Group Relative Policy Optimization (GRPO) On The Countdown Reasoning Task

Abhijeet Praveen[1,2] **Cormac Cureton**[1,2] **Aman Sidhu**[1,2]
COMP 579 - Group 4
[1]McGill University [2]MILA – Quebec AI Institute
`{abhijeet.praveen,cormac.cureton,aman.sidhu}@mail.mcgill.ca`

Abstract

We explore Group Relative Policy Optimization (GRPO) and its application to enhancing the reasoning capabilities of large language models (LLMs). This project implements GRPO in PyTorch and evaluates its effectiveness on the Countdown arithmetic reasoning benchmark using the Qwen2.5-1.5B-Instruct model. GRPO modifies Proximal Policy Optimization (PPO) by estimating advantages from group-level rewards, reducing dependency on value functions. Our experiments measure the performance of this GRPO algorithm, providing insights into its design choices. Our experiments show that the best GRPO fine-tuned model achieved a test mean reward of 0.18 and test mean accuracy of 10.9% with a group size of G=3, an improvement over the untrained baseline model, which achieved a mean reward of 0.122 and accuracy of 5%. These results highlight the potential of GRPO for improving LLM reasoning through more structured reinforcement learning. We release all our code to facilitate future research into scalable, efficient methods for enhancing LLM reasoning capabilities. [1]

1 Introduction

Large language models (LLMs) have demonstrated impressive capabilities across various domains, yet their performance on downstream, mathematical reasoning tasks remains limited [1]. Many reasoning tasks demand careful planning and numeric precision, capabilities that are difficult to master through pretraining alone. Therefore, LLM fine-tuning has become a popular technique to improve performance on specific tasks. Recent work has shown that fine-tuning with Supervised Fine-tuning (SFT) and Reinforcement Learning from Human Feedback (RLHF) can align model outputs with human preferences and improve factuality and helpfulness [2]. Among RLHF methods, Proximal Policy Optimization (PPO) [3] has become the de facto standard. However, PPO requires training a separate value network which increases memory requirements and is known to be sample-inefficient and sensitive to hyperparameter tuning [4].

To address these limitations, DeepSeek introduced Group Relative Policy Optimization (GRPO) [5], a variant of PPO, designed to enhance fine-tuning for multi-step reasoning by using group-level rewards. GRPO gained relevance after the release of DeepSeek-R1, a reasoning model trained much more efficiently than contemporary models [6]. GRPO eliminates the need for a value network and instead uses group-based sampling and reward normalization to compute advantages, and using an explicit KL divergence penalty. In this project, we implement GRPO from scratch in PyTorch and test its effectiveness on the countdown arithmetic reasoning task, a structured benchmark that challenges models to form valid expressions using a fixed set of integers to reach a target number. Our objective is to assess whether GRPO can yield meaningful improvements on downstream reasoning tasks, even for small-scale models such as Qwen2.5-1.5B-Instruct.

[1]`https://github.com/Cormac-C/grpo-project`

2 Background

Proximal Policy Optimization (PPO) Proximal Policy Optimization (PPO) is a policy gradient, actor-critic reinforcement learning algorithm designed to improve training stability by limiting how much the policy is updated at each step [3]. It builds on Trust Region Policy Optimization (TRPO) [7], but avoids the complexity of computing second-order gradients by instead introducing a clipped surrogate objective: $L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$ where the probability ratio is $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$, and $\hat{A}_t$ is an estimator of the advantage function at time step t. This formulation ensures that the policy does not shift too far from the current policy by clipping the objective when updates are too large, thus preventing performance collapse during training. In the context of LLM fine-tuning via reinforcement learning with human feedback (RLHF), PPO is often used to align the model's responses with preference scores or reward models, balancing exploration with stability.

Group Relative Policy Optimization (GRPO) Group Relative Policy Optimization Group Relative Policy Optimization (GRPO) addresses some limitations of PPO by removing the need for a separate value (critic) network, resulting in improved training stability and reduced memory overhead [5]. Instead of estimating the advantage via Generalized Advantage Estimation (GAE), GRPO samples a group of responses (actions) for each prompt (state) and computes a *group relative advantage* by standardizing the reward within that group: $A(s, a_i) = \frac{r(s,a_i)-\mu}{\sigma}, \quad \mu = \frac{1}{G}\sum_{j=1}^{G} r(s, a_j)$ where G is the number of sampled responses. This relative scoring focuses on ranking rather than absolute reward. Additionally, GRPO explicitly includes a KL divergence penalty between the current policy and a reference policy (usually the base model), rather than incorporating it into the advantage estimate like PPO. This KL term helps preserve the base model's behavior while allowing policy improvement based on relative performance. The new objective is,

$$J_{GRPO}(\theta) = \mathbb{E}\left[q \sim P(Q), \{o_i\}_{i=1}^{G} \sim \pi_{\theta_{\text{old}}}(O \mid q)\right]$$

$$\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\left\{\min\left[\frac{\pi_\theta(o_{i,t} \mid q, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t} \mid q, o_{i,<t})}\hat{A}_{i,t}, \text{clip}\left(\frac{\pi_\theta(o_{i,t} \mid q, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t} \mid q, o_{i,<t})}, 1-\epsilon, 1+\epsilon\right)\hat{A}_{i,t}\right] - \beta\mathbb{D}_{\text{KL}}[\pi_\theta||\pi_{ref}]\right\}$$

More recent works like DAPO [8] and Dr. GRPO [9] have looked to further improve the algorithms performance and reduce bias but this report only considers the original algorithm. There have been a variety of open-source implementations of GRPO which served as useful references for our work [10] [11] [12].

Countdown Reasoning Task Countdown is a constrained arithmetic task where the model receives a target number and a set of integers. The task is to generate an arithmetic expression using each number once to reach the target, requiring some mathematical reasoning [13]. Valid operations are limited to addition, subtraction, multiplication, and division. Each task has an unambiguous solution, allowing for simple automatic evaluation, making it well-suited for reinforcement learning [11]. See Appendix A for an example task.

3 Related Works

Supervised Fine-tuning (SFT) Supervised fine-tuning (SFT) is a widely used approach to adapt pretrained large language models (LLMs) to downstream tasks using labeled data consisting of input prompts and their corresponding target outputs. In SFT, the model is trained in a task-specific or domain-specific setting using standard supervised learning, this technique has been around since early LLMs like BERT [14] and GPT [15]. While SFT can significantly improve task accuracy, it often struggles to generalize beyond the training distribution meaning it's efficacy is limited by the diversity and quality of the labeled datasets available like GLUE [16] and SQuAD [17]. Moreover, SFT alone may not be sufficient for aligning model outputs with nuanced human preferences, such as helpfulness or tone. Instruction fine-tuning is a specialized form of SFT in which training examples are framed as instructions alongside relevant context. The model learns to follow a variety of natural language instructions, improving its ability to generalize across tasks and respond more coherently to user queries.

RL Fine-tuning LLMs To address the limitations of SFT discussed above, reinforcement learning has been employed to fine-tune language models to optimize objectives which may have multiple valid answers. To improve model helpfulness, InstructGPT introduced the concept of Reinforcement Learning from Human Feedback (RLHF) [2]. In this framework, the LLM acts as the agent, taking actions by generating tokens in response to a prompt, which serves as the state. To reflect human preferences, human annotators rank model-generated responses, and these rankings are used to derive reward signals. To improve model alignment, works like [18] leverage the related approach of Reinforcement Learning from AI Feedback (RLAIF) which derives its reward function from an LLM-based preference model. Additionally, RL fine-tuning has been leveraged to improve reasoning ability in LLMs in domains like math [19] [20] and coding [21]. The concept of RL finetuning for reasoning has been leveraged on a larger scale to create reasoning models like OpenAI o1 [22] and DeepSeek-R1 [6], the latter used GRPO during fine-tuning.

4 Methodology

We implemented the GRPO algorithm in PyTorch and using the Hugginface Transformers Library, with modular code structure to enable clean experimentation and reproducibility.

Dataset and Environment We created a custom PyTorch Dataset class that can load a local JSON or dataset hosted on HuggingFace [23]. Each instance includes a list of integers and a target value. Examples are formatted with prompts that follow a conversational format and prompt the assistant to output the final answer and justification in XML tags (ie. $< think >$ and $< answer >$). Prompts also include a one-shot example of the countdown problem, and formatting rules for the final answer. Prompts are included in Appendix B.

Reward Model and Evaluation A custom reward function parses the model-generated responses, and each response is rewarded based on the correctness of formatting and its final answer. If the response is syntactically valid and correctly evaluates to the target, the response receives a reward of 1.0. If the response is incorrect but syntactically correct, using only the available numbers and providing its final answer in $< answer >$ tags then it receives a reward of 0.1. If the syntax is incorrect then the response's reward is 0.0. These reward signals are computed for each of the G sampled outputs per prompt and stored for advantage calculation.

Sampling and Training Loop We sample G completions for each prompt using the current policy model with constant temperature T. Log probabilities are computed for each token in each output from three models: the current policy model, the reference model (frozen), and the model from the previous update (used for probability ratios). Thereafter, we compute the GRPO objective and loss function described in section 2 where the total loss is the negative mean of the GRPO objective across all outputs and tokens.

5 Experiments

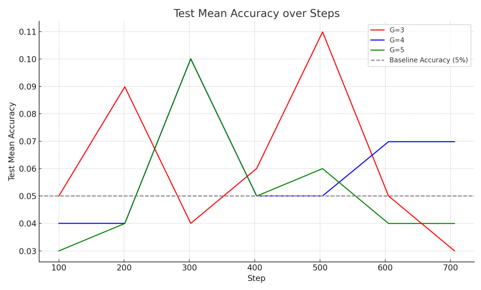
5.1 Experimental Setup

We conducted our experiments using the Qwen2.5-1.5B-Instruct model as the base policy network, loaded through the HuggingFace Transformers library. All training was performed on a single NVIDIA GH200 96GB GPU. The Countdown dataset was either generated via a search-based solver that ensures each sample has at least one valid arithmetic solution, or it was loaded through the available Countdown dataset on HuggingFace [23]. The training set was split into 90% training and 10% testing subsets. A PyTorch DataLoader was used with a batch size of 6 for training and 16 for evaluation. We fine-tuned the model for one epoch using the AdamW optimizer with a learning rate of 1×10^{-8}, betas of $(0.9, 0.999)$, and weight decay of 0.01.

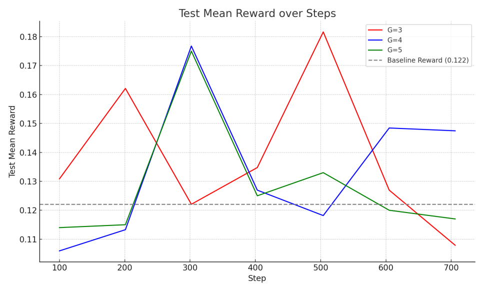
We evaluated different group sampling sizes of $G = 3, 4, 5$, varying the number of completions sampled for each prompt per iteration. The KL-divergence penalty coefficient was set to $\beta = 0.001$, and the clipping threshold ϵ was set to 0.1. Only one policy update ($\mu = 1$) was performed per GRPO iteration, simplifying the objective since the probability ratios are 1 between the old and current model. We limited generation to 512 new tokens with a temperature of 1.0. Model performance was measured using two metrics: average reward and accuracy. Accuracy was defined as the percentage of model completions that exactly matched the correct target using the allowed numbers and operations. Rewards were scored based on correctness as well as the formatting of the generated equation.

5.2 Results

Evaluation on the untrained base model achieved a mean reward of 0.122 and mean accuracy of 5%. Figure 1 shows the performance of models with different group sizes. We observe that the best performing GRPO fine-tuned model achieved a mean reward of 0.18 and accuracy of 10.9% with a group size of G=3, a moderate improvement over the baseline. The training curves for these experiments can be found in Appendix C.



(a) Test Accuracy



(b) Test Reward

Figure 1: Test Accuracy and Reward Throughout Training for Different Group Sizes (3, 4 and 5)

6 Conclusion

This project investigates the effectiveness of Group Relative Policy Optimization (GRPO) for enhancing the reasoning abilities of large language models in arithmetic settings. After implementing GRPO and fine-tuning Qwen2.5-1.5B-Instruct on the Countdown task with different group sizes we observe varying results, with one model moderately outperforms the base model. GRPO's group-level advantage calculation does reduce the memory requirements of the algorithm since it does not require a separate value network but the advantage signal was sparse in cases where the base model struggled to successfully reach the ground truth across the group. The results show GRPO's potential but motivate further investigation into the requirements for effectively implementing GRPO with smaller, less capable base models.

7 Future Works

Future work can extend this study in several directions. Firstly, due to computational constraints our hyperparameter ablation was limited to varying group size (G) but it would be valuable to investigate the impact of other hyperparameters like μ, β, and ϵ on the algorithm's performance in different domains. Additionally, since the advantage scores in GRPO are dependent on the relative quality of responses within a group then it would be interesting to study the performance of other relatively small open LLM models within the same context. It could also be valuable to scale GRPO to larger and more diverse reasoning benchmarks such as GSM8K [24] or SVAMP [25] to evaluate its performance in other arithmetic settings. Additionally, for these larger more diverse arithmetic benchmarks, integrating a learned reward model based on human feedback rather than evaluating strict correctness could help align model outputs with human preferences in open-ended tasks. Finally, this work was constrained to experiments on a single GPU, there is likely room to further optimize memory utilization with packages like verl, vLLM, or DeepSpeed to improve training performance. Additionally, further investigation into how this approach scales to larger models and more complex tasks with more computational resources would be valuable.

References

[1] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, and Ed H Chi. Self-Consistency Improves Chain of Thought Reasoning in Language Models. *arXiv preprint arXiv:2203.11171*, 2023.

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

[3] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, July 2017. URL `https://arxiv.org/abs/1707.06347v2`.

[4] Qisai Liu, Zhanhong Jiang, Hsin-Jung Yang, Mahsa Khosravi, Joshua R. Waite, and Soumik Sarkar. Enhancing PPO with Trajectory-Aware Hybrid Policies, February 2025. URL `http://arxiv.org/abs/2502.15968`. arXiv:2502.15968 [cs].

[5] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models, April 2024. URL `http://arxiv.org/abs/2402.03300`. arXiv:2402.03300 [cs].

[6] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, and et al. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning, January 2025. URL `http://arxiv.org/abs/2501.12948`. arXiv:2501.12948 [cs].

[7] John Schulman. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[8] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, and et al. DAPO: An Open-Source LLM Reinforcement Learning System at Scale, March 2025. URL `http://arxiv.org/abs/2503.14476`. arXiv:2503.14476 [cs].

[9] Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding R1-Zero-Like Training: A Critical Perspective, March 2025. URL `http://arxiv.org/abs/2503.20783`. arXiv:2503.20783 [cs].

[10] Amirhossein Kazemnejad, Milad Aghajohari, Alessandro Sordoni, Aaron Courville, and Siva Reddy. Nano Aha! Moment: Single File "RL for LLM" Library, 2025. URL `https://github.com/McGill-NLP/nano-aha-moment`.

[11] Jiayi Pan, Junjie Zhang, Xingyao Wang, Lifan Yuan, Hao Peng, and Alane Suhr. TinyZero, 2025. URL `https://github.com/Jiayi-Pan/TinyZero`.

[12] Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. TRL: Transformer Reinforcement Learning, 2020. URL `https://github.com/huggingface/trl`. Publication Title: GitHub repository.

[13] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. *Advances in Neural Information Processing Systems*, 36:11809–11822, December 2023. URL `https://proceedings.neurips.cc/paper_files/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract-Conference.html`.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL `https://aclanthology.org/N19-1423/`.

[15] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. Technical report, OpenAI, 2018. URL `https://openai.com/research/language-unsupervised`.

[16] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In *BlackboxNLP@EMNLP*, 2018. URL `https://api.semanticscholar.org/CorpusID:5034059`.

[17] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In Jian Su, Kevin Duh, and Xavier Carreras, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL `https://aclanthology.org/D16-1264/`.

[18] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, and et al. Constitutional AI: Harmlessness from AI Feedback, December 2022. URL `http://arxiv.org/abs/2212.08073`. arXiv:2212.08073 [cs].

[19] Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J. Liu, James Harrison, Jaehoon Lee, and et al. Beyond Human Data: Scaling Self-Training for Problem-Solving with Language Models, April 2024. URL `http://arxiv.org/abs/2312.06585`. arXiv:2312.06585 [cs].

[20] Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. VinePPO: Unlocking RL Potential For LLM Reasoning Through Refined Credit Assignment, October 2024. URL `http://arxiv.org/abs/2410.01679`. arXiv:2410.01679 [cs].

[21] Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. SWE-RL: Advancing LLM Reasoning via Reinforcement Learning on Open Software Evolution, February 2025. URL `http://arxiv.org/abs/2502.18449`. arXiv:2502.18449 [cs].

[22] OpenAI, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, and et al. OpenAI o1 System Card, December 2024. URL `http://arxiv.org/abs/2412.16720`. arXiv:2412.16720 [cs].

[23] Jiayi Pan. Countdown-Tasks-3to4, 2024. URL `https://huggingface.co/datasets/Jiayi-Pan/Countdown-Tasks-3to4`.

[24] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168*, 2021.

[25] Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are NLP Models really able to Solve Simple Math Word Problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.168. URL `https://aclanthology.org/2021.naacl-main.168`.

A Countdown Task

For example, given the numbers `[19, 36, 55, 7]` and a target of `65`, a valid solution would be:

```
(55 + 36) - (19 + 7) = 65
```

B Prompt

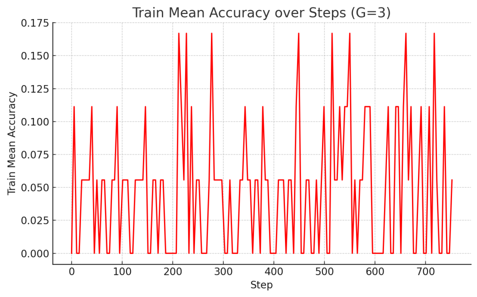
For base models, countdown tasks are formatted using the following prompt:

```
 A conversation between User and Assistant. The user asks a question, and
the Assistant solves it. The assistant first thinks about the reasoning
process in the mind and then provides the user with the answer. User:
Using the numbers {}, create an equation that equals {}. You can use basic
arithmetic operations (+, -, *, /) and each number can only be used once.
Show your work in <think> </think> tags. And return the final answer
in <answer> </answer> tags, for example <answer> (1 + 2) / 3 </answer>.
Assistant: Let me solve this step by step. <think>
```

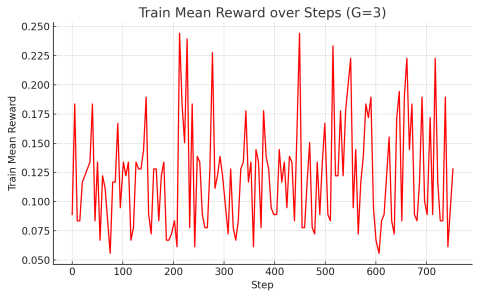
For instruction-tuned models, countdown tasks are formatted with the following prompt:

```
 <|im_start|>system You are a helpful assistant. You first thinks about
the reasoning process in the mind and then provides the user with the
answer.<|im_end|> <|im_start|>user Using the numbers {}, create an equation
that equals {}. You can use basic arithmetic operations (+, -, *, /) and
each number can only be used once. Show your work in <think> </think>
tags. And return the final answer in <answer> </answer> tags, for example
<answer> (1 + 2) / 3 </answer>.<|im_end|> <|im_start|>assistant Let me
solve this step by step.<think>
```

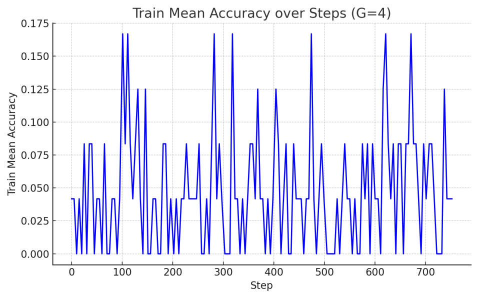
C Training Curves



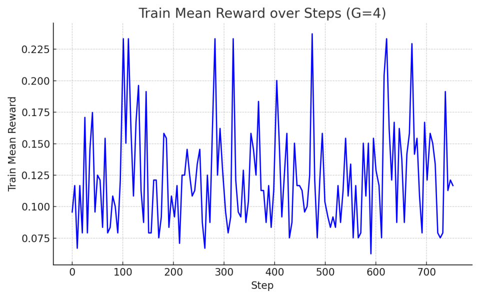
(a) Train Accuracy with Group size 3



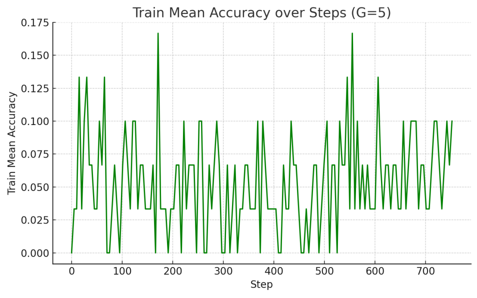
(b) Train Reward with Group size 3



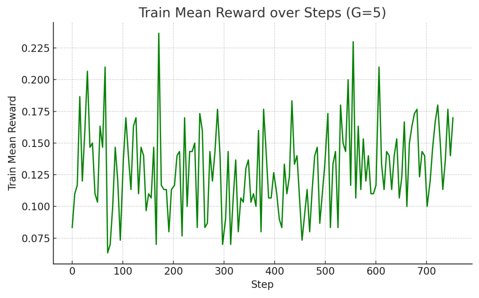
(c) Train Accuracy with Group size 4



(d) Train Reward with Group size 4



(e) Train Accuracy with Group size 5



(f) Train Reward with Group size 5

Figure 2: Train Accuracy and Rewards for Different Group Sizes (3, 4 and 5)