Training Small Language Models with Group Relative Policy Optimization for Calculator Tool Calling

Abinesh Mathivanan[*,1] **and Harshad Saykhedkar**[1]
[1]Ablecredit

This technical report studies the behavior of *Group Relative Policy Optimization* (GRPO) when applied to *small language models* (SLMs) in the 0.6B–2B parameter regime. We examine whether performance gains from reinforcement learning primarily arise from re-weighting the model's existing output distribution effectively improving Pass@k rather than inducing fundamentally new reasoning strategies. Experiments are conducted in two settings: multi-step arithmetic generation and GSM8K-style tool-calling. Training is performed on the full datasets without offline difficulty filtering, with easy and hard problems interleaved throughout optimization. A multi-objective scalar reward is used to jointly enforce XML structural validity, answer correctness, and conciseness. We observe model-dependent responses to GRPO. For `Llama3.2-1B`, accuracy improves from 24.55% to 51.63%, indicating that GRPO can significantly amplify pre-existing solution trajectories even in the absence of curriculum design. In contrast, `Olmo-2 1B` exhibits a performance regression of 2.53%, which we attribute to early-stage training instability marked by rapid KL divergence growth and a weak instruction-following prior. These results highlight the sensitivity of GRPO to the stability of the initial policy and to the interaction between reward scaling and policy regularization. In the second experiment, a comparison of tool-calling performance across architectures shows that explicit support for intermediate reasoning is a dominant factor for SLMs. The `Qwen3-0.6B (Thinking)` model achieves 49.5% accuracy, substantially outperforming larger non-reasoning models such as `Qwen2.5-1.5B` (23.50%) and `Llama3.2-1B` (14.56%). Overall, the results indicate that GRPO is effective at improving selection reliability in SLMs on edge-compatible hardware, but does not compensate for weak instruction tuning or the absence of architectural reasoning mechanisms.
This experimentation was conducted in association with *Ablecredit (Lightbees Technologies Private Limited)*. Models can be found on our official Hugging Face page.

1. Introduction

Reinforcement learning (RL) has emerged as an effective mechanism for improving the reliability and task performance of language models beyond supervised fine-tuning, particularly through policy-gradient methods derived from Proximal Policy Optimization (PPO) (Schulman et al., 2017). While recent work has demonstrated strong gains from RL in large-scale models, the behavior of RL algorithms in the *small language model* (SLM) regime remains insufficiently characterized. This gap is practically important, as models in the 0.6B - 2B parameter range are central to deployment on edge and resource-constrained hardware, where improvements must be achieved under strict memory and compute constraints. In this work, we study *Group Relative Policy Optimization* (GRPO) as a standalone RL algorithm for SLMs. GRPO was introduced in the context of mathematical reasoning and shown to be competitive with PPO-style RL while avoiding an explicit value function (Shao et al., 2024). Notably, prior analysis argues that RL primarily sharpens the model's existing output distribution, improving Pass@k through re-ranking rather than inducing new reasoning behaviors. Motivated by this claim, we evaluate GRPO under controlled settings to examine whether its effects

are limited to output selection or whether it can also enable qualitatively harder problem solving in small models. Our first experiment uses an artificially constructed arithmetic dataset consisting of 1,500 training samples and 158 test samples, covering multi-step numerical reasoning with mixed operations. Each sample is structured as `(question, expression, answer)`, though only the question and final answer are used during training. Reinforcement learning is performed using *pure GRPO*, without any supervised loss component or offline difficulty filtering; easy and hard problems are interleaved throughout training. A multi-objective scalar reward is applied, combining XML format validation, numerical correctness, conciseness, and strict structural constraints, yielding a maximum reward of 2.4. Evaluation is conducted using few-shot prompting to isolate the contribution of RL from prompt engineering effects. The second experiment focuses on GSM8K-style mathematical reasoning with explicit tool invocation. GSM8K is a well-established benchmark for evaluating multi-step arithmetic reasoning in language models (Cobbe et al., 2021). In this setting, the model is required to generate a structured `<calculator>` tool call while producing a short reasoning trace. The dataset is reformatted into `(question, answer)` pairs and trained using GRPO with purely scalar rewards, without intermediate reasoning supervision. Three reward components - XML structure, correctness, and brevity are used, with a maximum reward of 1.4. This formulation aligns with recent work on tool-augmented reasoning, where correctness is enforced through execution rather than textual explanation (Yao et al., 2023). Across both experiments, we train and evaluate multiple instruction-tuned SLMs, including `Llama3.2-1B`, `Olmo2-1B`, `LFM2.5-1.5B`, `GLM-Edge-1.5B`, `Qwen2.5-1.5B`, and `Qwen3-0.6B (Thinking)`. Our results yield three key observations. First, GRPO can substantially improve performance in SLMs, as shown by the increase of `Llama3.2-1B` from a 26.4% baseline to 51.63% accuracy after a single epoch of RL. Second, contrary to the assumption that RL only re-ranks existing solutions, we observe cases where models successfully solve problems for which no correct rollout was observed prior to training, indicating limited but non-trivial generalization beyond pure Pass@k optimization. Third, we identify clear failure modes: `Olmo2-1B` experiences performance degradation after one epoch, consistent with early RL destabilization caused by excessive KL divergence and weak instruction-following priors, a phenomenon documented in prior RL and RLHF studies (Ouyang et al., 2022; Pavse et al., 2023). Finally, results from the GSM8K tool-calling experiment emphasize the importance of architectural support for reasoning. Despite its smaller size, `Qwen3-0.6B (Thinking)` substantially outperforms larger non-reasoning models after RL, indicating that GRPO amplifies existing reasoning structure rather than compensating for its absence.

2. RL with Pure GRPO

We employ GRPO formulation as the sole optimization mechanism across all experiments. The choice is motivated by compute efficiency, stability considerations in the small-model regime, and the structure of our reward signals. In particular, all training is performed without a learned value function, without supervised auxiliary losses, and with purely scalar reward feedback.

Our target models operate in the 0.6B-2B parameter range and are trained under strict memory and throughput constraints. Introducing a value head, as in PPO-style methods, increases both memory footprint and optimization complexity, while offering limited benefit when rewards are sparse, non-dense, and evaluated only at sequence completion (Schulman et al., 2017). Instead, we rely on relative reward normalization across multiple rollouts per prompt, which provides a low-variance learning signal without requiring a learned baseline. This design choice is aligned with prior observations that RL fine-tuning of language models primarily sharpens selection among existing outputs rather than inducing new reasoning primitives (Shao et al., 2024).

2.1. Optimization Setup.

For each input prompt x, we sample a group of K rollouts $\{y^{(1)}, \ldots, y^{(K)}\}$ from the frozen reference policy $\pi_{\theta_{\text{old}}}$. Each rollout is assigned a scalar reward $r(y^{(k)}, x)$ computed at the sequence level. We define the group-relative advantage as

$$\hat{A}^{(k)} = r(y^{(k)}, x) - \frac{1}{K} \sum_{j=1}^{K} r(y^{(j)}, x),$$

which removes global reward scale dependence and ensures that policy updates are driven by intra-group ranking rather than absolute reward magnitude (Shao et al., 2024).

The policy update is performed using a clipped likelihood-ratio objective:

$$\mathcal{L}_{\text{GRPO}} = \mathbb{E}_x \left[\frac{1}{K} \sum_{k=1}^{K} \min\left(\rho^{(k)} \hat{A}^{(k)}, \text{clip}\left(\rho^{(k)}, 1-\epsilon, 1+\epsilon \right) \hat{A}^{(k)} \right) \right], \quad \rho^{(k)} = \frac{\pi_\theta(y^{(k)} \mid x)}{\pi_{\theta_{\text{old}}}(y^{(k)} \mid x)}.$$

No value regression term or entropy bonus is used. Regularization is applied exclusively through an explicit KL penalty with respect to the frozen reference policy:

$$\mathcal{L}_{\text{KL}} = \beta \, \mathbb{E}_{x, y \sim \pi_\theta} \left[\text{KL}(\pi_\theta(\cdot \mid x) \parallel \pi_{\text{ref}}(\cdot \mid x)) \right],$$

yielding the final optimization objective

$$\mathcal{L} = \mathcal{L}_{\text{GRPO}} - \mathcal{L}_{\text{KL}}.$$

2.2. Reward Structure.

All rewards are scalar and evaluated at sequence completion. For arithmetic generation, rewards combine XML structural validity, numerical correctness, brevity, and strict formatting constraints, with a maximum total reward of 2.4. For GSM8K tool-calling, rewards consist of XML validity, execution correctness, and conciseness, with a maximum reward of 1.4. No intermediate reasoning supervision or token-level shaping is applied.

3. Experiment 1: Complex Multi-Step Arithmetic Reasoning

3.1. Task and Dataset Design

The primary objective of Experiment 1 was to evaluate the capacity of 1B-class models to parse natural language arithmetic queries into executable, nested hierarchical structures. We synthesized an artificial dataset comprising 1,500 training samples and 158 test samples from the repository (Danau5tin, 2025). The dataset follows a curriculum-like distribution of difficulty:

- **Easy:** Single-step operations (e.g., "Add 5 and 10").
- **Medium:** Two-step operations requiring basic nesting (e.g., "Calculate 9876543210 divided by the sum of 987 and 654?".)
- **Hard:** Complex multi-step queries requiring deep recursive nesting of the YAML schema (e.g., "Calculate the result of 987654 divided by 321, multiplied by the result of 654321 divided by 987, multiplied by the result of 123456 divided by 789?").

The models were tasked with generating a valid YAML-based tool call wrapped in `<calculator>` tags, adhering to a recursive schema where `operands` can either be scalar values or nested `operation` dictionaries. An illustrative example of the expected hierarchical mapping is provided in Box 1.

Box 1: Sample Hierarchical Tool Call Encoding

Question: Subtract 50 from 100, divide result by 2, then multiply by 10.

Expected Tool Call:

```
<calculator>
operation: "multiply"
operands:
  - operation: "divide"
    operands:
      - operation: "subtract"
        operands:
          - 100
          - 50
      - 2
  - 10
</calculator>
```

3.2. Training Configuration and Infrastructure

Training was conducted on a single **NVIDIA A10 (24GB)** GPU. Due to memory constraints and the specific rollout requirements of GRPO, `vLLM` was disabled to prioritize memory allocation for the policy and reference models. Table 1 summarizes the primary hyperparameters.

Table 1 | Training configuration for Multi-Step Arithmetic fine-tuning.

Parameter	Setting
Epochs	1
Learning Rate	Fixed at 3×10^{-6} (constant, no warmup)
Advantage Estimator	GRPO
Rollouts (G)	8 per prompt
Max Prompt Length	512 tokens
Max Completion Length	128 tokens
Batch Size	8
Gradient Clipping	1.0
Precision	BF16

3.3. Reward Mechanism Design

We utilized a multi-objective scalar reward system $R_{total} = \sum r_i$, where $\max(R_{total}) = 2.4$. The reward functions were designed to decouple formatting from logical accuracy:

1. **Correctness Reward ($r_{corr} \in \{0, 2.0\}$):** The generated YAML is recursively parsed and executed. A reward of 2.0 is granted if the output y satisfies $|f(y) - \text{truth}| \leq 0.1$, accounting for floating-point variances.
2. **YAML Format Reward ($r_{xml} \in \{0, 0.2\}$):** Validates that the output is a valid YAML structure containing the required `'operation'` and `'operands'` keys.
3. **Strict Structure Reward ($r_{struct} \in \{-0.1, 0.2\}$):** Enforces a singular instance of the `<calculator>` and `</calculator>` tags.

4. **Anti-Yap Reward ($r_{yap} \in [-0.5, 0]$):** A penalty-based function designed to enforce conciseness. It penalizes:
    - Presence of "User:" or "Assistant:" headers (−0.1).
    - Text existing outside the XML tags (−0.1 per occurrence).
    - Completion length exceeding 200 tokens (−0.1).
    - Redundant tag sets (−0.1).

3.4. Prompt Design and Few-Shot Mimicry

The system prompt used during training (see Box 2) was designed to enforce strict adherence to a nested YAML schema. A critical observation during the initial tuning phase was that the inclusion of multiple few-shot examples induced a **degenerative mimicry behavior** across the Llama, Qwen, and LFM model families (Tang et al., 2025). Rather than generalizing the hierarchical logic, these models tended to replicate the specific operands or structures of the few-shot examples even when they were irrelevant to the current question.

Box 2: RL System Prompt

```
You are a mathematical translation engine. Convert math problems into
a SINGLE, VALID, NESTED YAML calculator tool call.

### OUTPUT FORMAT:
Your entire response must be wrapped in <calculator> tags.
Inside the tags, use YAML syntax.
Do not include any text, explanations, or thinking outside/inside tags.

### VALID OPERATIONS:
- add, subtract, multiply, divide

### YAML SCHEMA:
Each operation is a dictionary with:
- operation: (string)
- operands: (list of numbers OR nested operation dictionaries)

### EXAMPLE (Nested Problem: "Multiply 3 by 4, then add 10"):
<calculator>
operation: "add"
operands:
  - operation: "multiply"
    operands:
      - 3
      - 4
  - 10
</calculator>

### GOLDEN RULES:
1. ROOT FIRST: The last operation mentioned is the top-level operation.
2. NESTING: Represent nested steps as dictionaries, not strings.
3. NO EXTRA TEXT: Only output the tags and the YAML.
```

3.5. Results

The empirical evaluation of the 1B-class model cohort under the GRPO framework reveals a significant divergence in architectural plasticity. As illustrated in Figure 1, the Llama 3.2 1B model demonstrated the most substantial relative improvement, while the Qwen 2.5 1.5B model established the highest absolute performance baseline. In contrast, the Olmo-2 1B Instruct model exhibited a unique performance degradation, providing a critical data point regarding the stability thresholds of reinforcement learning in small-scale models. Notably, all measurements were conducted in a zero-shot setting using a single, fixed evaluation prompt augmented with illustrative examples.

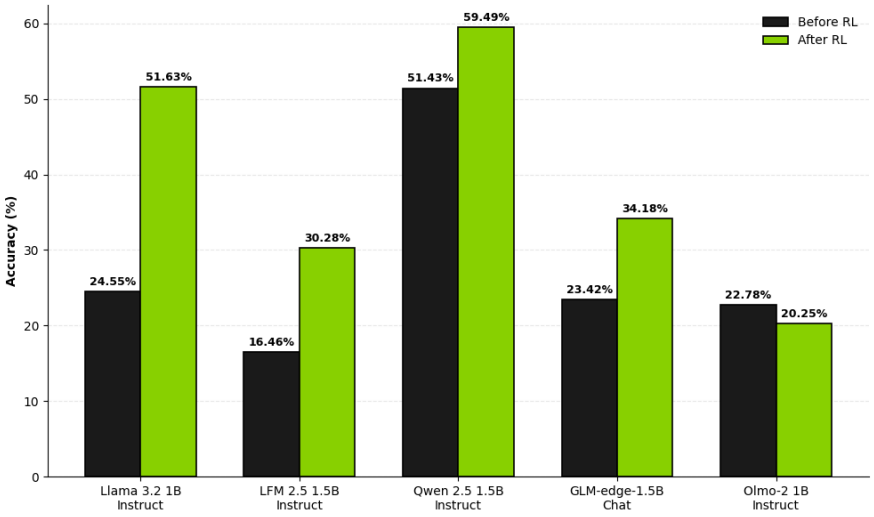
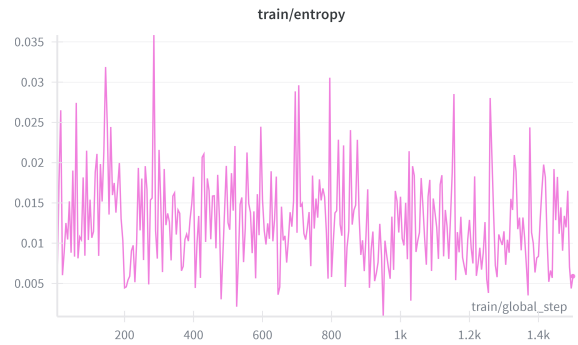


Figure 1 | Comparative Zero-Shot Accuracy of the 1B-class cohort before and after GRPO optimization. The green bars represent post-RL performance, showing a significant uplift in Llama, LFM, and Qwen families, while Olmo-2 1B shows a negative delta of -2.53%.

To evaluate the optimization trajectory of the models, we analyzed the reward and entropy logs across three distinct architectures as shown in (Figures 2, 3, and 4).

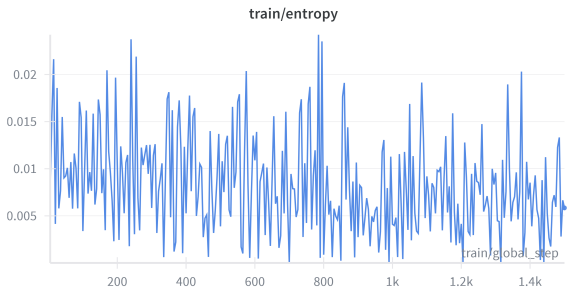
(a) Llama3.2-1B: Train Reward



(b) Llama3.2-1B: Train Entropy

Figure 2 | The reward trajectory for LLama3.2-1B (a) illustrates high initial variance but consistent convergence toward the 2.4 maximum. The sustained high entropy (b) indicates that the model maintained a broad exploration manifold throughout the training process. We confirm the exploration by cross-examining YAML & correctness rewards throughout the training.

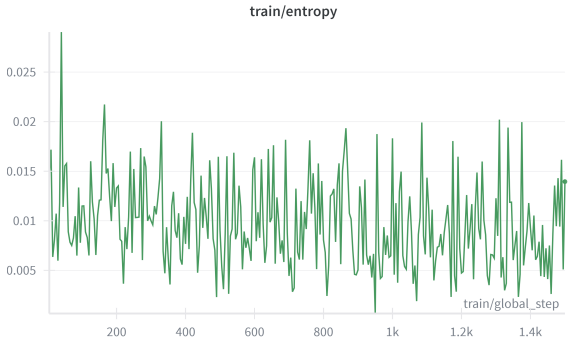
(a) Qwen2.5-1.5B: Train Reward



(b) Qwen2.5-1.5B: Train Entropy

Figure 3 | Qwen2.5-1.5B shows a denser reward saturation (a) and a smoother entropy decay (b) than the Llama variant, showing a more efficient alignment with the YAML schema due to its superior pre-trained instruct base.

(a) Qwen2.5-0.5B: Train Reward



(b) Qwen2.5-0.5B: Train Entropy

Figure 4 | Qwen2.5-0.5B, despite the limited parameter count, demonstrates rapid policy locking (a) and a sharp entropy reduction (b) to approximately 0.005, signifying a swift transition to formatting specialization.

In the case of Llama 3.2 1B, the dramatic shift from 24.55% to 51.63% was characterized by the model's ability to solve "Hard" tier problems that initially yielded zero successful completions in early-epoch rollout groups ($k = 8$). By isolating the relative advantage of infrequent successful paths within a group, GRPO effectively shifted the policy's log-probability mass toward these correct hierarchical structures while simultaneously suppressing conversational "yap" and structural non-compliance. We observe in the Llama family logs that further suggests that the model avoided premature local minima, allowing it to generalize the "ROOT FIRST" and nesting logic across the dataset (Ni et al., 2025). Similarly, reward trajectories reveal substantial fluctuations throughout training, indicating that the mixed-difficulty dataset, without explicit curriculum or difficulty stratification, frequently failed to produce consistent structural rewards in early epochs. In such cases, rollout groups often exhibited near-degenerate relative advantages, resulting in weak or uninformative gradient signals and reduced effective learning capacity. We hypothesize that `Llama3.2-1B` was less susceptible to this failure mode due to a more expressive and stable pre-trained policy manifold, which maintained sufficient diversity over valid hierarchical and XML-compliant outputs. This implicit exploratory capacity likely allowed the model to recover sparse reward signal and generalize structural patterns despite noisy supervision, ultimately leading to higher final accuracy. In contrast, the regression

observed in Olmo-2 1B Instruct (22.78% to 20.25%) highlights a critical "stability threshold" in RL for SLMs. We mathematically interpret this as a manifestation of Early RL Destabilization (Pavse et al., 2023). In architectures where the initial Supervised Fine-Tuning (SFT) manifold is insufficiently robust, the high-variance gradients produced by the relative advantage calculations in GRPO can overwhelm the policy, causing it to drift into low-reward regions of the distribution before a stable reasoning path is established.

4. Experiment 2: GSM8K Tool-Calling and Augmented Reasoning

In the second experimental phase, we transitioned from pure arithmetic parsing to a more complex reasoning-and-execution task using the GSM8K dataset (Cobbe et al., 2021). The objective was to evaluate the ability of SLMs to utilize a `<calculator>` tool to solve grade-school math problems within a structured GRPO environment.

4.1. Task and Dataset Specification

We conduct experiments on the standard **GSM8K** dataset, comprising 7,470 training samples and 1,319 test samples. In contrast to conventional chain-of-thought (CoT) prompting, where arithmetic reasoning is implicitly embedded within free-form text, we require the model to externalize computation through an explicit calculator tool invocation. Given a natural language question, the model must first perform a brief internal reasoning step and then emit a structured, executable tool call of the form define in Box 3. The predicted answer is obtained by executing the generated arithmetic expression, and correctness is evaluated solely based on the tool's output.

Box 3: GSM8K Tool-Calling Format

User: Natalia sold 48 clips in April. In May she sold half as many. How many clips did she sell in total?

Assistant:

```
<thought>
- April clips: 48
- May clips: 48 / 2 = 24
- Total clips: 48 + 24 = 72
</thought>

<calculator>
operation: "add"
operands:
  - 48
  - operation: "divide"
    operands: [48, 2]
</calculator>
```

4.2. Training Configuration and Infrastructure

All GSM8K tool-calling experiments were conducted under a constrained single-GPU setup, and BF16 was preferred over FP16 due to memory limitations (Qi et al., 2025).

Table 2 | Training Configuration for GSM8K Tool-Calling

Component	Configuration
Advantage Estimator	GRPO
Learning Rate	1×10^{-5} [cosine scheduler and warmup ratio of 0.1]
Rollouts (G)	4 generations per prompt
Sequence Lengths	Max prompt length = 512; Max Completion length = 512
Batch Size	Per-device batch size = 4 with gradient accumulation steps = 8
Gradient Clipping	`max_grad_norm` = 1.0
Numerical Precision	BF16
Sampling Temperature	0.6 (Qwen-based models)

4.3. Reward Mechanism Design

For GSM8K tool-calling, we employ a scalar reward formulation

$$R_{\text{total}} = r_{\text{xml}} + r_{\text{corr}} + r_{\text{yap}}, \qquad \max(R_{\text{total}}) = 1.4,$$

where each component targets a distinct failure mode observed during early GRPO rollouts: structural incompleteness, arithmetic inaccuracy, and excessive or malformed generation.

1. **YAML and Structural Reward ($r_{\text{xml}} \in [0, 0.4]$):** This reward validates the presence of required structural markers. A partial reward of +0.2 is granted if the completion contains a closing `</think>` tag, ensuring termination of the reasoning segment. An additional +0.2 is awarded if a well-formed `<calculator>` $\cdots$ `</calculator>` block is detected via regular-expression matching. This component enforces minimal structural completeness without constraining the internal reasoning content.
2. **Correctness Reward ($r_{\text{corr}} \in \{0, 0.1, 1.0\}$):** The arithmetic expression embedded in the `<calculator>` block is parsed and executed. If the resulting value $f(y)$ satisfies

   $$|f(y) - y_{\text{gt}}| \leq \epsilon, \quad \epsilon = 10^{-4},$$

   a reward of 1.0 is assigned. If execution succeeds but the result is incorrect, a partial reward of 0.1 is granted, providing a weak learning signal for near-miss or structurally valid but incorrect solutions. All parsing or execution failures receive zero reward.
3. **Anti-Yap Reward ($r_{\text{yap}} \in [-0.3, 0]$):** A penalty-based reward designed to suppress verbosity and malformed outputs. Penalties are applied for:
   - Excessive text appearing after the `</think>` tag outside the `<calculator>` block (length > 25 characters, -0.1).
   - Multiple occurrences of `<calculator>` or `</think>` tags (-0.1).
   - Completions exceeding 512 tokens (-0.1).

4.4. Prompt Design

The prompt instructs the model to complete a bounded reasoning trace within a dedicated `<think>` block and to emit exactly one executable calculator invocation in a structured YAML format. The prompt restricts arithmetic operations to a fixed operator set (`add`, `subtract`, `multiply`, `divide`)

and enforces a single tool call per completion, thereby preventing both tool overuse and free-form arithmetic reasoning as defined in Box 4.

Box 4: System Prompt for GSM8K Tool-Calling

```
You are a mathematical reasoning agent.
1. Complete the logical steps started inside the <think> tag.
2. Close the reasoning with </think>.
3. Provide a SINGLE calculator tool call inside <calculator> tags.
4. Strictly follow YAML rules as stated above

Operations: add, subtract, multiply, divide.

Example:
<think>
[... thoughts ...]
</think>
<calculator>
operation: "add"
operands:
  - 48
  - operation: "divide"
    operands: [48, 2]
</calculator>
```

4.5. Results

The results, visualized in Figure 5, demonstrate that RL training significantly bridges the performance gap for 1B-class models. Notably, the **Qwen3-0.6B (Thinking)** model achieved a dominant **49.50%** accuracy post-RL. We observed that prior to RL, the thinking model produced excessive tokens and frequent "None" values, making baseline evaluation unreliable; however, GRPO successfully calibrated these thinking traces. Notably, all measurements were conducted in a zero-shot setting using a single, fixed evaluation prompt augmented with illustrative examples.

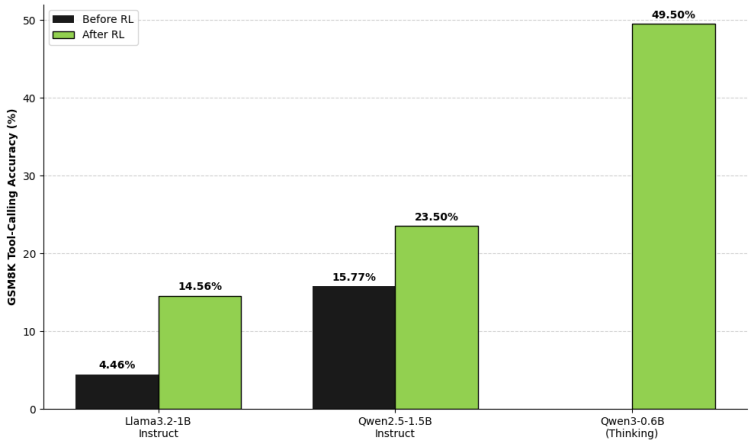


Figure 5 | All models show significant gains post-RL, with the 0.6B Thinking model establishing a performance ceiling nearly 2× higher than the 1.5B non-thinking variant.

(a) Llama3.2-1B: Train Reward

(b) Qwen2.5-1.5B: Train Reward

(c) Qwen3-0.6B (Think): Train Reward

(d) Qwen3-0.6B (Think): Max Completion Length

Figure 6 | Reward trajectory (a), (b) shows a steady climb, while entropy remains stable around 0.15 -0.2, indicating a well-regularized learning process without premature collapse. We could observe that the model quickly learnt the YAML format and trying to generalize over the correctness reward. Similarily Qwen3-0.6B (think) (c) displays stable reward growth fluctuating entropy (the major optimization task was to reduce the Max Completion length (d) from 512, which we gradually achieve over the steps resulting in the mean length of 180 tokens.)

4.5.1. Stabilization of Thinking Traces

A critical finding of Experiment 2 is that GRPO optimization improves not only task accuracy but also *reasoning efficiency*, as reflected in shorter internal thought traces and substantially reduced inference latency. In the baseline setting, the thinking model frequently exhibits high-entropy deliberation characterized by circular reasoning loops, redundant hypothesis testing, and delayed identification of the governing arithmetic constraints (Wang et al., 2025). Such behavior leads to an accumulation of logical noise across tokens, increasing the probability of constraint violation and incorrect tool invocation. Similar phenomena have been observed in prior work, where excessive chain-of-thought length was shown to degrade reasoning reliability due to cumulative logical drift and unstable intermediate representations (Hassid et al., 2026).

Post-GRPO, the policy exhibits a marked reduction in reasoning entropy. By optimizing relative advantage within rollout groups, GRPO implicitly sharpens the action distribution over intermediate reasoning steps, suppressing low-reward exploratory branches while reinforcing concise, high-utility transitions. This entropy contraction enables the model to converge more rapidly to invariant problem structure, as evidenced by the reduction in thought-token count from 408 to 139 in Table 3. Importantly, this compression does not sacrifice correctness; rather, it mitigates early destabilization effects that arise in long-horizon reasoning, where minor arithmetic or structural errors compound across steps

and propagate irreversibly to the final tool call. In our experiments, the average time-to-answer decreased from 45 seconds in the baseline model to 13.4 seconds after GRPO training, representing a 3.4× reduction in latency.

Overall, we observe that GRPO functions as an implicit regularizer over reasoning trajectories rather than merely an optimizer of final-answer reward. By favoring low - entropy, high-advantage reasoning paths within rollout groups, GRPO suppresses pathological over-deliberation while preserving the minimal logical structure required for correct tool execution. We additionally present these results as empirical support for prior work showing that shorter chain-of-thought representations can improve stability and accuracy in problem-solving models.

5. Conclusion

This technical report demonstrates that Small Language Models (0.6B–2B) can achieve strong arithmetic reasoning and structured tool-calling performance when optimized using Pure GRPO. Across multiple controlled experiments, we observe that GRPO is particularly effective in settings where correctness depends on ranking sparse successful reasoning trajectories rather than maximizing token-level likelihood. Our key findings are summarized as follows:

1. **Llama3.2-1B** can be optimized to solve complex multi-step arithmetic problems with nested YAML tool calls, achieving over 51% accuracy. This improvement is driven by GRPO's ability to amplify latent Pass@k successes and reallocate probability mass toward structurally correct reasoning paths.
2. **Thinking Architectures** exhibit a clear advantage for tool-augmented reasoning. We observe that a 0.6B thinking-oriented model can outperform a 1.5B standard SLM by nearly 2×, implies that architectural inductive bias is more impactful than parameter count alone in structured reasoning tasks.
3. **Reinforcement Learning as a Pruning Mechanism:** GRPO consistently reduces reasoning verbosity without sacrificing accuracy. In our analysis, thinking traces were shortened from 408 to 139 tokens, accompanied by a reduction in inference latency.

We additionally present these results as empirical support for prior work suggesting that shorter chain-of-thought representations can improve stability and accuracy in problem-solving models. The observed reduction in reasoning entropy and avoidance of premature local minima further indicate that GRPO implicitly regularizes long-horizon reasoning dynamics.

Future work will focus on characterizing the *stability threshold* that governs whether a given SLM architecture can sustain RL-based post-training without regression. In particular, we plan to investigate whether multi-stage supervised fine-tuning prior to GRPO can mitigate early RL destabilization effects observed in models such as Olmo-2 1B. Beyond this, we aim to extend this line of research toward more expressive thinking-model architectures, exploring how structured reasoning priors interact with group-based policy optimization at scale.

References

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Danau5tin. calculator_agent_rl. https://github.com/Danau5tin/calculator_agent_rl, 2025. Accessed: 2026-02-08.

Michael Hassid, Gabriel Synnaeve, Yossi Adi, and Roy Schwartz. Don't overthink it: Preferring shorter thinking chains for improved llm reasoning. *arXiv preprint arXiv:2505.17813*, 2026.

Kangqi Ni, Zhen Tan, Zijie Liu, Pingzhi Li, and Tianlong Chen. Can grpo help llms transcend their pretraining origin? *arXiv preprint arXiv:2510.15990*, 2025.

Long Ouyang, Jeffrey Wu, Xu Jiang, et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.

Brahma S. Pavse, Chen Xiao, Jemin Lee, and Swarat Chaudhuri. Learning to stabilize online reinforcement learning in unbounded state spaces. *arXiv preprint arXiv:2306.01896*, 2023.

Penghui Qi, Zichen Liu, Xiangxin Zhou, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Defeating the training-inference mismatch via fp16. *arXiv preprint arXiv:2510.26788*, 2025.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Zhihong Shao, Peiyi Wang, Qun Zhu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Yongjian Tang, Doruk Tuncel, Christian Koerner, and Thomas Runkler. The few-shot dilemma: Over-prompting large language models. *arXiv preprint arXiv:2509.13196*, 2025.

Zengzhi Wang, Fan Zhou, Xuefeng Li, and Pengfei Liu. Octothinker: Mid-training incentivizes reinforcement learning scaling. *arXiv preprint arXiv:2506.20512*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, et al. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2023.

Table 3 | Detailed Comparison of Reasoning Logic: Base vs. RL-Optimized Model

Question	Every day, Wendi feeds each of her chickens three cups of mixed chicken feed, containing seeds, mealworms and vegetables to help keep them healthy. She gives the chickens their feed in three separate meals. In the morning, she gives her flock of chickens 15 cups of feed. In the afternoon, she gives her chickens another 25 cups of feed. How many cups of feed does she need to give her chickens in the final meal of the day if the size of Wendi's flock is 20 chickens?	
Model State	**Base Thinking Model**	**RL Thinking Model (GRPO)**
Token Count	**408 Tokens**	**139 Tokens**
Thinking Trace	Okay, let's see. Wendi feeds her chickens three cups each day, in three separate meals. The problem says she gives 15 cups in the morning, 25 in the afternoon, and then the final meal? Wait, but the question is asking for the final meal. Wait, but if she feeds them three cups each day, how many total cups would that be? Let me check. Wait, the problem says she gives each chicken three cups each day. So if there are 20 chickens, then total feed per day would be 20 chickens multiplied by 3 cups. But the problem mentions three separate meals: morning, afternoon, and final meal. Wait, but maybe the final meal is the same as the third one? Let me read again. "In the morning, she gives her flock of chickens 15 cups of feed. In the afternoon, she gives her chickens another 25 cups of feed. How many cups of feed does she need to give her chickens in the final meal of the day if the size of Wendi's flock is 20 chickens?" So the final meal is the third one. So the total feed per day would be 15 + 25 + (3 cups per chicken * 20 chickens). Let me calculate that. 15 + 25 is 40, then 3*20 is 60. So total 40 + 60 = 100 cups per day. But the question is asking for the final meal, which would be 3 cups per chicken times 20 chickens. So that's 60 cups. But wait, the problem says "how many cups of feed does she need to give her chickens in the final meal". So the answer is 60. Let me confirm. Yes, because each chicken gets 3 cups, 20 chickens would be 60. So the final meal is 60 cups.	Okay, let's break down the problem step by step. First, Wendi feeds each chicken three cups each day. There are 20 chickens, so the total feed per day is 20 chickens * 3 cups = 60 cups. Then, she gives them in three meals: morning 15, afternoon 25, and the final meal. So the sum of the first two meals is 15 + 25 = 40. Subtract that from the total 60 to find the final meal: 60 - 40 = 20 cups. Therefore, the final meal requires 20 cups.
Tool Call	`<calculator> operation: "multiply" operands: [3, 20] </calculator>`	`<calculator> operation: "subtract" operands: [60, 40] </calculator>`
Status	**Incorrect**	**Correct**