Task Coverage or Repeated Exposure? A Study of Learning from Execution Feedback

Abstract

Task-set scaling in execution-feedback post-training changes both task coverage and repetition under a fixed budget, so nominal task count does not specify the training allocation. We study this problem with Qwen3.5-4B and group-normalized policy gradient (PG) on nested Structured Query Language (SQL) and arithmetic task sets of $N = 64$, 256, and 1,024. We compare rollout-count and generated-response-token endpoints, measure realized task coverage, generated tokens, updates, wall-clock time, and acquisition cost, vary task composition and reference complexity, and evaluate six adaptation methods on reserved final tasks under a registered 7,200-second pipeline budget. $N = 256$ gives the highest development mean, while $N = 1{,}024$ does not continue the gain and the original quantity endpoints are close on the reserved test. Rollout-count and token endpoints produce different exposure and cost, task composition and reference complexity change outcomes at fixed N, and no tested adaptation method exceeds the unadapted checkpoint under the common time budget. These results establish that task-set scaling is a training-allocation claim: interpretation requires separating new task coverage from repeated feedback and accounting for endpoint and resource costs. This allocation diagnosis gives researchers a basis for deciding whether limited budget should purchase new tasks, repeated feedback, or further adaptation, and for judging whether an observed gain transfers when the budget or endpoint changes.

1 Introduction

Execution feedback makes it possible to judge a model's response through a concrete consequence: the rows returned by a query or the answer produced by a program. Seq2SQL uses query-execution rewards for learning, and InterCode treats code execution as an interactive feedback channel (Zhong et al. 2017; Yang et al. 2023). A training collection supplies the tasks on which these consequences are observed. Expanding that collection can expose a model to more inputs, but a fixed response budget also reduces the number of opportunities to revisit each input. For example, 1,024 prompt presentations allow sixteen visits to each of 64 tasks, four visits to each of 256 tasks, or one visit to each of 1,024 tasks. These allocations differ even before the learning algorithm changes.

Prior studies vary environment composition: ScaleEnv changes the number of domains while holding the training-task count fixed, and Beyond Simply Environment Scaling changes environment types under a fixed training-sample budget (Tu et al. 2026; Zhu et al. 2026). We study a different allocation: within two task domains and a fixed model, how does exchanging repeated exposure for additional task instances affect learning under a limited training budget? The results characterize this allocation for the tested model, task domains, and budgets.

To specify this allocation, we define task quantity as the number of distinct task instances available for training. A prompt presentation means one scheduled sampling event for one task; each scheduled iteration presents one Structured Query Language (SQL) and one arithmetic prompt and samples four responses for each. A scheduled iteration produces an optimizer update only when at least one prompt has responses with different rewards. We distinguish quantity from domain composition, semantic diversity, and reference-solution complexity. The present experiment varies quantity while retaining a 50/50 SQL/arithmetic mixture. SQL table identifiers and number-masked arithmetic questions supply conservative input-grouping rules: at most one representative is retained per SQL table and per exact numeric-masked arithmetic string. The selection groups keys and representatives after the prompt-plus-length filter; the source manifests and serialized schedules define the sampled population, and no model score enters selection. These grouping rules remove repeated input groups. Task count measures distinct instances; it does not measure independent reasoning skills.

Comparing these task allocations requires three measurement choices. First, equal response counts do not imply equal token counts: a response may terminate after a short query or continue to the length limit. Second, endpoint choice can change the observed outcome even along the same training trajectory. Third, development evaluation can guide development and expose failure modes without supplying an independent test of the resulting research hypothesis. We use these distinctions to define the endpoints and interpret the results.

We propose an experimental design with nested task sets, fixed training schedules, and two endpoints specified before training. Adapter identities link each evaluation to its trained parameters. Every run begins with the same base checkpoint and a fresh low-rank adaptation (LoRA) adapter. Each generated completion retains its token sequence, reward, and policy log probabilities. Evaluation preserves task order and compares the base model with itself before and after the adapted conditions. The supplementary overview in Appendix A summarizes the study.

The quantity study contains nine training runs with paired development evaluations. We report all three seeds at every rollout endpoint, preserve the unavailable response-token endpoint for $N = 256$, and analyze in-domain and transfer outcomes separately. The results show a modest non-monotonic development pattern and that rollout equality leaves substantial resource differences. The final evaluation on tasks reserved from project tuning further shows that none of the six supplementary methods under the

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.

training pipeline time budget exceeds the unadapted Qwen3.5-4B checkpoint in mean macro accuracy. These results place task quantity, endpoint definition, and realized cost at the center of the allocation question; the reserved evaluation and resource ledger then determine whether a development gain transfers.

2 Related work

2.1 Learning from execution feedback

Learning from execution feedback spans code, tool-use, and interactive-agent settings. Seq2SQL uses query execution to provide rewards for semantic parsing, and InterCode treats code execution as an interactive feedback channel (Zhong et al. 2017; Yang et al. 2023). AgentGym and AgentGym-RL study exploration and multi-turn decision-making across diverse environments (Xi et al. 2024, 2025). In software engineering, SWE-Gym supplies repository tasks for training agents, SWE-smith synthesizes additional training tasks by introducing test-breaking changes into codebases, and ToRL studies reinforcement learning for computational tool use (Pan et al. 2024; Yang et al. 2025; Li et al. 2025). These works differ in trajectory length, action space, and repository context. We study task allocation in single-response SQL and arithmetic training.

Training objectives also differ across these settings. Direct preference optimization (DPO), REINFORCE with a leave-one-out baseline (RLOO), proximal policy optimization (PPO), and the group-relative objective proposed in DeepSeekMath use distinct update rules (Rafailov et al. 2023; Ahmadian et al. 2024; Schulman et al. 2017; Shao et al. 2024). STaR investigates iterative self-training with generated reasoning (Zelikman et al. 2022). Our comparison includes supervised fine-tuning, rejection sampling fine-tuning, DPO, RLOO, PPO, and group-normalized policy gradient. The comparison keeps supervision, sampling, and training time visible instead of treating algorithm names as interchangeable controls.

2.2 Task quantity and resource accounting

Language-model scaling studies distinguish parameter count, training data, repetition, and computation (Kaplan et al. 2020; Hoffmann et al. 2022). In data-constrained pretraining, Muennighoff et al. vary repetition and compute to study how repeated data contributes to learning (Muennighoff et al. 2023). Those studies provide an allocation perspective, while our experiments examine post-training with execution feedback and a small set of verified tasks.

Data content is another allocation axis. LIMA studies alignment using a small curated supervised set, whereas Textbooks Are All You Need emphasizes selected and synthetic educational material for code modeling (Zhou et al. 2023; Gunasekar et al. 2023). ScaleEnv varies domains while holding the training-task count fixed, and Beyond Simply Environment Scaling examines environment types under a fixed training-sample budget (Tu et al. 2026; Zhu et al. 2026). We hold the SQL/arithmetic mixture fixed when varying task quantity, then measure task composition and reference complexity as separate factors.

2.3 Evaluation coverage

HumanEval and MBPP evaluate short-program synthesis through functional behavior (Chen et al. 2021; Austin et al. 2021). EvalPlus shows that stronger tests can reveal missed errors and change model rankings (Liu et al. 2023). BigCodeBench extends evaluation toward diverse library calls and complex instructions, whereas SWE-bench uses repository issues requiring codebase changes (Zhuo et al. 2024; Jimenez et al. 2023). LiveCodeBench collects time-indexed contest problems to address contamination and broaden code evaluation (Jain et al. 2024). These benchmarks motivate our use of reserved tests, transfer diagnostics, and task-level outcome records.

Sampling coverage also affects comparisons. Yue et al. examine how comparisons between base and reinforcement-learning-trained models change across pass@k budgets (Yue et al. 2025). Our evaluation records greedy outcomes on fixed development and reserved sets, while Appendix B, Tables 10 and 11 retain condition-level results and seed records. This design makes the measured task allocation and endpoint costs inspectable alongside the reported accuracy.

3 Experimental design

The study varies task count under fixed response budgets and evaluates the resulting policies on development tasks and a reserved final test.

3.1 Model and update objective

We use low-rank adaptation (LoRA) to train adapters on selected projections of the text model with a group-normalized on-policy policy gradient (PG) objective. Figure 1 separates the model architecture from the feedback-driven parameter update. The Transformer uses a sequence-modeling architecture based entirely on attention (Vaswani et al. 2017); the Qwen3.5-4B text backbone uses a hybrid rather than an all-attention stack. Its official configuration contains eight groups of three Gated DeltaNet blocks followed by one Gated Attention block, each with a feed-forward network (Qwen Team 2026). Our inputs are text only. The diagram omits normalization, residual connections, and the unused vision path to expose the components relevant to adaptation.

We adapt selected projections with LoRA while keeping base parameters frozen; the update objective and response normalization are defined below.

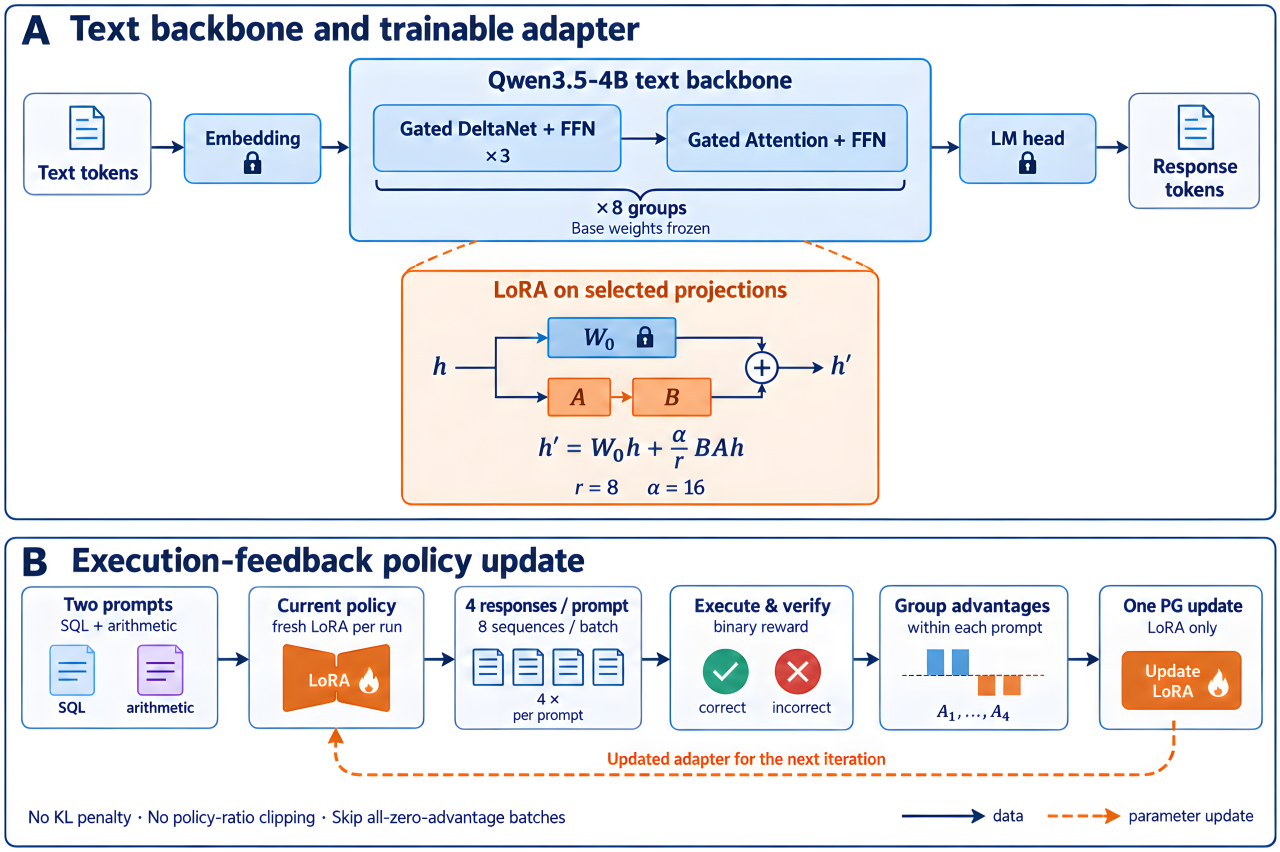

Figure 1: Model and execution-feedback update. Panel A follows text tokens through the frozen embedding, the Qwen3.5-4B backbone (eight repeated groups of Gated DeltaNet–FFN blocks followed by Gated Attention–FFN), and the frozen language-model head to response tokens. The orange inset expands a selected projection: W_0 is the frozen weight matrix, A and B are trainable LoRA factors, $r = 8$ is the adapter rank, and $\alpha = 16$ is the scaling parameter. Panel B shows two prompts (SQL and arithmetic), four sampled responses per prompt, binary execution verification, within-prompt group advantages, and one LoRA-only policy-gradient update. The dashed return path carries the updated adapter to the next iteration; the footer records the absence of a KL penalty, policy-ratio clipping, and updates for all-zero-advantage batches.

We train Qwen3.5-4B with the LoRA adapters described above (Hu et al. 2021), using rank 8, scaling parameter 16, and zero adapter dropout. The configured target modules are the query, key, value, output, and gated feed-forward projection names supported by the stated settings. Base parameters remain frozen. AdamW uses a constant learning rate of 2×10^{-5}, zero weight decay, and gradient-norm clipping at 1. Each run starts from a fresh adapter. We ran the experiments on an NVIDIA Tesla PG503-216 graphics processing unit (GPU) with 32 GiB of memory.

At training iteration s, the schedule presents two prompts, one per training domain. For each prompt x, the current policy generates $K = 4$ responses. Let $r_i \in \{0, 1\}$ denote the fixed verifier's reward. For each prompt, we compute

$$\bar{r} = \frac{1}{K}\sum_{i=1}^{K} r_i, \qquad \sigma_r = \sqrt{\frac{1}{K}\sum_{i=1}^{K}(r_i - \bar{r})^2}, \qquad A_i = \frac{r_i - \bar{r}}{\sigma_r + 10^{-4}}.$$

Let T_i denote the number of sampled response tokens in sequence i, excluding padding, and let θ denote the trainable adapter parameters. For the batch of $B = 8$ response sequences, the minimized loss is

$$\mathcal{L}(\theta) = -\frac{1}{B}\sum_{i=1}^{B}\frac{A_i}{T_i}\sum_{t=1}^{T_i}\log \pi_\theta(y_{i,t} \mid x_i, y_{i,<t}).$$

Advantages are fixed during the update. Response tokens are averaged within a sequence and sequences across the complete batch, so long responses do not receive more sequence-level weight than short responses, although length still affects generation and forward/backward cost. Padding is masked from the token sum and end-of-sequence (EOS) contributes only when it was actually sampled. The optimizer makes one update on the newly sampled batch, with no policy-ratio clipping and no Kullback–Leibler (KL) divergence penalty. Homogeneous reward groups have zero advantage; an entirely inactive batch skips the optimizer update. We refer to this update as group-normalized on-policy PG. Group Relative Policy Optimization (GRPO), proposed in DeepSeekMath (Shao et al. 2024), uses a clipped policy surrogate and a reference-policy KL penalty. Our update uses the loss above, with four

responses per prompt and within-prompt reward normalization. It omits the policy-ratio surrogate and reference-policy penalty. Gradient-norm clipping limits the gradient magnitude independently of the policy objective.

To sample the responses used in this loss, generation uses temperature 1, top-p 1, disabled thinking, a 1,024-token response cap, and a 4,096-token context cap. The sampled response IDs, including EOS only when actually sampled, are preserved. We use the vLLM inference engine for generation (Kwon et al. 2023), with an explicit LoRA request carrying a unique adapter path and identity at each iteration. The training pass recomputes the sampled-token log probabilities. Each request uses the recorded adapter identity. We compare the log probabilities from generation with those recomputed during training on sampled, unmasked tokens: the run stops if mean absolute disagreement exceeds 0.1 or any sequence-mean disagreement exceeds 0.5. The run also stops on invalid artifacts, non-finite numerical values, or inconsistent token and task counts.

3.2 Task construction and selection

With the model and update fixed, we construct the training population from Structured Query Language (SQL) tasks derived from WikiSQL (Zhong et al. 2017) and arithmetic tasks derived from Grade School Math 8K (GSM8K), answered by Python programs (Cobbe et al. 2021). We use GSM as the domain label for these arithmetic tasks. SQL reference queries use a column-index execution protocol; arithmetic references are converted from explicit source calculations and must yield the source answer. These wrappers are part of our study rather than the original dataset protocols. Candidate construction excludes the original 64 development identities and 200 reserved held-out identities. Additional input-based exclusions remove reserved SQL tables and matching number-masked arithmetic questions. Reserved reference solutions are not executed during training-pool preparation. The wrappers validate the table schema or source calculation and retain the source final answer as the target.

The eligible inventory contains 2,541 SQL instances across 755 tables and 6,598 arithmetic instances across 6,597 number-masked question strings. The quantity collection contains 1,024 tasks, half from each domain. A fixed input-grouping procedure selects representatives without consulting model scores. Within the collection, each SQL table and each arithmetic grouping string contributes at most one task. The collections at N = 64, 256, and 1,024 are nested and share this single draw; stored per-domain shuffle cycles determine task order, so the smaller sets are prefixes of the same registered ordering. Three training seeds change training randomness rather than constructing three independent datasets. Seed-level standard deviation (SD) therefore excludes task-draw variability and subset-selection uncertainty.

All tasks in the collection fit the model context budget of 4,096 tokens; the largest observed prompt is 297 tokens, so no additional per-task prompt cap is active. No task is removed on the basis of response success or prompt length during construction of these nested sets. The population requires available reference solutions and differs from pools screened by model success; comparisons across these selection rules would confound population and training effects.

3.3 Training schedules and endpoints

We apply the same schedule to each nested task set. Each run executes 512 scheduled iterations, yielding 1,024 prompt presentations and 4,096 responses. Scheduled iterations and actual optimizer updates are distinct: inactive batches remain in the schedule but do not advance the optimizer. At N = 64, 256, and 1,024, each task is presented exactly sixteen, four, and one times, respectively. Domain balance is enforced within every iteration. Collection size changes alongside per-task repetition under this fixed total allocation; the experiment does not separately identify these two mechanisms.

The rollout endpoint is the model state after scheduled iteration 512. The response-token endpoint is the model state after the first complete scheduled iteration for which cumulative generated response tokens reach 500,000; its actual update count is trajectory-dependent and is reported in the training ledgers. We assess the token threshold after the entire scheduled iteration, so overshoot is less than 8,192 tokens. If the threshold is not reached by iteration 512, that endpoint is unavailable. Training is never extended in response to the score or the missing endpoint.

We report both endpoints and the number of distinct tasks observed at each endpoint. Generated response tokens exclude prompt processing, reference-model computation, backward passes, and other training work. They measure the response-generation budget; total computation also includes these other operations.

3.4 Evaluation

Evaluation combines 48 development tasks, sixteen each from SQL, Python arithmetic, and Mostly Basic Python Problems (MBPP) (Austin et al. 2021), with fourteen canonical shell fixtures. These subsets measure performance on the selected development tasks. The 48-task subset and the shell subset each retain their original task order and are evaluated in batches of eight prompts. Task identifiers and selection records were fixed before final scoring. Each condition uses greedy decoding, seed 42, disabled thinking, and the same response and context caps as training. Available adapters are identified explicitly before scoring.

The condition order is base, available fixed endpoints, and base-repeat. A correctness discrepancy between the two base evaluations stops the queue. Token disagreement is separately recorded. We also compare base outputs across runs before aggregating their results. We score both successful and failed task outputs. The predefined extractor and verifier score extraction failures, timeouts, truncations, and incorrect final artifacts as failures.

We define SQL/GSM macro accuracy as the unweighted mean of SQL and arithmetic accuracy. MBPP and shell are reported separately as transfer diagnostics. The raw number correct out of 62 weights tasks equally, whereas the four-domain macro weights

the domains equally. Development tasks were used during development and analysis. Shell scores measure performance on the original fixtures.

We evaluated all 47 fixed original endpoints and 18 separately registered budget endpoints on 200 official GSM8K test tasks and 200 sanitized MBPP test tasks. These tasks were reserved from this project's tuning, although their presence in model pretraining cannot be excluded. Final-test macro accuracy is the unweighted mean of GSM8K and MBPP accuracy; it covers different domains from the development SQL/GSM macro. All endpoints used the same frozen greedy decoding configuration: seed 42, an output cap of 1,024 tokens, eager execution, batch size eight, and the validated split-one LoRA kernel configuration. Across the original queue and two supplementary shards, all six baseline or adapter repeat controls agreed in tokens and scored correctness. The 47 original endpoints, 18 supplementary budget endpoints, and nine baseline or repeat conditions yield 74 generated conditions and 29,600 outputs. The final tasks were excluded from tuning, acquisition, and development selection.

4 Quantity results

4.1 Fixed endpoints

We report development performance for the nine runs using the protocol in Section 3.3. Structured Query Language (SQL) and GSM, our label for arithmetic tasks derived from Grade School Math 8K, form the training domains. SQL/GSM macro accuracy averages their accuracies. Standard deviation (SD) summarizes variation across training seeds. The aggregate contains seventeen available endpoint observations and one explicitly unavailable endpoint. Base correctness and generated tokens agree across all nine runs and their repeated evaluations. The base answers 41 of 62 tasks correctly: 11/16 SQL, 15/16 arithmetic, 9/16 Mostly Basic Python Problems (MBPP), and 6/14 shell. The condition-level endpoint values are retained in Appendix B, Table 4; Figure 2 presents the corresponding development accuracy and paired endpoint changes.

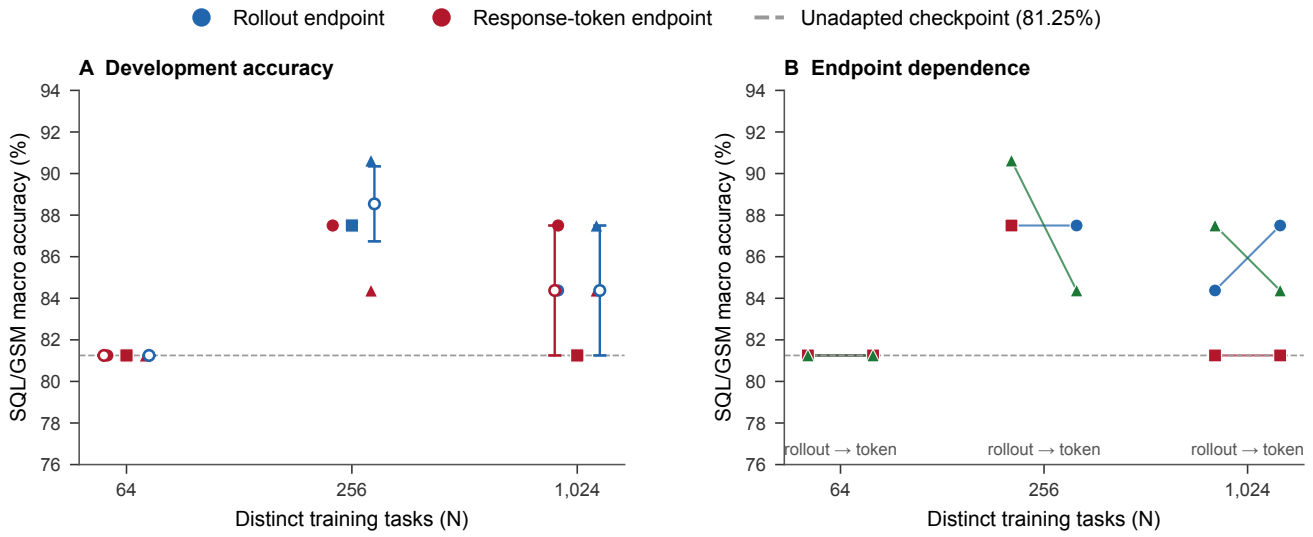

Figure 2: Task-set allocation and endpoint dependence on the development set. Panel A shows individual seeds and mean ± sample SD for SQL/GSM macro accuracy at the rollout and response-token endpoints across nested task populations. Panel B shows paired changes between endpoints for each available seed; the $N = 256$ run with seed 1 has no response-token endpoint. The dashed line marks the unadapted checkpoint; error bars show sample SD across training seeds.

At 4,096 rollouts, the mean SQL/GSM macro accuracies are 81.25%, 88.54%, and 84.38% for $N = 64$, $N = 256$, and $N = 1{,}024$. Their sample standard deviations across training seeds are 0.00, 1.80, and 3.13 percentage points. The prespecified differences between the $N = 256$ and $N = 64$ conditions are 6.25, 6.25, and 9.38 points; their mean is 7.29 points with a seed standard deviation of 1.80. The corresponding differences between the $N = 1{,}024$ and $N = 64$ conditions are 3.13, 0.00, and 6.25 points, averaging 3.13 with a standard deviation of 3.13. Increasing task-set size from $N = 256$ to $N = 1{,}024$ therefore does not continue the improvement in these observed runs. For allocation design, $N = 256$ is the most informative starting point in this grid: it improves development accuracy relative to $N = 64$ without carrying the nominal increase to $N = 1{,}024$. A larger collection should therefore be justified by a measured coverage benefit rather than by task count alone.

At the response-token endpoint, the differences between the $N = 1{,}024$ and $N = 64$ conditions are 6.25, 0.00, and 3.13 points, again averaging 3.13 with a seed standard deviation of 3.13. The $N = 256$ run with seed 1 generated only 408,685 response tokens during the registered 512 scheduled iterations, so it did not reach the 500,000-token threshold and cannot contribute to a complete three-seed token contrast. We did not extend training beyond the registered schedule; the two $N = 256$ response-token results therefore summarize two seeds. This boundary is itself informative: a fixed token target can become unavailable when response lengths and update activity interact, so endpoint feasibility should be reported before comparing allocations.

4.2 Transfer and endpoint dependence

The in-domain averages in Section 4.1 do not capture all changes on the 62 development tasks. For the $N = 1{,}024$ run with seed 0, the token endpoint answers 44 tasks correctly, while the later rollout endpoint answers 40. MBPP changes from 10/16 to 7/16, arithmetic from 15/16 to 14/16, and SQL remains 13/16; shell remains 6/14. The later endpoint thus performs worse on this transfer evaluation.

Endpoint differences also occur alongside differences in the number of optimizer updates. Across the nine quantity runs, active optimizer updates are 164/197/176 for $N = 64$, 192/204/187 for $N = 256$, and 217/182/166 for $N = 1{,}024$ (seeds 0, 1, and 2); the remaining scheduled iterations are skipped because both prompts have homogeneous rewards. These counts describe how often the sampled rewards produce an active update; they do not by themselves explain the observed accuracy differences.

Across seeds, transfer outcomes can also differ when in-domain accuracy is unchanged. For the $N = 64$ runs, the rollout endpoint leaves SQL and arithmetic unchanged from the base in all three seeds. MBPP is 10/16 for seed 0 and 8/16 for seeds 1 and 2, compared with the base's 9/16. Shell is 7/14, 6/14, and 6/14. Consequently, the $N = 64$ raw score is 43, 40, and 40, despite identical in-domain macro accuracy. By contrast, the $N = 256$ run with seed 2 reaches 48/62, with 13/16 SQL, 16/16 arithmetic, 12/16 MBPP, and 7/14 shell. This variation shows the sensitivity of transfer outcomes to the training seed. It also gives a practical evaluation rule: in-domain macro accuracy alone is not sufficient for selecting an allocation. Transfer tasks should remain in the reporting set because they expose endpoint choices and seed-specific regressions that the SQL/GSM average hides.

Individual task outcomes show regressions as well as gains. In one $N = 64$ run, the token endpoint improves one Python task and one shell task but regresses on another Python task after a truncated response. In one $N = 256$ run, a shell task performs the requested rename and then moves the target a second time to an incorrect final location. The verifier scores the final artifact, so an intermediate correct action can still lead to a failed task.

Table 1 gives representative transfer records that make these endpoint and seed effects explicit.

Table 1: Representative transfer records on the 62-task development set.

Condition	Correct / 62	Observed transfer detail
$N = 64$ rollout endpoint, seeds 0, 1, and 2	43 / 40 / 40	same in-domain macro
$N = 256$ rollout endpoint, seed 2	48	12/16 MBPP; 7/14 shell
$N = 1{,}024$ response-token endpoint, seed 0	44	10/16 MBPP; 6/14 shell

4.3 Rollout and resource differences

Figure 3 shows the generated-token and wall-clock costs of the fixed-rollout runs.

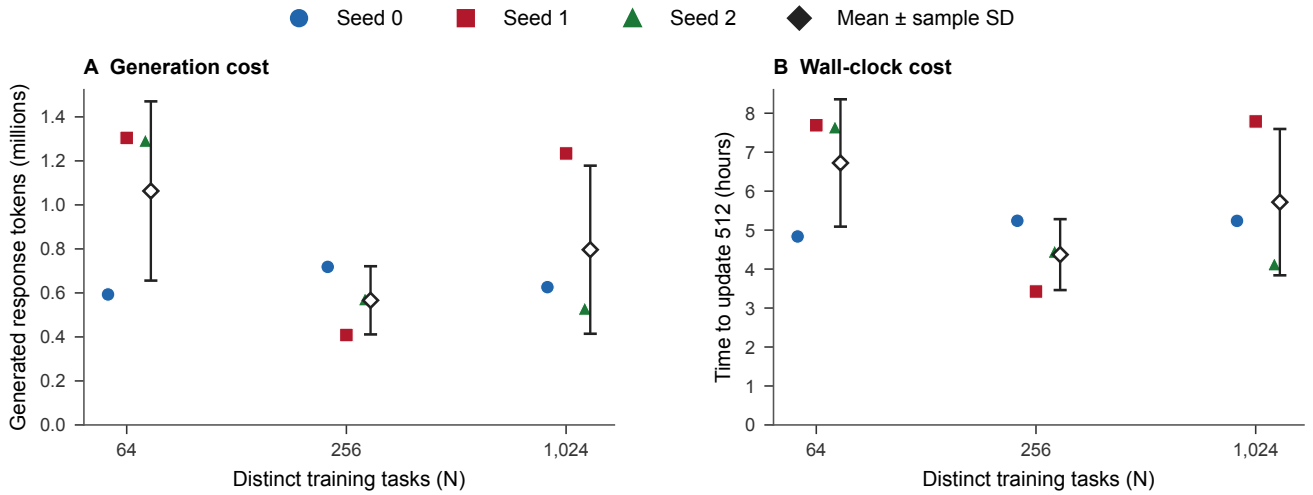

Figure 3: Resource variation at the fixed-rollout endpoint. Panel A shows generated response tokens and Panel B shows elapsed time to update 512 for all three seeds at each task population. Diamonds and vertical bars show mean ± sample SD; the quantities exclude subsequent evaluation and do not represent total FLOPs.

The fixed-rollout runs generate between 408,685 and 1,304,377 response tokens, a difference of approximately 3.19 times. Time to the last update ranges from approximately 3.42 to 7.79 hours. The complete serial quantity queue, including its evaluations and orchestration, lasts approximately 52.86 hours. Equal rollout counts therefore fail to equalize even these directly observed cost components. A rollout budget is thus a reproducible sampling rule, not a compute budget: studies comparing task coverage should report realized response length and wall time alongside rollout counts.

Actual coverage also changes at the token endpoint. The three $N = 1{,}024$ runs have observed 900, 434, and 964 distinct tasks when they first reach the threshold, rather than all 1,024. The token endpoint therefore compares trajectories with different observed coverage as well as different available collection sizes. Token matching should consequently be paired with an exposure ledger; otherwise a nominal 500,000-token endpoint can conceal how many distinct tasks actually received feedback.

5 Method comparison

The resource differences in Section 4.3 motivate two comparisons of group-normalized on-policy policy gradient (PG) with five training approaches: one using a shared task population and method-specific training schedules, and another using a common training pipeline time budget.

5.1 Public methods

We compare supervised fine-tuning (SFT), direct preference optimization (DPO), rejection sampling fine-tuning (RFT), REINFORCE with a leave-one-out baseline (RLOO), and proximal policy optimization (PPO) with PG. RFT is a single-round adaptation. We use the $N = 256$ population for this comparison after the quantity study.

A fixed acquisition stage generates sixteen base-model responses per task. The first qualifying correct response supplies an RFT target; the first qualifying correct and incorrect responses supply a DPO pair. Successful targets cover 237 tasks and preference pairs cover 170. SFT uses verified source references. Acquisition cost is reported separately because support and supervision differ across methods.

The shared-population method details are retained in Appendix B, Table 5; Table 2 gives the development comparison in compact form. The offline schedule balances 1,024 presentations over each method's eligible support before the official trainer shuffles the examples. RLOO and PPO retain all 256 tasks. Common adapter rank, learning rate, model identity, context cap, and evaluation conditions align the shared controls. Supervision, sampling structure, and compute remain explicit parts of the method comparison. RLOO uses a Kullback–Leibler (KL) divergence coefficient of 0.05 and an unnormalized leave-one-out advantage. PPO uses a separate low-rank critic with a trainable value head, generalized advantage estimation, policy and value clipping, and two optimization epochs per rollout batch. These runs compare methods on a shared task population. Section 5.2 examines a separate constraint on pipeline duration.

The development comparison includes fifteen runs: five methods with three seeds each. The PPO seed 1 result comes from a fresh full run after the original process was killed at update 476; the failed run consumed 55,910.21 seconds of worker time. Repeated and cross-run base correctness and tokens agree. The base answers 41 of 62 tasks correctly.

Table 2: Development comparison on the shared $N = 256$ population. "Correct / 62" is the total correct on the 62-task development set; "SQL/GSM macro" is the mean of the SQL and GSM accuracies. Values are mean ± sample SD across three training seeds.

Method	Correct / 62	SQL/GSM macro
PG	45.67 ± 2.08	88.54% ± 1.80%
SFT	35.67 ± 0.58	62.50% ± 5.41%
DPO	36.67 ± 5.13	62.50% ± 13.62%
RFT	41.00 ± 1.73	81.25% ± 3.12%
RLOO	42.67 ± 1.15	84.38% ± 0.00%
PPO	45.33 ± 2.08	87.50% ± 0.00%

The fixed PG rollout endpoint at $N = 256$ has a mean of 45.67 correct tasks, compared with 45.33 for PPO, the highest mean among the five public approaches. Its SQL/GSM macro is 88.54%, compared with 87.50% for PPO. PG and PPO have similar development means, and the per-seed PG-minus-PPO correct-count differences are -2, -2 and 5. The shared-population comparison therefore places the two on a similar observed range while showing that supervision, eligibility, sampling, and optimization remain part of the result. Acquisition support and supervision differ across methods; Section 5.2 reports their costs.

The method comparison varies the learning rule on a fixed task population. A separate factor experiment instead varies task composition and reference complexity while retaining group-normalized PG.

The factor experiment contains five conditions with all three training seeds: mixed SQL/arithmetic, SQL-only, arithmetic-only, and low/high reference-complexity strata. Every run uses 256 tasks, 512 scheduled iterations and 4,096 responses, with four presentations per task. All fifteen runs use verified source data, and repeated baseline outputs agree. The complexity strata use a proxy derived from reference solutions; they do not measure interaction horizon.

The task-composition values are retained in Appendix B, Table 7; Figure 4 visualizes the composition, reference, and limited-budget comparisons. SQL-only exceeds mixed training in SQL/GSM macro accuracy in all three seeds, by 3.125, 6.25 and 3.125 percentage points. Arithmetic-only is lower than mixed in all three seeds. In this grid, the SQL-heavy collection is the strongest development choice and the arithmetic-only collection is the weakest; the result supports matching task composition to the target domain rather than assuming that a broader mixture is always better. The SQL-only runs also average 163,251 generated response tokens, compared with 750,812 for mixed and 1,277,390 for arithmetic-only. Composition changes therefore alter both learning signal and realized generation cost, so a domain-mix result should be reported with its resource profile. Aggregate transfer outcomes remain seed-dependent.

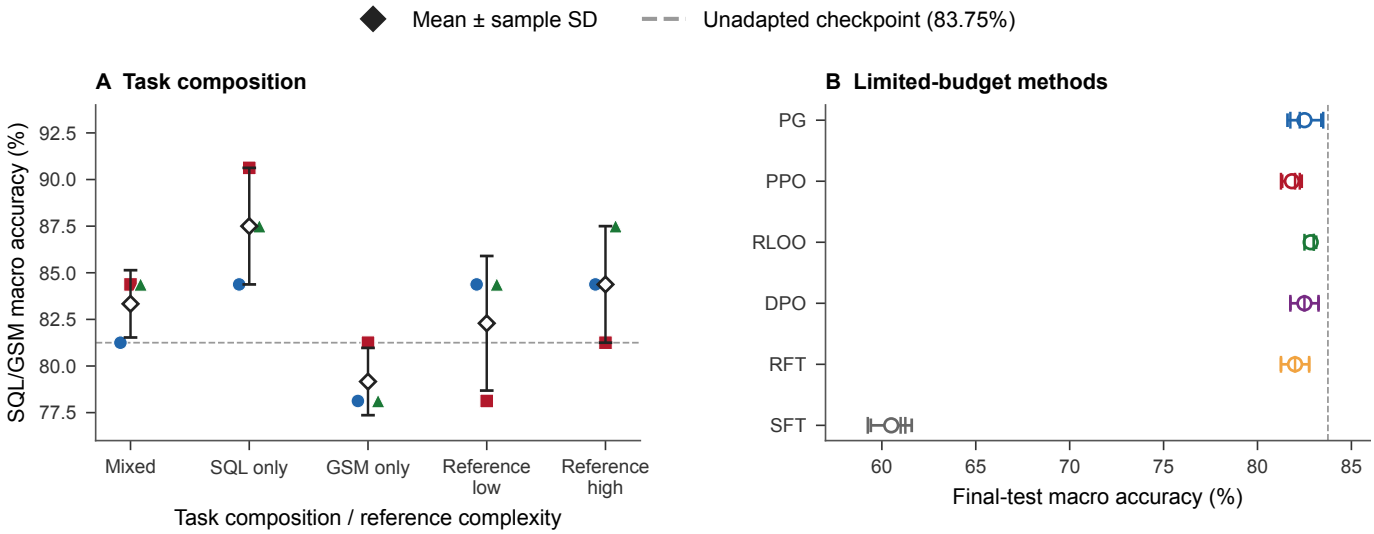

Figure 4: Task composition and limited-budget method comparison. Panel A shows SQL/GSM macro accuracy for five task-composition and reference-complexity conditions; Panel B shows final-test macro accuracy for six methods under the registered 7,200-second pipeline budget. Individual seed marks and mean ± sample SD are shown; the dashed line denotes the unadapted Qwen3.5-4B checkpoint.

The high-minus-low reference-complexity contrast changes the aggregate correct count by −1, +6 and 0 across seeds, while the final-test mean favors high complexity by one percentage point. The factor result supports keeping reference complexity in the allocation design: it can improve the observed outcome without changing task count, although the effect should be checked alongside task identity and realized cost. Different physical GPUs and concurrent host workloads also limit interpretation of recorded elapsed times.

5.2 Final evaluation

To compare methods under a common time constraint, the supplementary experiment fixes a 7,200-second training pipeline time budget on the same physical graphics processing unit (GPU). We evaluate these endpoints on the reserved GSM8K and MBPP final test described in Section 3.4. The comparison measures performance after a fixed allocation of training pipeline wall time. The number of completed optimizer updates can therefore differ across methods. Each run stops at the first complete update crossing that budget, with a prespecified maximum of 7,920 seconds. Setup and final export are excluded; acquisition is reported separately. All six methods and three seeds met the stopping rule and produced valid exported adapters. The comparison holds pipeline duration approximately constant while supervision, update counts, and floating-point operation counts can differ.

The final-test method values are retained in Appendix B, Table 8; panel B of Figure 4 provides the visual comparison.

Every supplementary method has lower mean macro accuracy than the unadapted Qwen3.5-4B checkpoint. PG reaches 82.50%, compared with 83.75% for the base and 82.83% for RLOO. The registered time budget therefore favors retaining the unadapted checkpoint as the primary reference: none of the tested adaptation rules converts this fixed pipeline allocation into a mean final-test gain. SFT has the largest aggregate regression, at 60.50%, which shows that supervision source and acquisition quality must be treated as first-order factors in time-budget comparisons. The three-seed results support reporting method performance together with completed updates and acquisition cost.

Training time and update counts are retained in Appendix B, Table 9 and provide the cost context for the method-level accuracy comparison. The shared acquisition requires 4,096 attempts, 780,509 response tokens, and 2,980.33 seconds of elapsed time. RFT and DPO each incur this full acquisition cost when treated as a standalone procedure; successful examples cover 237 tasks and preference pairs cover 170. SFT uses source-reference supervision on 256 tasks, an additional information advantage. A failed RFT seed-zero attempt consumes another 7,200.75 pipeline seconds and produces no exported adapter because of a post-training accounting assertion; this cost is retained separately from its accepted replacement. Full PPO profiling exceeded 128 GiB and was not repeated at a larger limit. The profiling failure limits the available resource measurements. Methods complete different numbers of updates within the measured time budget. Floating-point operation counts remain unmatched.

5.3 Original endpoints

The original schedules were also evaluated on the reserved final tasks, without checkpoint selection. Across the rollout endpoints, macro means are 83.25%, 83.42%, and 83.17% for $N = 64$, $N = 256$, and $N = 1{,}024$. The two available response-token seeds for $N = 256$ average 83.50%, while response-token training for $N = 1{,}024$ reaches 83.75%. The task-count differences are small on the final test, so the development advantage at $N = 256$ does not transfer as a consistent final-test improvement. The high-reference condition reaches 83.92% versus 82.92% for low reference, and is higher for two of three seeds. This result makes reference complexity a useful candidate factor for future allocation studies.

Table 3 summarizes these final-test endpoint means.

Table 3: Reserved final-test macro accuracy for the original quantity endpoints. Each cell is the macro accuracy for the indicated task count and endpoint; the dash marks the unavailable response-token condition for $N = 64$.

Endpoint	$N = 64$	$N = 256$	$N = 1{,}024$
Rollout	83.25%	83.42%	83.17%
Response tokens	—	83.50%	83.75%

Appendix B, Table 11 retains all seed-level contrasts, including the unavailable response-token endpoint for the $N = 256$ run with seed 1. These records show why each allocation should be carried through a reserved evaluation: development accuracy, endpoint definition, and realized exposure can point in different directions.

6 Conclusion

We studied how task coverage and repeated exposure trade off in learning from execution feedback. We built nested SQL/arithmetic task populations with 64, 256, and 1,024 tasks; trained Qwen3.5-4B with group-normalized on-policy policy gradient; compared rollout-count and response-token endpoints; measured transfer, generated-token, update, wall-clock, and acquisition costs; varied task composition and reference complexity; and evaluated six training methods on reserved final tasks under a registered 7,200-second pipeline budget. We retained all seeds, unavailable endpoints, failed runs, and resource shortfalls.

Taken together, the experiments support four general conclusions for limited-budget post-training with execution feedback. First, task-set scaling is an allocation problem rather than a count-only property: changing N simultaneously changes the number of distinct tasks receiving feedback, the number of times each task is revisited, and the resources spent on responses and updates. In our nested grid, $N = 256$ produced the highest development mean, but $N = 1{,}024$ did not continue the gain and the original quantity endpoints were close on the reserved test. A larger task collection therefore changes the allocation of feedback; its value depends on whether the added coverage produces useful exposure within the available budget.

Second, the endpoint is part of the scientific claim. Equal rollout counts left token and wall-clock costs unequal, and equal token targets left observed task coverage unequal. Third, task composition and reference complexity are independent allocation axes: both changed outcomes at fixed task count, so nominal N does not specify the training distribution. Fourth, an adaptation method should be judged by cost-adjusted transfer against the unadapted checkpoint. Under the limited pipeline-time comparison, every tested method remained below that checkpoint in mean final-test macro accuracy. Acquisition cost and the unadapted checkpoint are therefore part of the result, alongside accuracy, whenever the budget is limited.

The practical contribution is a way to turn an ambiguous scaling claim into an allocation diagnosis. A task-scaling experiment can use a nested task population, a declared endpoint and budget, realized response tokens and active updates, observed task coverage, wall-clock and acquisition costs, a reserved final evaluation, and an unadapted checkpoint to separate three possible sources of an observed gain: new task coverage, repeated feedback on existing tasks, and additional training expenditure. The distinction gives a concrete next-step rule. After locating the coverage level at which reserved-task performance stops improving, the remaining budget can be assigned to repeated feedback, new task instances, or evaluation according to the desired exposure; if adaptation does not exceed the unadapted checkpoint under the same time budget, the base checkpoint remains the more efficient choice. By separating coverage, repetition, and resource expenditure, this study gives the field an empirical basis for interpreting task-set scaling as a training-allocation claim. It clarifies which apparent gains are likely to transfer when the endpoint or budget changes, identifies when a larger task collection changes exposure without improving transfer, and provides a common basis for choosing between new tasks, repeated feedback, and further adaptation under a fixed resource budget.

Reproducibility Statement

The paper provides the fixed model, task, training, evaluation, and result records needed to reproduce and audit the reported comparisons. Section 3.1 specifies the Qwen3.5-4B backbone, LoRA configuration, NVIDIA Tesla PG503-216 graphics processing unit (GPU) with 32 GiB of memory, and policy-gradient update. Section 3.2 specifies the task sources, filtering, nested task sets, and selection procedure. Section 3.3 specifies the training schedules, endpoint thresholds, and stopping rules, and Section 3.4 specifies decoding, base and repeat controls, and the reserved final tests. Sections 4 and 5 report the quantity, task-composition, reference-complexity, and method-comparison results. Appendix B, Tables 4–11 provide the condition-level results, seed-level records, endpoint costs, and paired contrasts, including unavailable endpoints and failed runs.

AI Use Disclosure

We used generative artificial intelligence (AI) tools to polish the manuscript. For Figure 1 in the main text and Figure 5 in Appendix A, the authors designed the workflows and specified the scientific content. AI tools assisted with drawing the figures, after which the authors manually revised the illustrations and checked them against the study design and methods. We take responsibility for the final content of this work.

References

Ahmadian, Arash, Chris Cremer, Matthias Gallé, et al. 2024. "Back to Basics: Revisiting REINFORCE Style Optimization for Learning from Human Feedback in LLMs." *arXiv Preprint arXiv:2402.14740*. https://arxiv.org/abs/2402.14740.

Austin, Jacob, Augustus Odena, Maxwell Nye, et al. 2021. "Program Synthesis with Large Language Models." *arXiv Preprint arXiv:2108.07732*. https://arxiv.org/abs/2108.07732.

Chen, Mark, Jerry Tworek, Heewoo Jun, et al. 2021. "Evaluating Large Language Models Trained on Code." *arXiv Preprint arXiv:2107.03374*.

Cobbe, Karl, Vineet Kosaraju, Mohammad Bavarian, et al. 2021. "Training Verifiers to Solve Math Word Problems." *arXiv Preprint arXiv:2110.14168*. https://arxiv.org/abs/2110.14168.

Gunasekar, Suriya, Yi Zhang, Jyoti Aneja, et al. 2023. "Textbooks Are All You Need." *arXiv Preprint arXiv:2306.11644*.

Hoffmann, Jordan, Sebastian Borgeaud, Arthur Mensch, et al. 2022. "Training Compute-Optimal Large Language Models." *arXiv Preprint arXiv:2203.15556*.

Hu, Edward J., Yelong Shen, Phillip Wallis, et al. 2021. "LoRA: Low-Rank Adaptation of Large Language Models." *arXiv Preprint arXiv:2106.09685*. https://arxiv.org/abs/2106.09685.

Jain, Naman, King Han, Alex Gu, et al. 2024. "LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code." *arXiv Preprint arXiv:2403.07974*.

Jimenez, Carlos E., John Yang, Alexander Wettig, et al. 2023. "SWE-Bench: Can Language Models Resolve Real-World GitHub Issues?" *arXiv Preprint arXiv:2310.06770*.

Kaplan, Jared, Sam McCandlish, Tom Henighan, et al. 2020. "Scaling Laws for Neural Language Models." *arXiv Preprint arXiv:2001.08361*.

Kwon, Woosuk, Zhuohan Li, Siyuan Zhuang, et al. 2023. "Efficient Memory Management for Large Language Model Serving with PagedAttention." *arXiv Preprint arXiv:2309.06180*. https://arxiv.org/abs/2309.06180.

Li, Xuefeng, Haoyang Zou, and Pengfei Liu. 2025. "ToRL: Scaling Tool-Integrated RL." *arXiv Preprint arXiv:2503.23383*.

Liu, Jiawei, Chunqiu Steven Xia, Yuyao Wang, et al. 2023. "Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of LLMs for Code Generation." *Advances in Neural Information Processing Systems 36*.

Muennighoff, Niklas, Alexander M. Rush, Boaz Barak, et al. 2023. "Scaling Data-Constrained Language Models." *arXiv Preprint arXiv:2305.16264*.

Pan, Jiayi, Xingyao Wang, Graham Neubig, et al. 2024. "Training Software Engineering Agents and Verifiers with SWE-Gym." *arXiv Preprint arXiv:2412.21139*.

Pineau, Joelle, Philippe Vincent-Lamarre, Koustuv Sinha, et al. 2021. "Improving Reproducibility in Machine Learning Research (a Report from the NeurIPS 2019 Reproducibility Program)." *Journal of Machine Learning Research* 22 (164): 1–20.

Qwen Team. 2026. *Qwen3.5-4B: Model Card and Configuration*. https://huggingface.co/Qwen/Qwen3.5-4B.

Rafailov, Rafael, Archit Sharma, Eric Mitchell, et al. 2023. "Direct Preference Optimization: Your Language Model Is Secretly a Reward Model." *Advances in Neural Information Processing Systems 36*.

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. "Proximal Policy Optimization Algorithms." *arXiv Preprint arXiv:1707.06347*. https://arxiv.org/abs/1707.06347.

Shao, Zhihong, Peiyi Wang, Qihao Zhu, et al. 2024. "DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models." *arXiv Preprint arXiv:2402.03300*.

Tu, Dunwei, Hongyan Hao, Hansi Yang, et al. 2026. "ScaleEnv: Scaling Environment Synthesis from Scratch for Generalist Interactive Tool-Use Agent Training." *arXiv Preprint arXiv:2602.06820*.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, et al. 2017. "Attention Is All You Need." *arXiv Preprint arXiv:1706.03762*. https://arxiv.org/abs/1706.03762.

Xi, Zhiheng, Yiwen Ding, Wenxiang Chen, et al. 2024. "AgentGym: Evolving Large Language Model-Based Agents Across Diverse Environments." *arXiv Preprint arXiv:2406.04151*.

Xi, Zhiheng, Jixuan Huang, Chenyang Liao, et al. 2025. "AgentGym-RL: Training LLM Agents for Long-Horizon Decision Making Through Multi-Turn Reinforcement Learning." *arXiv Preprint arXiv:2509.08755*.

Yang, John, Kilian Lieret, Carlos E. Jimenez, et al. 2025. "SWE-Smith: Scaling Data for Software Engineering Agents." *arXiv Preprint arXiv:2504.21798*.

Yang, John, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. 2023. "InterCode: Standardizing and Benchmarking Interactive Coding with Execution Feedback." *Advances in Neural Information Processing Systems 36 (Datasets and Benchmarks Track)*.

Yue, Yang, Zhiqi Chen, Rui Lu, et al. 2025. "Does Reinforcement Learning Really Incentivize Reasoning Capacity in LLMs Beyond the Base Model?" *arXiv Preprint arXiv:2504.13837*.

Zelikman, Eric, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. 2022. "STaR: Bootstrapping Reasoning with Reasoning." *arXiv Preprint arXiv:2203.14465*. https://arxiv.org/abs/2203.14465.

Zhong, Victor, Caiming Xiong, and Richard Socher. 2017. "Seq2SQL: Generating Structured Queries from Natural Language Using Reinforcement Learning." *arXiv Preprint arXiv:1709.00103*. https://arxiv.org/abs/1709.00103.

Zhou, Chunting, Pengfei Liu, Puxin Xu, et al. 2023. "LIMA: Less Is More for Alignment." *arXiv Preprint arXiv:2305.11206*.

Zhu, Kejian, Zhuoran Jin, Dongqi Huang, et al. 2026. "Beyond Simply Environment Scaling: Designing Effective Environment Distributions for Multimodal Agent Learning." *arXiv Preprint arXiv:2608.03571*.

Zhuo, Terry Yue, Minh Chien Vu, Jenny Chim, et al. 2024. "BigCodeBench: Benchmarking Code Generation with Diverse Function Calls and Complex Instructions." *arXiv Preprint arXiv:2406.15877*.

A Additional figure

The study workflow is provided here as an appendix overview; the model and update schematic appears in the main text.

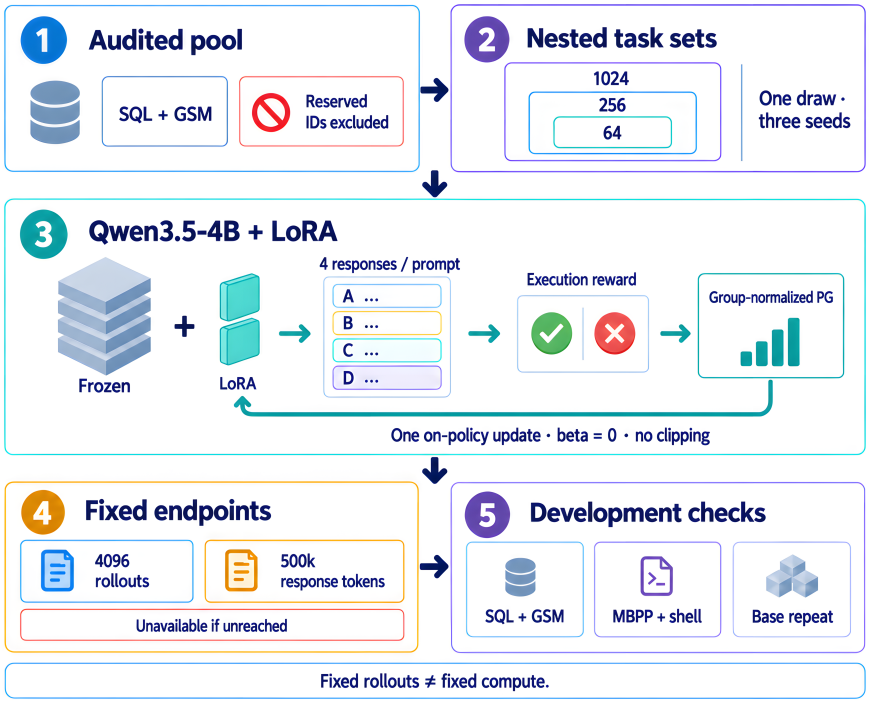

Figure 5: Appendix overview of the study workflow. Step 1 defines the audited SQL–GSM pool and excludes reserved IDs. Step 2 forms nested task sets with $N = 64$, 256, and 1,024 from one draw and evaluates three training seeds. Step 3 combines the frozen Qwen3.5-4B backbone with a trainable LoRA adapter, samples four responses per prompt, verifies execution, and applies group-normalized policy gradient. Step 4 measures the fixed 4,096-rollout and 500,000-response-token endpoints, marking an endpoint unavailable when the registered threshold is not reached. Step 5 checks SQL–GSM, MBPP–shell transfer, and repeated base outputs. The model architecture and update path are detailed in Figure 1.

B Detailed result tables

The following tables preserve the complete condition-level records that support the main-text summaries. They are supplementary and are not required to follow the main narrative.

Table 4: Every frozen quantity condition. N is the number of available training tasks and Seed is the training random seed. "Correct / 62" is the total number correct on the 62-task development set; the two SQL/GSM columns are macro accuracy at the 500,000-token and 4,096-rollout endpoints. Percentages average the SQL and GSM task-domain accuracies; "Unavailable" means the registered token threshold was not reached.

N	Seed	500k-token correct / 62	4096-rollout correct / 62	500k-token SQL/GSM (%)	4096-rollout SQL/GSM (%)
64	0	42	43	81.25	81.25
64	1	43	40	81.25	81.25
64	2	41	40	81.25	81.25
256	0	43	45	87.50	87.50
256	1	Unavailable	44	Unavailable	87.50
256	2	44	48	84.38	90.63
1024	0	44	40	87.50	84.38
1024	1	42	42	81.25	81.25
1024	2	43	44	84.38	87.50

Table 5: Fixed-endpoint shared-population comparison, not a compute-matched ranking. "Approach" is the training method; "Training signal" identifies the supervision or reward source; "Registered fixed endpoint" gives the update or response limit; and "Distinct cost or support issue" records the method-specific acquisition or optimization difference. Three seeds are used for each approach, and one final checkpoint is evaluated per run.

Approach	Training signal	Registered fixed endpoint	Distinct cost or support issue
SFT	Verified source reference	128 updates; 1024 presentations	Additional reference information
DPO	Base-generated correct/incorrect pair	128 updates; 1024 pairs	Pair availability and reference forward passes
RFT adaptation	Base-generated correct response	128 updates; 1024 presentations	Positive-response support and sampling cost
RLOO	Online reward from execution; leave-one-out baseline	4096 responses	KL term; official sampler
PPO	Online reward from execution; learned critic	4096 responses	Critic, KL, two PPO epochs per batch

Table 6: Fixed final checkpoints on the exposed development set. "Correct / 62" counts correct tasks, and "SQL/GSM macro" averages the two in-domain accuracies. Each value is the mean ± sample SD across three training seeds for one task and acquisition draw; SD is seed variation, not uncertainty over independent datasets.

Method	Correct / 62, mean ± SD across training seeds	SQL/GSM macro, mean ± SD across training seeds
PG ($N = 256$, rollout endpoint)	45.67 ± 2.08	88.54% ± 1.80%
SFT	35.67 ± 0.58	62.50% ± 5.41%
DPO	36.67 ± 5.13	62.50% ± 13.62%
RFT	41.00 ± 1.73	81.25% ± 3.12%
RLOO	42.67 ± 1.15	84.38% ± 0.00%
PPO	45.33 ± 2.08	87.50% ± 0.00%

Table 7: All registered conditions retained. "Training condition" identifies the task mix or reference stratum; "Correct counts" lists the three seed-level totals; and "SQL/GSM macro" is the mean ± sample SD across seeds. These are development observations, not independent final-test results.

Training condition	Correct counts, seeds 0, 1, and 2	SQL/GSM macro, mean ± SD across training seeds
mixed	45 / 42 / 41	83.33% ± 1.80%
sql only	44 / 46 / 44	87.50% ± 3.12%
gsm only	40 / 42 / 39	79.17% ± 1.80%
reference low	44 / 39 / 44	82.29% ± 3.61%
reference high	43 / 45 / 44	84.38% ± 3.12%

Table 8: Final test results at the registered pipeline-budget endpoints. "GSM8K accuracy" and "MBPP accuracy" are domain-specific percentages; "Macro accuracy" is their arithmetic mean; and "Macro change from base" is the percentage-point difference from the unadapted checkpoint. Entries are mean ± sample SD over three training seeds for one fixed training population, not confidence intervals over independent datasets.

Method	GSM8K accuracy (%)	MBPP accuracy (%)	Macro accuracy (%)	Macro change from base (pp)
Base	75.50	92.00	83.75	0.00
SFT	65.33 ± 1.26	55.67 ± 1.26	60.50 ± 1.09	-23.25
RFT	74.50 ± 0.87	89.50 ± 0.87	82.00 ± 0.75	-1.75
DPO	75.00 ± 1.80	90.00 ± 1.32	82.50 ± 0.75	-1.25
RLOO	74.50 ± 0.50	91.17 ± 0.29	82.83 ± 0.29	-0.92
PPO	73.50 ± 1.80	90.17 ± 1.04	81.83 ± 0.52	-1.92
PG	75.33 ± 1.26	89.67 ± 0.76	82.50 ± 0.90	-1.25

Table 9: Training pipeline time only. "Mean pipeline seconds" is elapsed training-pipeline time at the registered budget; "Complete updates, seeds 0, 1, and 2" lists the number of completed optimizer updates for each seed. Update counts remain method-dependent and are not treated as equal computational work.

Method	Mean pipeline seconds	Complete updates, seeds 0, 1, and 2
SFT	7205.10	812 / 774 / 875
RFT	7204.70	793 / 773 / 796
DPO	7205.53	653 / 657 / 659
RLOO	7250.46	170 / 158 / 178
PPO	7280.03	41 / 55 / 85
PG	7237.17	141 / 266 / 144

Table 10: Means and sample SD over the listed seeds. "Condition / endpoint" identifies the task allocation and stopping rule; "Seeds" lists the included seed IDs; GSM8K, MBPP, and Macro are final-test percentages. The two-seed $N = 256$ response-token result is explicitly incomplete and is not substituted for a three-seed estimate.

Condition / endpoint	Seeds	GSM8K (%)	MBPP (%)	Macro (%)
$N = 64$ / 4,096 rollouts	0,1,2	75.83 ± 0.29	90.67 ± 1.04	83.25 ± 0.43
$N = 64$ / 500k tokens	0,1,2	74.67 ± 1.76	90.50 ± 0.50	82.58 ± 1.13
$N = 256$ / 4,096 rollouts	0,1,2	75.17 ± 1.53	91.67 ± 0.76	83.42 ± 0.80
$N = 256$ / 500k tokens	0,2	75.25 ± 0.35	91.75 ± 0.35	83.50 ± 0.35
$N = 1{,}024$ / 4,096 rollouts	0,1,2	76.50 ± 1.73	89.83 ± 1.15	83.17 ± 0.29
$N = 1{,}024$ / 500k tokens	0,1,2	76.67 ± 1.15	90.83 ± 2.57	83.75 ± 1.52
Mixed	0,1,2	75.17 ± 1.76	91.33 ± 1.44	83.25 ± 1.56
SQL only	0,1,2	75.00 ± 2.29	91.00 ± 0.87	83.00 ± 0.75
GSM only	0,1,2	76.67 ± 2.02	90.00 ± 0.87	83.33 ± 1.42
Reference low	0,1,2	75.17 ± 1.26	90.67 ± 1.04	82.92 ± 0.76
Reference high	0,1,2	77.17 ± 1.26	90.67 ± 2.08	83.92 ± 0.52
SFT / fixed128	0,1,2	65.33 ± 1.04	58.33 ± 2.75	61.83 ± 1.28
DPO / fixed128	0,1,2	74.50 ± 1.32	91.17 ± 0.58	82.83 ± 0.80
RFT / fixed128	0,1,2	75.50 ± 1.32	90.50 ± 0.50	83.00 ± 0.50
RLOO / step512	0,1,2	73.33 ± 0.58	90.00 ± 0.50	81.67 ± 0.52
PPO / step512	0,1,2	75.67 ± 1.26	91.83 ± 0.29	83.75 ± 0.75

Table 11: Differences use the same final tasks and shared seed indices. "Prespecified contrast" names the comparison, "Paired seeds" lists the shared seeds, and "Macro differences" gives the seed-level percentage-point differences. These are descriptive paired differences, not significance tests; the incomplete token contrast uses only seeds zero and two.

Prespecified contrast	Paired seeds	Macro differences (pp)
$N = 256$ minus $N = 64$ / rollout4096	0,1,2	+0.25 / -1.00 / +1.25
$N = 1{,}024$ minus $N = 64$ / rollout4096	0,1,2	-0.50 / +0.00 / +0.25
$N = 256$ minus $N = 64$ / response500k	0,2	-0.50 / +2.25
$N = 1{,}024$ minus $N = 64$ / response500k	0,1,2	+0.75 / -0.50 / +3.25
sql_only minus mixed	0,1,2	-2.00 / -0.50 / +1.75
gsm_only minus mixed	0,1,2	-3.25 / +1.00 / +2.50
reference high minus low	0,1,2	+1.75 / +1.50 / -0.25
$N = 256$ PG minus SFT	0,1,2	+23.00 / +21.00 / +20.75
$N = 256$ PG minus DPO	0,1,2	+1.25 / -1.25 / +1.75
$N = 256$ PG minus RFT	0,1,2	+0.25 / -0.50 / +1.50
$N = 256$ PG minus RLOO	0,1,2	+2.25 / +1.25 / +1.75
$N = 256$ PG minus PPO	0,1,2	+0.75 / -1.25 / -0.50