
Improving GRPO Data Efficiency with Budget Forcing

Jonah Mackey
University of Toronto, Vector Institute
jonahmackey@cs.toronto.edu

Yuchong Zhang
University of Toronto, Vector Institute
yuchongz.zhang@mail.utoronto.ca

Abstract

Reinforcement learning (RL) fine-tuning has proved to be highly effective at inducing strong reasoning capabilities of large language models. However, it is data intensive, requiring large amounts of high-quality data, which is often difficult to collect. We attempt to improve the data efficiency of RL fine-tuning by *budget forcing*: forcing the model to use a minimum number of tokens during the RL sampling process. The idea is that by forcing the model to spend more tokens during thinking, it will generate more correct responses, leading to more training signal per reasoning problem. We test this idea on two base models and two reasoning tasks. We find that although budget forcing leads to more reward signal in the early stages of training, the advantage diminishes and the method ultimately under-performs compared to the baseline. Finally, we analyze why our budget forcing algorithm fails. We release our code at <https://github.com/jbxnes/deepseek-s1>.

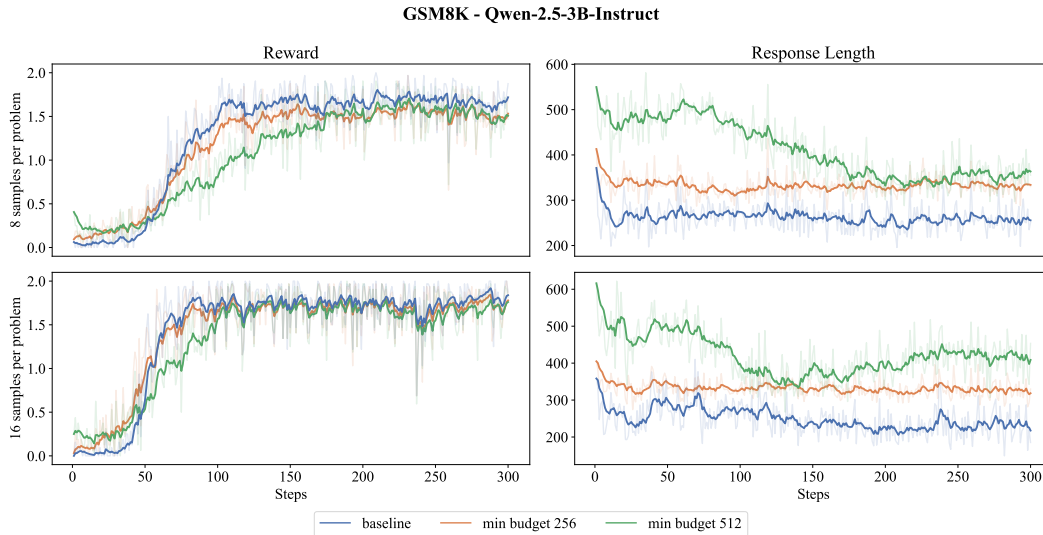


Figure 1: Budget forcing leads to higher reward signal in the early stages of GRPO training. However, these benefits diminish as training progresses and the model underperforms in the long run compared to the baseline.

1 Introduction

DeepSeek-R1 demonstrated that reinforcement learning (RL) is effective in inducing strong reasoning capabilities of large language models (LLMs), but the process is data-intensive DeepSeek-AI et al. [2025]. Although DeepSeek did not open-source their data, community efforts to reproduce R1

suggest that at least hundreds of thousands of high-quality data samples are needed Face [2025]. Since gathering high-quality data is often difficult and costly, improving the data efficiency of RL fine-tuning is an important problem: we want to achieve the same level of reasoning capabilities using less data.

DeepSeek uses a rule-based correctness reward in their RL sampling process. For each reasoning problem, multiple answers are sampled from the model. A correct answer receives a reward of 1, otherwise the reward for that answer is 0. DeepSeek-AI et al. [2025]. Under this premise, the amount of training signal the model receives from each problem is determined by the number of correct generations. Thus, a natural idea for improving data efficiency is to increase the number of correct generations by intervening the model’s generation process. Such interventions might increase the computational cost of the generation process, but since data is often more tightly constrained than computation in RL fine-tuning Foster et al. [2025], a method that achieves higher data efficiency at the cost of more computation can still be useful.

In this work, we investigate whether applying *budget forcing* during RL sampling improves data efficiency. Formulated in Muennighoff et al. [2025], budget forcing is an intervention that ensures that the number of tokens generated by the model is at least n and at most N for some desired budgets n and N . We consider a less restricted form of budget forcing to ensure that in most responses, the model uses at least n tokens on thinking. Specifically, if the model terminates thinking before reaching the desired budget, then we append the token "Wait" to the tokens generated so far, and feed the result back into the model to continue generation. This process is repeated as necessary until the desired minimum budget is reached.

We apply budget forcing to sample responses during RL fine-tuning while keeping the other components of the pipeline unchanged. We experiment with different choices of the minimum budget n . We consider two models and two tasks: the Qwen-2.5-3B/7B-Instruct models Qwen et al. [2025] and the GSM8K Cobbe et al. [2021] and MATH (levels 3-5) Hendrycks et al. [2021] datasets.

In our experiments, we observe that budget forcing leads to higher training reward in the very early stage of training, indicating that the model gets more training signals from the first few data points. However, this advantage quickly diminishes as training progresses and the model sees more data. Eventually, budget forcing leads to both lower training reward in the later stage of training and worse test performance.

This article is organized as follows:

- In Section 2, we briefly survey related works.
- In Section 3, we describe GRPO Shao et al. [2024], the RL algorithm we work with.
- In Section 4, we provide a formal description of our budget forcing method. We define our hypothesis that it improves data efficiency and describe our framework for testing it.
- In Section 5, we present the details of our experimental setup and results. We analyze our results to conclude that our method fails to improve data efficiency in a meaningful way.
- In Section 6, we discuss why our method fails and potential future directions.
- In Section 7, we summarize the main conclusions of our work.

2 Related Work

RL fine-tuning for reasoning: While RL has been used to align LLMs with human preferences Ouyang et al. [2022] and training models to self-correct its reasoning mistakes Kumar et al. [2024], DeepSeek-R1 is the first open research achieving on-par performance with OpenAI’s o1 models using pure RL fine-tuning DeepSeek-AI et al. [2025]. There are also theoretical results suggesting that verifier-based RL fine-tuning is superior to verifier-free supervised fine-tuning (SFT) given a fixed data and compute budget Setlur et al. [2025]. Many works since the release of R1 study the data and base model requirements for effective RL fine-tuning Zeng et al. [2025] Gandhi et al. [2025]. DeepSeek-R1 uses the Group Relative Policy Optimization (GRPO) algorithm proposed in Shao et al. [2024], and some works propose alternative RL optimization algorithms to GRPO Chu et al. [2025]. In this work, we fix the algorithm to be GRPO and choose a data/base model combination where RL fine-tuning already works well-enough to investigate whether budget forcing improves the training data efficiency.

Budget forcing: Budget forcing is introduced in Muennighoff et al. [2025] as a method for scaling up inference-time compute. Specifically, they first apply SFT to a pre-trained LLM on a small high-quality reasoning dataset. Next, given a desired minimum budget n and maximum budget N , if the model terminates before generating at least n tokens, they use the word "Wait" to replace the end of thinking token so the model continues generation. If the model generates more than N tokens, then the generation is forcefully terminated. This method has shown to significantly improve test performance as the budget scales up, allowing a 32B base model to achieve the same level of performance as OpenAI’s o1-preview Muennighoff et al. [2025]. This makes budget forcing an attractive candidate for increasing training signal during RL. In this work, we adapt budget forcing and apply it during RL training rather than during inference. We only enforce a minimum budget and not a maximum budget. Further, instead of replacing the end of thinking token with "Wait", we append "Wait" after it.

3 Background

In this work, we adopt the Group Relative Policy Optimization (GRPO) algorithm used in DeepSeek-AI et al. [2025]. Unlike proximal policy optimization (PPO), GRPO does not require a critic model which is typically the same size as the base model. Instead, it estimates the advantage from group-normalized scores, allowing for less memory costs. For each reasoning problem q , GRPO samples a group of outputs $\{o_1, o_2, \dots, o_G\}$ from the old base model $\pi_{\theta_{old}}$ and optimizes the model π_{θ} by maximizing the following objective:

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}_{\substack{q \sim P(Q) \\ \{o_i\}_{i=1}^G \sim \pi_{\theta_{old}}(O|q)}} \left[\frac{1}{G} \sum_{i=1}^G \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \left\{ \min \left(\frac{\pi_{\theta}(o_{i,t}|q, o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t}|q, o_{i,<t})} \hat{A}_{i,t}, \text{clip} \left(\frac{\pi_{\theta}(o_{i,t}|q, o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t}|q, o_{i,<t})}, 1 - \epsilon, 1 + \epsilon \right) \hat{A}_{i,t} \right) - \beta \mathbb{D}_{KL}[\pi_{\theta} \parallel \pi_{ref}] \right\} \right], \quad (1)$$

$$\mathbb{D}_{KL}[\pi_{\theta} \parallel \pi_{ref}] = \frac{\pi_{\theta_{ref}}(o_i|q)}{\pi_{\theta_{old}}(o_i|q)} - \log \frac{\pi_{\theta_{ref}}(o_i|q)}{\pi_{\theta_{old}}(o_i|q)} - 1, \quad (2)$$

where ϵ and β are hyper-parameters. A_i is the advantage computed using a group of rewards $\{r_1, r_2, \dots, r_G\}$, where each r_i is a rule-based correctness reward computed using o_i . For example, we can use a verifier on o_i , and return $r_i = 1$ if o_i is correct, and $r_i = 0$ otherwise.

$$A_i = \frac{r_i - \text{mean}(\{r_1, r_2, \dots, r_G\})}{\text{std}(\{r_1, r_2, \dots, r_G\})} \quad (3)$$

4 Method

The main hypothesis we want to test is that budget-forced generation leads to more correct responses per reasoning problem and gives the model more training signal during RL fine-tuning, allowing it to reach the same level of performance using less data.

Our hypothesis comes from the observation from Muennighoff et al. [2025] that applying budget forcing during inference increases performance and often leads the model to correct its own faulty reasoning steps. Thus, we hypothesize that applying budget forcing during GRPO sampling leads to more training signal and higher sample efficiency. Intuitively, budget forcing makes the model think "harder" in each response, so each data point is "better utilized" compared to normal generation.

Due to resource constraints, in contrast to s1 Muennighoff et al. [2025], we apply a non-strict form of budget forcing. First, we use a system prompt to instruct the model to put its reasoning trace between "<think>" and "</think>" (see Appendix C for the full system prompt). Next, we fix a desired budget n , representing the (soft) minimum number of tokens we want the

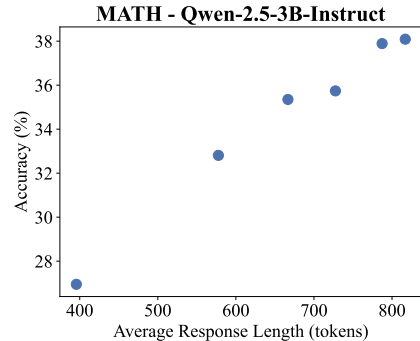


Figure 2: Both average response length and accuracy scale with minimum budget, showing that budget forcing leads to more correct responses per problem.

model to use for thinking. We let the model generate its response token by token for a given prompt, and whenever the generation so far ends with a "`</think>`" tag and the number of tokens generated so far is less than n , we append the word "Wait" to the end of the generation, and feed the generation back into the model to force it to keep thinking. An example of the budget-forced generation produced during training is illustrated in Figure 3 (see Appendix B for other example model outputs). Formally, the algorithm is described in algorithm 1.

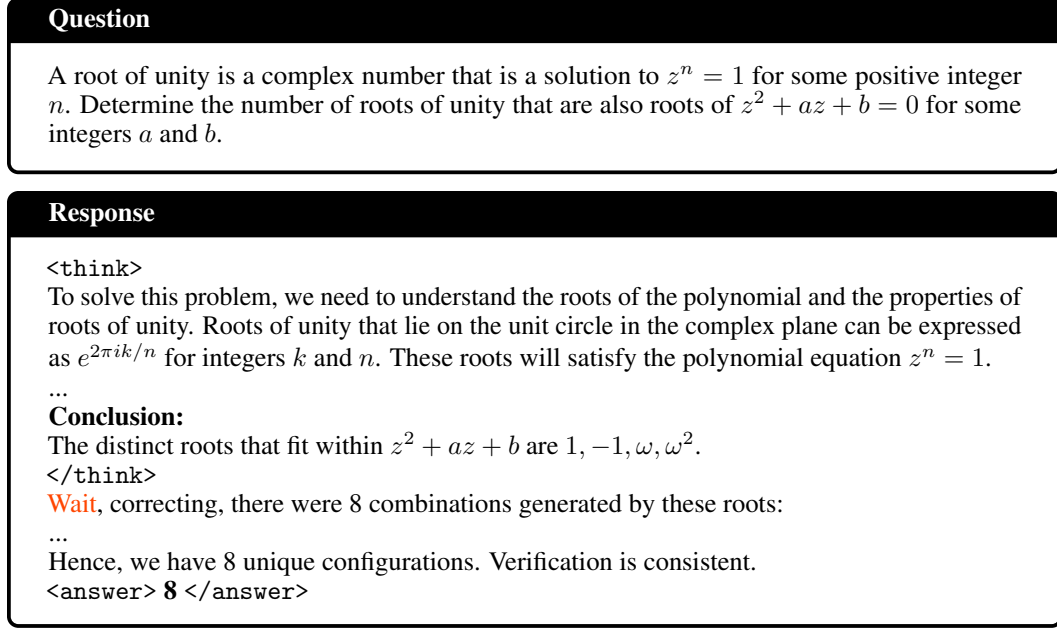


Figure 3: **Budget forcing example.** We add the word "Wait" after `</think>` to force the model to think more.

Algorithm 1 Budget-Forced-Generation

Require: Prompt x , system prompt S , budget n , language model M

```

1: Prepend  $S$  to  $x$ 
2:  $g \leftarrow$  empty string
3: while model has not terminated generation do
4:    $t \leftarrow M.\text{next\_token}(x + g)$ 
5:   Append  $t$  to  $g$ 
6:   if  $g$  ends with </think> and  $\text{len}(g) < n$  then
7:     Append " Wait" to  $g$ 
8:   end if
9: end while
10: return  $g$ 
```

To validate that our non-strict budget forcing can still lead to more correct responses, we compute the accuracy of the Qwen-2.5-3B-Instruct model on 32 MATH (level 3-5) problems, evaluated over 16 responses per problem. We do budget forcing for each n in $\{0, 512, 640, 768, 896, 1024\}$ and observe that as n increases, both the average response length and accuracy increase, as shown in Figure 2.

Although the idea of budget forcing is to ensure that the model uses at least n tokens for thinking, our algorithm does not always guarantee this. For example, the model could fail to output a `</think>` token. Even after "Wait" is appended once, the model might not output more `</think>` tokens in the future, and our algorithm would fail to enforce the budget. However, we choose this algorithm because it admits an implementation through the `LogitsProcessor` class of the `vLLM` library Kwon et al. [2023] that is fast enough for sampling model outputs during GRPO. In comparison, the budget forcing algorithm used in Muennighoff et al. [2025] decodes model outputs into text and

does string-level replacement, which cannot be done efficiently over batches of outputs. In addition, we found that although the guarantee is not strict, the algorithm enforces the budget well enough in practice. This is shown by the average response lengths of the model during training in Section 5. We discuss the potential limitations of our approach more in Section 6.

GSM8K			
min budget	responses-per-problem	Qwen-2.5-3B-Instruct	Qwen-2.5-7B-Instruct
0 (baseline)	8	77.6%	84.1%
256	8	67.9%	39.80%
512	8	67.9%	37.1%
0 (baseline)	16	82.0%	84.2%
256	16	79.0%	48.5%
512	16	72.6%	49.3%

Table 1: Final test accuracies of Qwen-2.5-3B/7B Instruct models on GSM8K at various min token budgets and responses-per-problem.

To test our hypothesis, we run two sets of experiments:

1. We run GRPO training where the model responses are generated normally. This will be the baseline to compare against.
2. We run GRPO training where the model responses are generated via budget-forced generation (Algorithm 1) while keeping everything else the same as the baseline above. We keep the budget n fixed throughout any particular run and experiment with various choices of n across different runs.

We only apply budget forcing during RL training, not during inference. For both sets of experiments, the key metrics and measurements we focus on are:

- **Reward throughout training.** If our hypothesis is correct, we expect to see models trained with budget forcing to reach a certain reward using less data. We also expect that after the same number of training steps, models trained with budget forcing achieves higher reward.
- **Final testing performance.** Since budget forcing is not applied during inference, we use test performance to evaluate whether the models generalize to the test-time environment.
- **Average response length.** Because our algorithm does not always strictly enforce the budget, we monitor the response lengths during training.

5 Experiments

5.1 Setup

In our experiments, we aim to evaluate the effect of budget forcing during GRPO training. To do this, we consider various base model sizes, reasoning tasks, minimum token budgets, and responses-per-problem. Our experiments were designed to test our hypotheses discussed in Section 4 and answer the following questions:

- Does budget forcing improve GRPO training? We hypothesize the benefits of budget forcing during generation, i.e., more correct responses per problem, may lead to improved sample efficiency, higher reward signal, and faster convergence during RL training.
- Does our method generalize at test time? Budget forcing is applied during GRPO training but not at test time, thus the train and test RL environment is different. As such, the model may learn to rely on budget forcing during training and underperform at test time.

Dataset: To understand how our method performs in different task settings, we considered reasoning tasks of various difficulty. In particular, we chose two commonly used datasets for training and evaluation: GSM8K Cobbe et al. [2021] and MATH (level 3-5) Hendrycks et al. [2021]. Following

Zeng et al. [2025], we categorize the GSM8K and MATH (level 3-5) datasets as *easy* and *hard*, respectively.

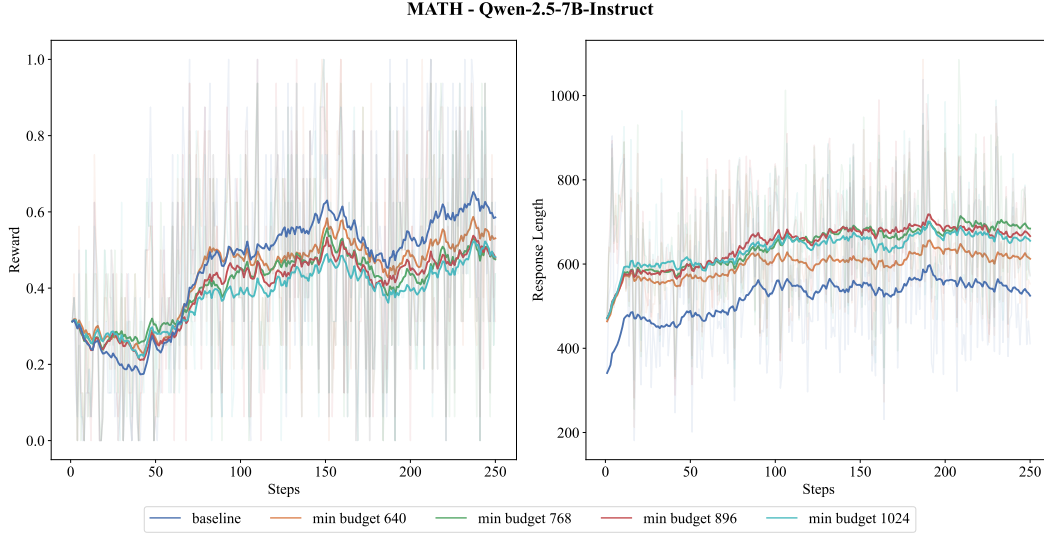


Figure 4: Reward and response length of Qwen-2.5-7B-Instruct during training on MATH at various minimum token budgets.

Reward: We chose to use a rule-based reward function based purely on correctness, giving 2.0/1.0 reward (GSM8K/MATH) for a correct response and 0.0 otherwise. We chose not to apply any formatting based reward as it has been shown to penalize exploration Zeng et al. [2025]. As well, we thought formatting reward would complicate our analysis as our method utilizes formatting related tokens (i.e., `</think>`) during generation.

Models: We conducted experiments on the Qwen-2.5-3B/7B Instruct models Qwen et al. [2025] to investigate the effect of our method at different base model scales. Due to compute constraints, we use LoRA Hu et al. [2021] with rank 64 to train all of our models.

Training Setup: For GSM8K, we trained our models for 300 steps and considered token budgets {0, 256, 512} and responses-per-problem {8, 16}. For MATH, we trained our models for 250 steps and considered token budgets {0, 640, 768, 896, 1024}. The token budgets were selected to all be above the corresponding baseline models’ (min budget 0) average response length. Due to compute constraints, we used a fixed batch size of 16 for all experiments (this includes all samples per problem), so the models see a total of 600/300 problems (with responses-per-problem 8/16) for GSM8K and 500 problems for MATH. Finally, we used 4 gradient accumulation steps in GSM8K experiments and 1 gradient accumulation step for MATH experiments.

MATH		
min budget	Qwen-2.5-3B-Instruct	Qwen-2.5-7B-Instruct
0 (baseline)	52.2%	50.30%
640	44.2%	40.6%
768	51.6%	37.4%
896	42.7%	38.0%
1034	49.7%	37.80%

Table 2: Final test accuracies of Qwen-2.5-3B/7B Instruct models on MATH at various min token budgets.

Evaluation: We applied the same evaluation scheme in all experiment configurations. During training, we monitored reward and average response length (in tokens). We also report final test accuracies on a held-out dataset of 1.32k/1k problems (GSM8K/MATH). We used standard GRPO training without budget forcing for comparison as a baseline.

5.2 Results

Does budget forcing improve GRPO training?: Reward training curves are shown in Figures 1 and 4 (also see Figures 5 and 6 in Appendix A). We observed a fairly consistent pattern across all our experiments: GRPO with budget forcing typically achieves higher rewards early in training compared to the baseline. However, these initial gains diminish over time, and the model underperforms in the long run. Additionally, models with larger token budgets show a higher initial reward but also experience a greater decline in performance as training progresses. This suggests a trade-off inherent in our method—while higher budgets provide a stronger early reward signal, they tend to impair long-term model performance.

Do models trained with budget forcing generalize to test-time?: As shown in Tables 1 and 2, models trained with budget forcing consistently show lower test accuracy compared to the baseline. In most cases, this drop in test accuracy aligns with the corresponding decline in reward during training, suggesting that budget forcing does generalize to test-time. However, we observed an exception with Qwen-2.5-7B-Instruct on GSM8K. As shown in Figure 5 in Appendix A, models trained with budget forcing achieve high reward during training, competitive with the baseline. Despite this strong training performance, they show a significant drop in test accuracy, as seen in Table 1.

6 Limitations

There are several limitations of our budget forcing method in its current formulation. The most significant limitation is compute. RL fine-tuning methods such as GRPO Shao et al. [2024] require large amounts of compute. As such, existing works in the LLM reasoning literature typically work at a larger scale compared to our setup. This includes larger models, larger batch sizes, longer response lengths, and longer training runs. In this project, we were limited to a few NVIDIA A40 GPUs and two NVIDIA A100 GPUs with long waiting times and limited run times. Due to these restrictions, we had to make several sacrifices, such as: relatively smaller models, shorter response lengths, small batch sizes, shorter training runs, and using LoRA Hu et al. [2021] to reduce memory overhead during training. We believe the setting we evaluated our method in, due to the compute restrictions, was not the most suitable.

While budget forcing has shown to be successful during test-time Muennighoff et al. [2025], it is still uncertain if it can be successful during GRPO training. The responses generated from our budget forcing algorithm have an irregular structure, and training on these responses could harm model performance. To keep our approach simple, we resorted to appending the “Wait” token whenever the `</think>` token is generated (assuming we are below budget). We believe this approach can harm model performance because it is possible the model can detect when our budget forcing intervention occurs, thus allowing the model rely on it during training. We believe *replacing* the `</think>` token with “Wait” instead, as done Muennighoff et al. [2025], could address this.

In our method, budget forcing is applied during training but not at test-time. The model may learn to rely on the budget forcing during training and under perform at test-time because the environment is different. While we did discuss this in Section 5.2, we believe our experiments were not entirely conclusive due to the compute constraints.

On easier tasks, the base model may already have sufficiently high reward signal. Thus, applying budget forcing may do more harm than good. We believe our approach is more suitable for difficult tasks, where the base model has low reward signal. We include some example outputs in Appendix A showing that budget forcing may not help with reasoning when the output is already correct: the model simply repeats itself.

7 Conclusion

In conclusion, our budget forcing method fails to improve the data efficiency of GRPO. In our experiments, while budget forcing produces more reward signal during the early stages of training, this advantage diminishes as the model sees more data and budget forcing ultimately performs worse than normal sampling. However, because budget forcing does lead to more signal early on during training, we believe that for a right model-task combination, the advantage could be magnified. Moreover, we believe the idea of using more tokens during thinking to increase training signal has

potential, and a method that does not rely heavily on specific format structures of the model outputs could be more successful.

Contributions

Both team members contributed equally to developing, debugging, and implementing the method and code repository. Jonah Mackey executed and managed the experiments on the GSM8K dataset, and Yuchong Zhang executed and managed the experiments on the MATH dataset. Both team members contributed equally to writing the report.

Acknowledgments

We thank Chris Maddison and Haonan Duan for their helpful feedback and stimulating discussions. This research was enabled by computing resources at the Vector Institute (<https://vectorinstitute.ai/>). Our code repository was built on top of a GRPO reference implementation developed by Jacob Junqi Tian and John Willes for the Vector Institute Researchers (<https://github.com/VectorInstitute/vector-trl-references>).

References

- Xiangxiang Chu, Hailang Huang, Xiao Zhang, Fei Wei, and Yong Wang. Gpg: A simple and strong reinforcement learning baseline for model reasoning, 2025. URL <https://arxiv.org/abs/2504.02546>.
- 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, 2021. URL <https://arxiv.org/abs/2110.14168>.
- DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojuan Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-rl: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL <https://arxiv.org/abs/2501.12948>.

- Hugging Face. Open r1: A fully open reproduction of deepseek-r1, January 2025. URL <https://github.com/huggingface/open-r1>.
- Dylan J. Foster, Zakaria Mhammedi, and Dhruv Rohatgi. Is a good foundation necessary for efficient reinforcement learning? the computational role of the base model in exploration, 2025. URL <https://arxiv.org/abs/2503.07453>.
- Kanishk Gandhi, Ayush Chakravarthy, Anikait Singh, Nathan Lile, and Noah D. Goodman. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective stars, 2025. URL <https://arxiv.org/abs/2503.01307>.
- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021. URL <https://arxiv.org/abs/2103.03874>.
- Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL <https://arxiv.org/abs/2106.09685>.
- Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. Training language models to self-correct via reinforcement learning, 2024. URL <https://arxiv.org/abs/2409.12917>.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.
- Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. sl: Simple test-time scaling, 2025. URL <https://arxiv.org/abs/2501.19393>.
- Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022. URL <https://arxiv.org/abs/2203.02155>.
- Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL <https://arxiv.org/abs/2412.15115>.
- Amrith Setlur, Nived Rajaraman, Sergey Levine, and Aviral Kumar. Scaling test-time compute without verification or rl is suboptimal, 2025. URL <https://arxiv.org/abs/2502.12118>.
- 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, 2024. URL <https://arxiv.org/abs/2402.03300>.
- Weihaoh Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. Simplerl-zoo: Investigating and taming zero reinforcement learning for open base models in the wild, 2025. URL <https://arxiv.org/abs/2503.18892>.

Appendix

A Additional Experimental Results

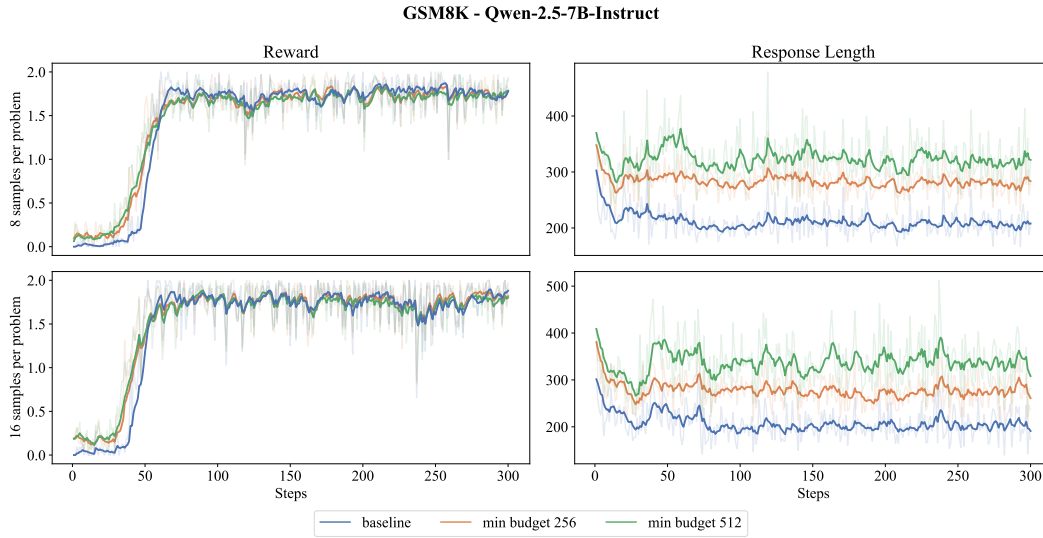


Figure 5: Reward and response length of Qwen-2.5-7B-Instruct during training on GSM8K at various minimum token budgets and responses-per-problem.

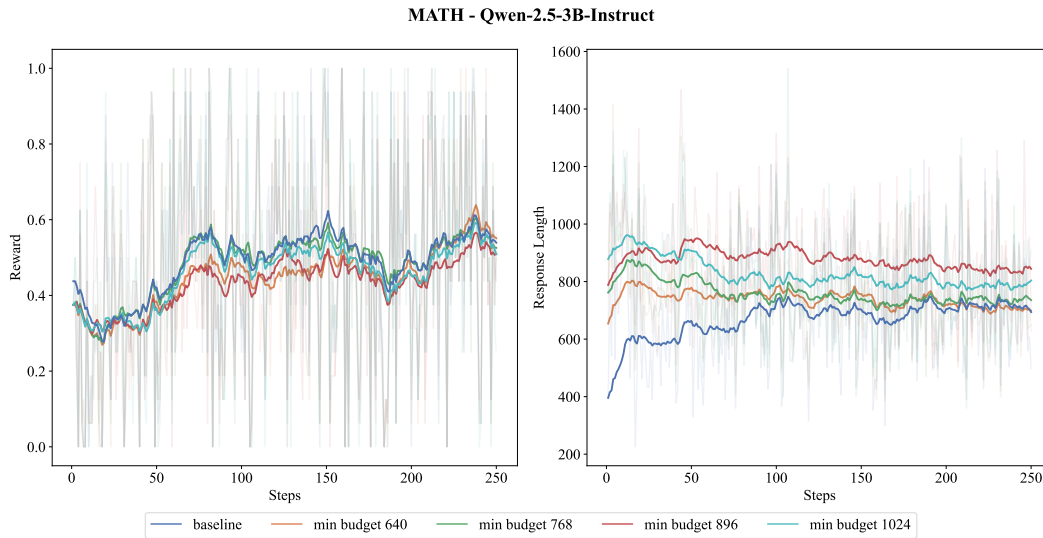


Figure 6: Reward and response length of Qwen-2.5-3B-Instruct during training on MATH at various minimum token budgets.

B Example Outputs of Budget Forcing During GRPO Training.

Question

Three local dance studios have 376 students. The first two have 110 and 135 students. How many students does the third studio have?

Response

<think>

To find out how many students the third studio has, we need to subtract the number of students in the first two studios from the total number of students.

...

The third studio has 131 students.

</think>

Wait, I realize I have the correct approach but made a slight error in my mental addition and subtraction. Let's recheck:

The total number of students in the first two studios adding up is actually 110 plus 135, which is 245.

Now, subtracting this from the total number of students, 376, gives:
 $376 - 245 = 131$.

Therefore, the third studio has indeed 131 students.
</think>

Wait, I'll double-check the calculations just to be sure:
 $110 + 135$ is correctly 245.

When we subtract 245 from 376, we get:
 $376 - 245 = 131$.

The third studio indeed has 131 students.
</think>

Wait a moment, I want to confirm this again with final clarity to avoid any doubt. The sum of the students in the first and second studios is $110 + 135 = 245$. From the total, 376, subtracting 245:
 $376 - 245 = 131$.

So, finally:

The third studio has 131 students.
</think>

Wait, after rechecking—again, the sum of the students in the first two studios is $110 + 135 = 245$. Subtract 245 from the total 376 leads to:
 $376 - 245 = 131$.

After every verification, I can confidently state that the third studio has 131 students.
</think>

The calculation is right, and I've verified it multiple times.
The third studio has 131 students.
</think>

<answer>
131

</answer>

Figure 7: Example output of Qwen-2.5-7B-Instruct during training on GSM8K (min budget = 512). "Wait" is appended multiple times but the model does not modify its already correct answer.

Question

Violet has 3 more than twice as many nails as Tickletoe. If Violet has 27 nails, how many nails do they have together?

Response

<think> We are given that Violet has 3 more than twice as many nails as Tickletoe, and Violet has 27 nails in total. Let's first find out how many nails Tickletoe has, then we'll calculate the total number of nails they have together.

First, let N represent the number of nails Tickletoe has. Therefore, Violet's nails can be described by the equation:

$$2N + 3 = 27$$

We solve for N :

$$2N = 27 - 3 \quad 2N = 24 \quad N = 12$$

So, Tickletoe has 12 nails. To find out how many nails they have together, we add Violet's and Tickletoe's nails.

$$27(\text{Violet}) + 12(\text{Tickletoe}) = 39$$

Thus, the total number of nails is:

$$\text{Total nails} = 27 + 12$$

</think> Wait, my calculation seems to be missing the steps to correctly interpret the total. Let's clearly add the quantities:

$$\text{Tickletoe's nails} = 12 \quad \text{Violet's nails} = 27 \quad \text{Total nails} = 12 + 27$$

</think> Wait, my previous thought process correctly identified Tickletoe's nails but I incorrectly simplified the total. Let's recalculate:

Tickletoe's nails: 12 Violet's nails: 27

Total nails together: $12 + 27$

</think> Wait, I should have clearly added the totals:

$$12 + 27 = 39$$

The total number of nails they have together is 39.

</think> Wait, I need to be sure. I should double-check the steps.

Tickletoe's nails: 12 Violet's nails: 27

Add them: $12 + 27 = 39$

Indeed, the correct total of nails they have together is 39.

</think> Wait, I should ensure that everything is correct.

Tickletoe: $27 - 3 = 24/2 = 12$ nails Violet: 27 nails

Combining: $12 + 27 = 39$

That's the correct total.

</think> Final confirmation: Violet: 27 nails, Tickletoe: 12 nails.

$$27 + 12 = 39$$

</think> The correct total is 39.

</think> The total nails they have together is 39.

</think> The total number of nails they have together is 39.

</think> The solution is 39.

</think> The answer is clear and correct.

</think> The total number of nails they have together is 39.

Response (continued)

</think> The correct and final answer is 39.
</think> Therefore, the total number of nails they have together is:

39

</think> The answer is:
The total nails they have together is 39.
</think> The solution is 39.
</think> The total nails Violet and Tickletoe have together is:

$\text{Total nails} = 27 + 12 = 39$

</think> The solution is confirmed and the total is:

39

</think> The total number of nails is correctly calculated and the answer is:

39

</think> The correct total of nails Violet and Tickletoe have together is:

39

</think> The correct total, after verifying all steps, is:

39

</think> The solution is verified and the total nails they have together is:

39

</think> We have confirmed all calculations, and the answer is:

39

</think> Let's box the final answer:

39

</think> The final answer is boxed as follows:

39

</think> Finally, the correct and boxed answer is:

39

</think> The boxed and final answer is:

39

</think> The total nails Violet and Tickletoe have together is boxed:

39

</think> The answer is boxed and clear:

39

</answer> <answer> 39 </answer>

Figure 8: **Example output of Qwen-2.5-7B-Instruct during training on GSM8K (min budget = 512).** After several 'Wait' tokens are appended, the model repeats its answer many times.

C System Prompt

System Prompt

Respond in the following format:

<think>

...

</think>

<answer>

...

</answer>