 ± 0.015
PPO with PTGS	**61.1 ± 8.6**	**69.7 ± 9.7**	49.7 ± 14.9	0.081 ± 0.039	**65.0 ± 3.2**	**80.0 ± 8.3**	42.5 ± 15.2	0.020 ± 0.047

Table 10: **Rollout decoding ablation of PPO, compared with PTGS**. Each PPO row applies one truncation rule to the training rollouts at T=1.0, and the shaded row is the PPO baseline with ancestral sampling (the same runs as T=1.0 in Table 9). Values are the mean over five runs ± the 95% confidence interval.

	Sokoban				FrozenLake			
Method	pass@1	pass@128	pass^{128}	Tax$_S$(128)	pass@1	pass@128	pass^{128}	Tax$_S$(128)
Qwen2.5-7B-Instruct (base)	20.7	76.6	0.0	–	26.3	89.1	0.0	–
PPO, ancestral (default)	46.5 ± 14.7	55.0 ± 11.3	36.2 ± 13.1	0.094 ± 0.016	63.7 ± 5.9	74.1 ± 7.2	50.9 ± 10.1	0.039 ± 0.026
PPO, top-p 0.9	42.5 ± 12.7	46.6 ± 15.2	33.1 ± 9.2	0.111 ± 0.012	**66.4 ± 6.2**	74.1 ± 7.1	**52.8 ± 13.3**	0.041 ± 0.039
PPO, top-p 0.95	47.3 ± 2.8	55.9 ± 11.4	36.6 ± 9.9	0.091 ± 0.054	61.9 ± 8.0	72.5 ± 4.5	46.2 ± 16.4	0.030 ± 0.027
PPO, top-k 20	45.5 ± 9.3	55.3 ± 14.2	37.5 ± 7.3	0.092 ± 0.028	61.3 ± 7.3	75.6 ± 13.2	40.0 ± 24.7	0.031 ± 0.034
PPO, top-k 40	45.1 ± 19.4	57.2 ± 15.3	27.8 ± 18.3	0.082 ± 0.009	64.6 ± 1.2	75.6 ± 6.2	51.9 ± 9.4	0.034 ± 0.024
PPO, min-p 0.05	44.7 ± 9.6	55.0 ± 18.0	36.9 ± 6.9	**0.076 ± 0.048**	61.3 ± 6.7	75.3 ± 15.3	44.1 ± 12.8	**0.011 ± 0.080**
PPO, min-p 0.1	41.1 ± 13.4	55.0 ± 19.1	29.1 ± 10.5	0.081 ± 0.039	60.0 ± 8.5	75.0 ± 9.9	40.6 ± 14.2	0.028 ± 0.033
PPO with PTGS	**61.1 ± 8.6**	**69.7 ± 9.7**	**49.7 ± 14.9**	0.081 ± 0.039	65.0 ± 3.2	**80.0 ± 8.3**	42.5 ± 15.2	0.020 ± 0.047

Is PTGS just a hotter or truncated sampler? PTGS changes only the temperature of the training rollouts, so one may ask whether a fixed rollout temperature or a fixed truncation rule gives the same benefit. Tables 9 and 10 vary only the rollout sampler of PPO under the setup of Table 2 (Appendix C.5). The first table uses a fixed temperature $T \in \{0.5, 0.7, 1.0, 1.3, 1.5\}$, where T=1.5 is the highest temperature that PTGS can assign. The second keeps T=1.0 and truncates the rollout distribution with top-p, top-k, or min-p decoding. The results give several implications. First, a fixed rollout temperature moves the policy along the accuracy-coverage trade-off without improving both. On Sokoban, a higher temperature tends to improve pass@1 and consistency, but it also increases the tax. On FrozenLake, every temperature above the default lowers pass@128 and increases the tax, and the only fixed temperature with a clearly smaller tax (T=0.5) has the lowest pass@1. Second, truncation rules barely change the baseline. Most variants stay within about two points of the ancestral baseline in pass@128, and none of them gives a clear gain in pass@1 or pass@128. Third, PTGS is the only sampler that improves accuracy and coverage at the same time. It gives the best pass@128 in both environments and the best or near-best pass@1, with a smaller tax than every fixed temperature $T \geq 1.0$. Its consistency on FrozenLake is lower than that of the default, although the intervals overlap. **Prompt-adaptive tempering is therefore not equivalent to a global temperature or a fixed truncation of the rollout distribution**. This agrees with Theorem 4, which suggests that what matters is heating the prompts that the policy keeps failing, rather than heating or truncating all prompts in the same way.

G.4 Inference-Time PTGS Application

Estimate-then-sample decoding. The tempering rule of PTGS needs only a per-prompt difficulty estimate, so it can also be applied to a frozen policy at test time. For each test prompt x, we split a budget of B rollouts into m pilot rollouts and $n = B - m$ solution rollouts. We draw m pilot rollouts at a reference temperature T_{ref} and count their successes s_x, form the posterior

$\mathrm{Beta}\big(2\tilde{p}+s_x,\ 2(1-\tilde{p})+m-s_x\big)$ with the same pivot-centered prior as in training, draw $\hat{p}_x$ from it by Thompson sampling, and generate the n solution rollouts at $T_x = T_{\mathrm{ref}}\,\tau^{h(\hat{p}_x)} \in [T_{\mathrm{ref}}/\tau,\ \tau T_{\mathrm{ref}}]$ with h from Eq. (4). Unlike training-time PTGS, the posterior is built from scratch for each prompt and discarded afterwards, without a forgetting factor or pivot schedule, and no log-probability correction is needed because the policy is not updated. Only the n solution rollouts are scored, and pass@k for $k \le n$ is computed from their success count with the unbiased estimator of §2.

Setup. We apply inference-time PTGS to the frozen `Qwen2.5-7B-Instruct`, i.e., the base model of Table 2, on the same 64 offline evaluation tasks of Sokoban and FrozenLake (§C.5). We use $B = 256$ with $m = n = 128$, $T_{\mathrm{ref}} = 0.5$ (the RAGEN evaluation default), and $\tilde{p} = 0.5$, and vary $\tau \in \{1.2, 1.3, \ldots, 2.0\}$. The pilot rollouts are scored by the same binary environment success as the evaluation, so this setting measures what per-prompt adaptivity can offer with a reliable difficulty signal. As a baseline, we evaluate a global temperature $T \in \{0.1, 0.3, 0.5, 0.7, 1.0, 1.3, 1.5\}$ with ancestral sampling and 128 scored rollouts per task.

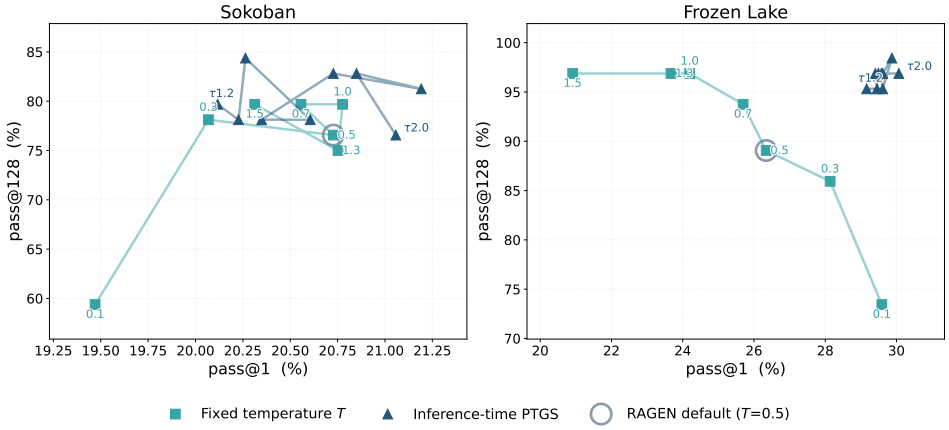


Figure 35: **Inference-time PTGS versus global temperature scaling.** pass@1 (x-axis) and pass@128 (y-axis) of the frozen `Qwen2.5-7B-Instruct` on 64 tasks of each environment, where the upper right is better. Squares show global temperatures $T \in [0.1, 1.5]$, triangles show inference-time PTGS with $\tau \in [1.2, 2.0]$ and $T_{\mathrm{ref}} = 0.5$, and the ring marks the default inference temperature $T = 0.5$. PTGS lies beyond the global-temperature curve in both environments.

Can per-prompt tempering improve the trade-off without training? Appendix E.4 showed that a global temperature cannot improve accuracy and coverage together. Figure 35 tests whether inference-time PTGS does better on a frozen policy. On FrozenLake, the global temperature gives a clear monotone trade-off, where a lower temperature improves pass@1 but reduces pass@128. All PTGS configurations lie beyond this curve and get close to the accuracy of the coldest temperature and the coverage of the hottest one at the same time. On Sokoban, the global temperature shows no clear trade-off above T=0.1, and only two PTGS configurations exceed every fixed temperature on both axes, by one or two of the 64 tasks in pass@128.

H Deeper Analysis of PTGS

H.1 Accuracy of the Difficulty Estimate and Realized Temperatures

Is the difficulty estimate of PTGS accurate? PTGS chooses the temperature of each prompt from a running Beta posterior over its success rate, so its benefit depends on whether this estimate predicts the actual success of the policy. To check this, we record the posterior mean $\bar{p}_x$ of each prompt before its rollout group is drawn and compare it with the realized success rate of the group. From early in training, the estimate separates the 8 prompts of each step well by their realized success, with an average Pearson correlation of 0.81 on Sokoban and 0.71 on FrozenLake. On FrozenLake, the correlation becomes stronger as the posterior accumulates evidence (Figure 36). Not that the estimate is conservative (the realized success is about 0.1 higher on average) because the discounted counts still reflecting the weaker policy of earlier steps and with the prior centered on a pivot below $1/2$. Since a conservative estimate errs toward heating, PTGS tends to explore more.

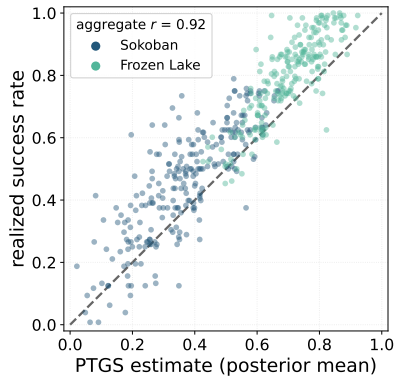


Figure 36: **The difficulty estimate of PTGS tracks the realized success at the end of training.** Each point is one of the last 50 training steps of one of five PPO w/ PTGS runs per environment. The x-axis is the mean posterior estimate $\bar{p}_x$ over the 8 prompts of the step, recorded before the rollouts are drawn, and the y-axis is the realized success rate of its 128 rollouts.

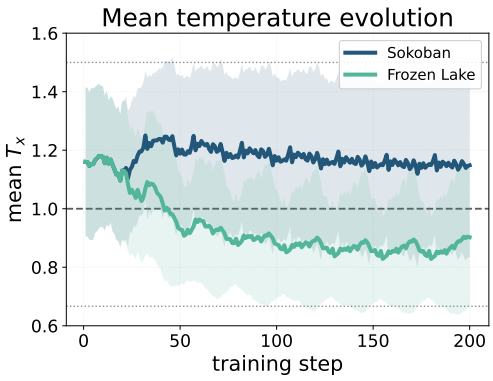


Figure 37: **Mean rollout temperature of PTGS over training on Sokoban and FrozenLake.** Mean temperature T_x over the 8 prompts of each step (9-step moving average over five runs). Bands show ± 1 standard deviation of T_x across the prompts of a step. On average, PTGS heats on the harder Sokoban and cools on FrozenLake once the policy solves most prompts, while using a range of temperatures across prompts in both environments.

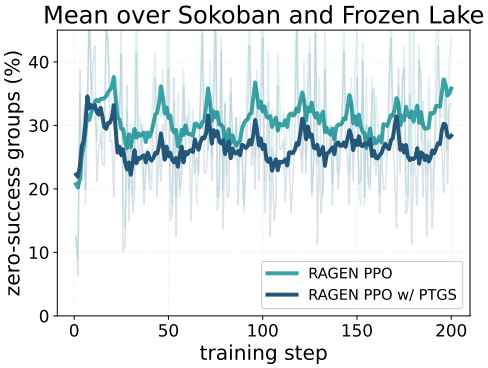


Figure 38: **PTGS reduces the proportion of zero-success rollout groups.** Percentage of the rollout groups at each training step in which none of the 16 rollouts succeeds, i.e., groups that carry no success signal for the update, for PPO and PPO with PTGS. The proportion is measured before the reward-variance filter and averaged over Sokoban and FrozenLake. Thin lines show the mean over the ten runs of each method (five per environment) with a 9-step moving average as the think line.

How hot does PTGS actually sample? Figure 37 shows that PTGS does not simply sample at a higher temperature than PPO. On Sokoban, where the policy remains weak, the mean temperature stays above $T=1$ throughout training. On FrozenLake, where the policy solves most prompts after the first 50 steps, PTGS cools on average, from about 1.1 early in training to below 0.9 at the end. In both environments, the temperatures of prompts within the same step vary widely, with a standard deviation of about 0.2 to 0.3. In other words, PTGS heats the prompts that the policy fails and cools the ones it solves at the same time, instead of shifting a single global temperature.

Does PTGS give RL more to learn from? A rollout group in which none of the 16 rollouts succeeds carries no success signal for the update, and Theorem 4 (i) states that heating a hard prompt makes such groups rarer. Averaged over the two environments, PTGS keeps the proportion of these zero-success groups below that of PPO for most of training (Figure 38). That is, PTGS indeed turns more of the rollout budget into groups that carry a learning signal.

H.2 Does PTGS Induce a More Exploratory Policy?

Is the higher entropy just a higher temperature? The entropy curves in Appendix G.1 compare PTGS with PPO at the default temperature $T=1$. However, that entropy is computed at the rollout temperature of each run, so part of the gap could come from PTGS sampling at higher temperatures. Therefore, we also compare PTGS with PPO runs that sample every rollout at $T=1.3$ or $T=1.5$

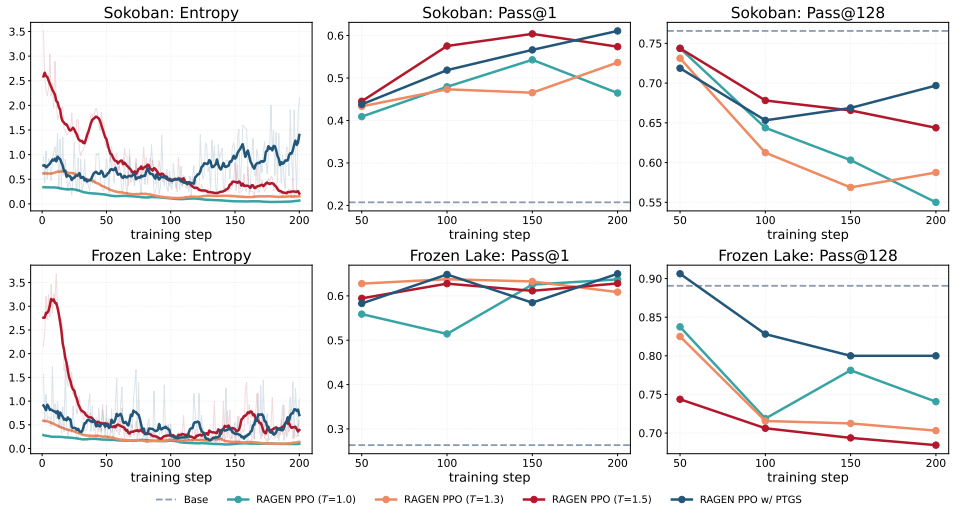


Figure 39: **PTGS versus PPO with a globally higher temperature** on Sokoban (top) and FrozenLake (bottom). We compare PPO at the default rollout temperature (T=1.0), PPO with every rollout sampled at T=1.3 or T=1.5 (the highest temperature PTGS can assign), and PPO with PTGS (τ=1.5). The left panels show the average entropy of the output token distribution during training, computed at the rollout temperature of each run, with thin lines for the mean over five runs and thick lines for a 9-step moving average. The middle and right panels show pass@1 and pass@128 of the intermediate checkpoints, averaged over five runs, and the dashed line marks the base model.

(Figure 39). On Sokoban, PTGS keeps a much higher entropy than both heated PPO variants over the last 50 steps. On FrozenLake, PPO at T=1.5 reaches an entropy similar to that of PTGS, but only by sampling every rollout at a temperature about 60% above the mean temperature of PTGS.

Does the extra exploration improve coverage? The extra entropy also has different effects under global heating and under PTGS (Figure 39, middle and right). On Sokoban, heating every rollout to T=1.5 improves both pass@1 and pass@128 over the default, but both remain below PTGS. On FrozenLake, heating even lowers pass@128 below the default. PTGS instead achieves the highest pass@1 and pass@128 at step 200 in both environments. In short, **the higher entropy of PTGS is not simply due to a higher sampling temperature, and unlike the entropy from a global temperature, it comes with higher coverage**.

H.3 Case Study: Tasks That PPO Loses and PTGS Recovers

We say that an RL post-training run *loses* a task if the base model solves it at least once in 128 rollouts but the run never does. Among the tasks lost by a PPO run, a PPO w/ PTGS run solves about half, whereas the reverse rate is much lower. Figures 40 and 41 show one such task per environment. In both cases, the base model solves the task only occasionally, and PPO never solves it in four of its five runs. PTGS, in contrast, solves it in most runs and masters it in some. This is consistent with Appendix D.4, i.e., by heating a prompt that the policy keeps failing, PTGS keeps it contributing to learning until the policy solves it reliably.

Sokoban task `sokoban_25`: push the box onto the target on a 6×6 grid (walls in dark gray, box in brown, target circled).
Base model's winning opening: *"The box is one left of the target. Pushing it directly would move it off the target. Moving right first would place the box next to the target, then push."* `Right ||Push Right`

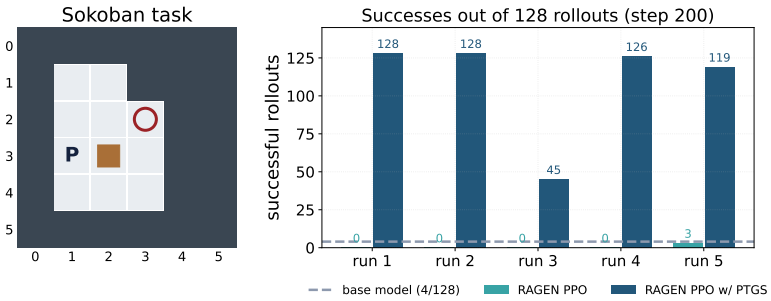


Figure 40: **A Sokoban task that PPO loses and PTGS recovers.** The above example case describes the task and the opening of a successful rollout of the base model, and the left panel shows the initial state. The right panel shows the number of successful rollouts out of 128 at step 200 in each of the five training runs of PPO and PPO with PTGS, and the dashed line marks the base model, which solves the task in 4 of 128 rollouts. PPO never solves the task in four of its five runs, but PPO w/ PTGS solves it in all five runs and masters it in two.

FrozenLake task `frozen_lake_43`: reach the goal G on slippery 4×4 ice without falling into the hole (dark circle).
Base model's winning opening: *"The player is at (2,2) and needs to move left twice to reach the goal at (3,0). However, be cautious of the hole at (3,1)."* `Left ||Left`

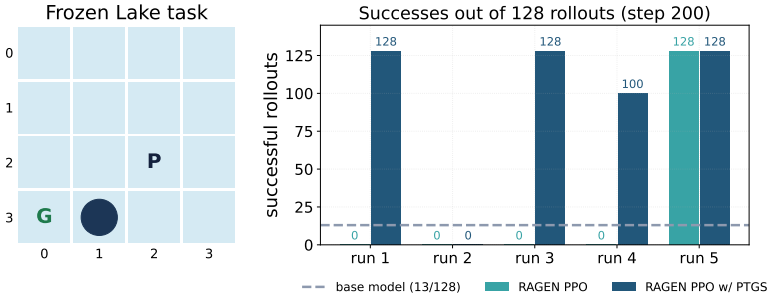


Figure 41: **A FrozenLake task that PPO loses and PTGS recovers.** The above example case describes the task and the opening of a successful rollout of the base model, and the left panel shows the initial state. The right panel shows the number of successful rollouts out of 128 at step 200 in each of the five training runs of PPO and PPO with PTGS, and the dashed line marks the base model, which solves the task in 13 of 128 rollouts. PPO never solves the task in four of its five runs, but PPO w/ PTGS solves it in four runs and masters it in three.