Carnegie Mellon University

Not All Rollouts are Useful: Down-Sampling Rollouts in LLM Reinforcement Learning

Yixuan (Even) Xu
Carnegie Mellon University

Yash Savani
Carnegie Mellon University

Fei Fang
Carnegie Mellon University

Zico Kolter
Carnegie Mellon University

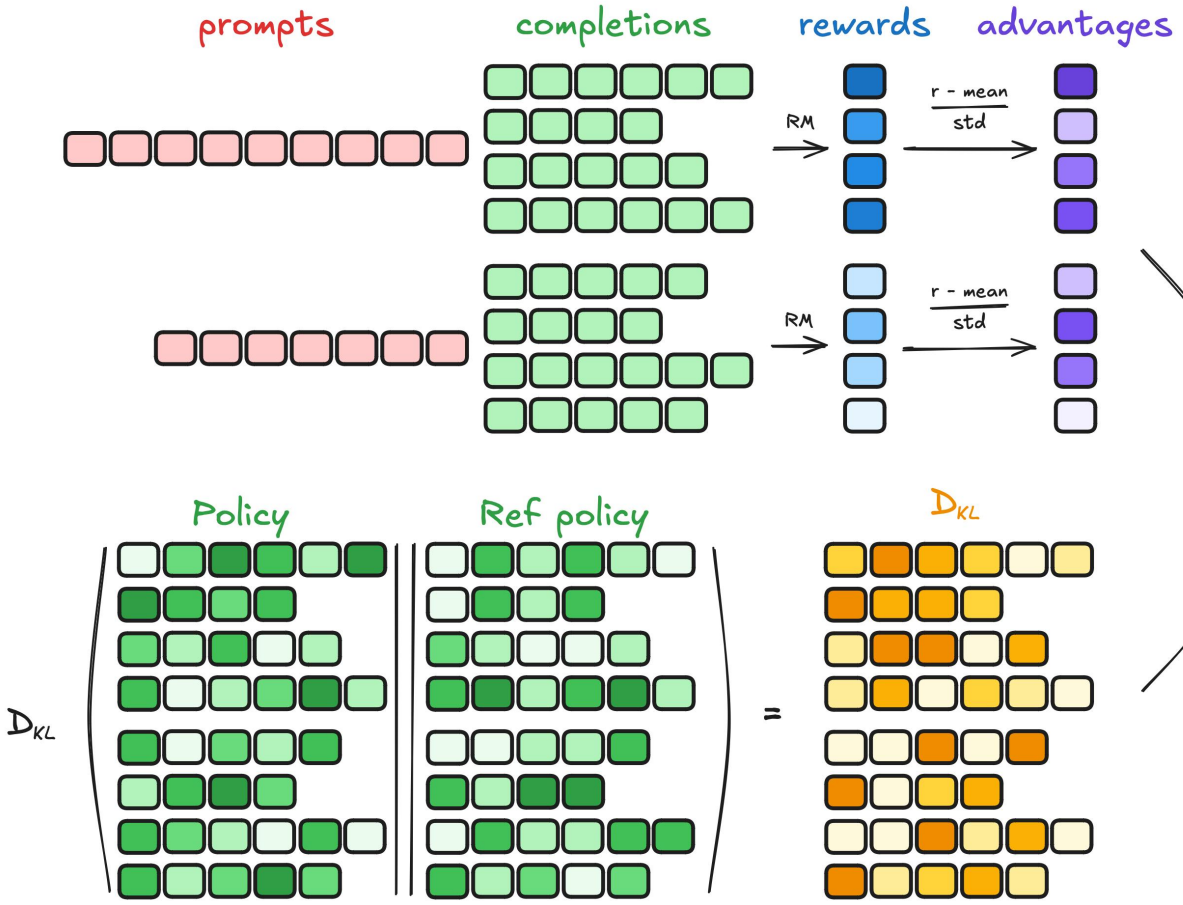
RL with Verifiable Rewards (RLVR)

- **RLVR:** A recent paradigm of improving the **reasoning** capabilities of LLMs, like math, coding, general problem solving

- **RL:** The LLM is trained with reinforcement learning methods
  - Consider the **LLM** as an **agent** whose **action** is outputing **tokens**

- **VR:** Ground truth reward is available (can check correctness)
  - For math with numeric answers, extract and check the final answer
  - For competitive programming, check if the test cases are passed

A Popular RLVR Algorithm: GRPO



- For each prompt
  - Generate completions
  - Reward model scores them
  - Compute advantages
  - Update the model
- A **two-phase** structure: Inference & Policy-update

Figure source: huggingface.co/docs/trl/main/grpo_trainer

GRPO Example

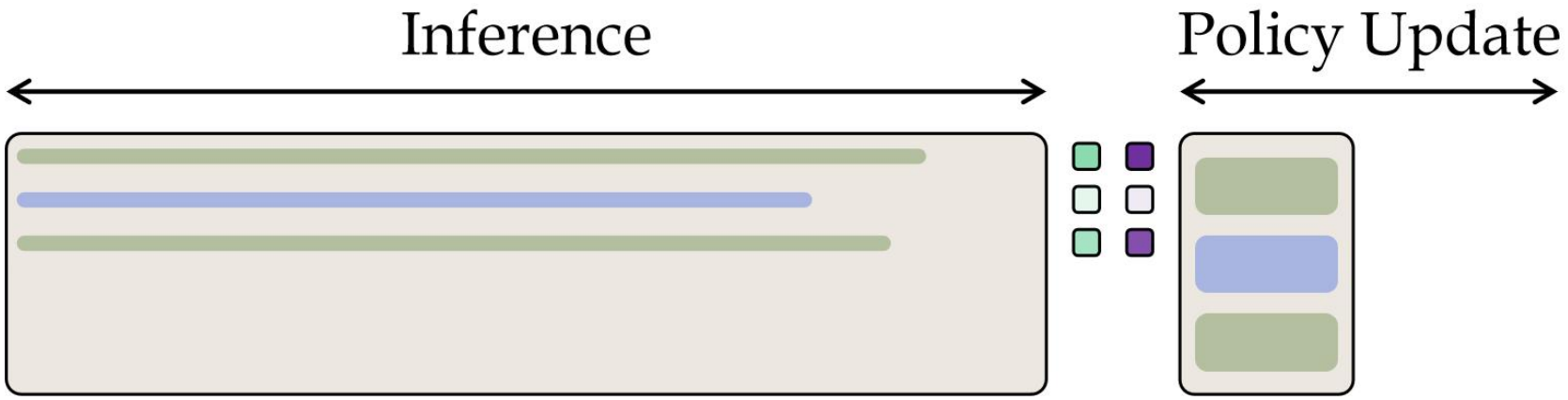
- **Prompt:** What is 3 + 7? Answer in format: “Answer: {result}”

- **Inference:** Generate rollouts & score them
  - **R1.** “Answer: 10.” Score: 80 (answer) + 20 (format)
  - **R2.** “Answer: 0.” Score: 20 (format)
  - **R3.** “The answer is 10.” Score: 80 (answer)

- **Policy Update:** Reinforce the **high-score ones** and penalize the **low-score ones** using policy gradient update rule from RL

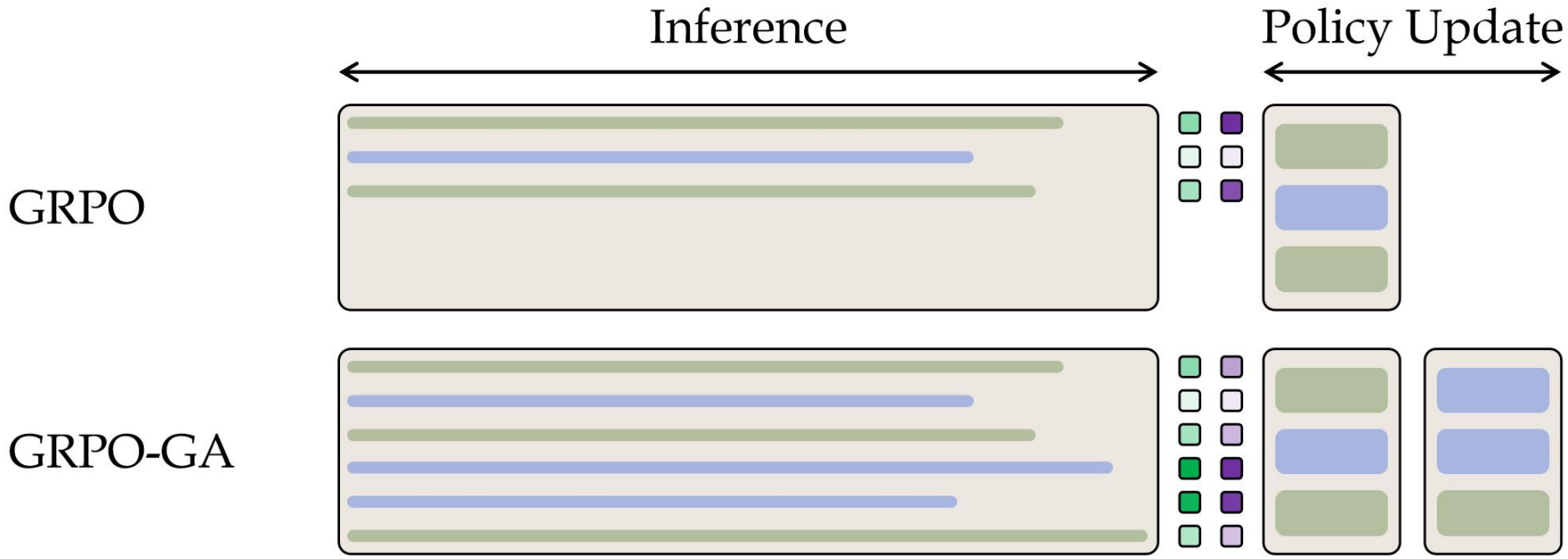
Computation Asymmetry in RLVR

- RLVR algorithms (PPO & GRPO) share a **two-phase** structure:
  - **Inference phase:** Generate rollouts & score them
  - **Policy-update phase:** Update model parameters
  - Computation is **asymmetric** in these two phases
    - Inference is **embarrassingly parallel** and **modest in memory**
    - Policy-update **requires synchronization** and is **intense in memory**

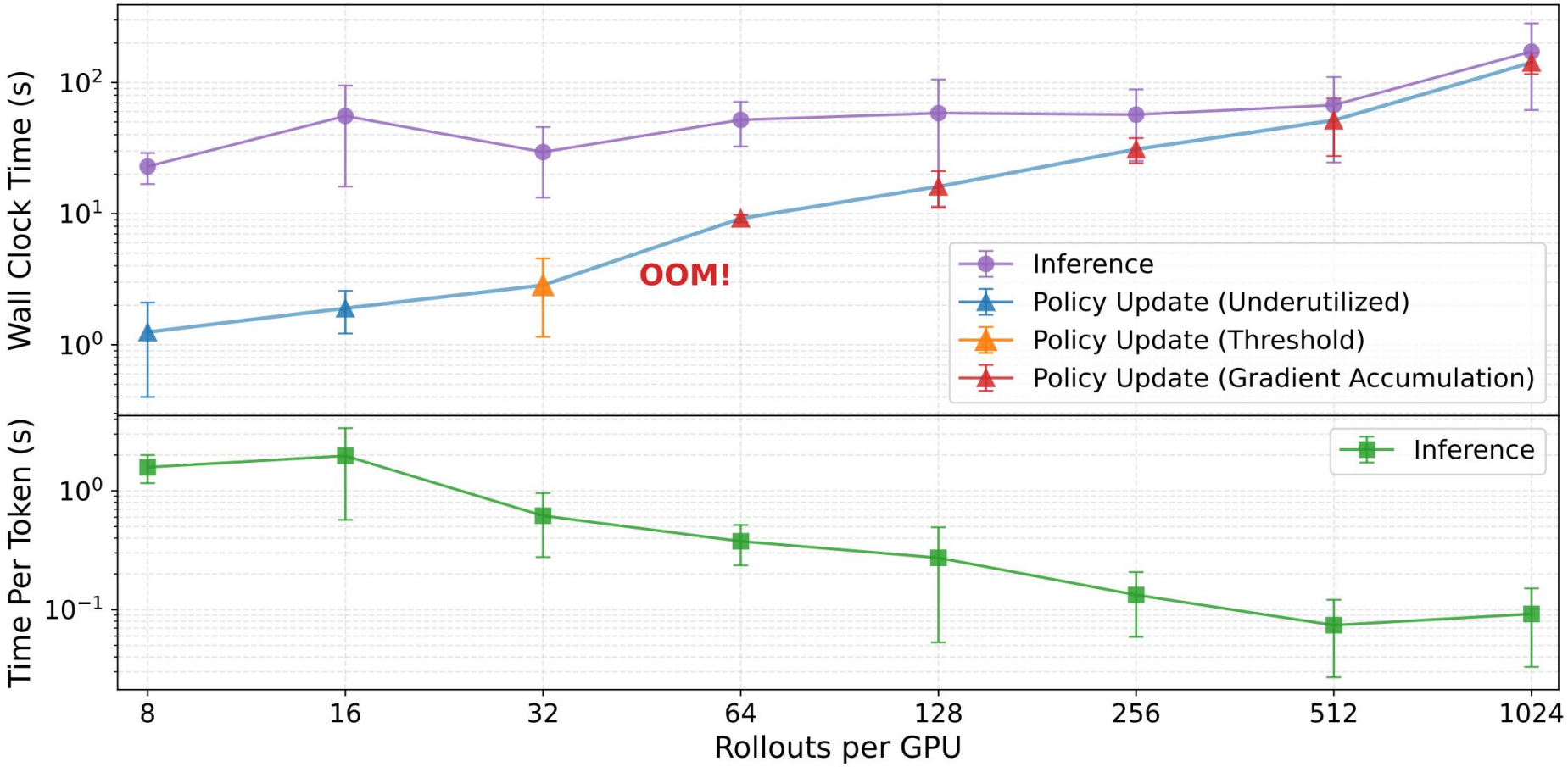


A Solution: Memory-Saving Techniques

- One possible such technique: **gradient accumulation (GA)**
  - Fully utilizes the GPU at inference phase
  - Splits the generated rollouts into multiple policy update steps



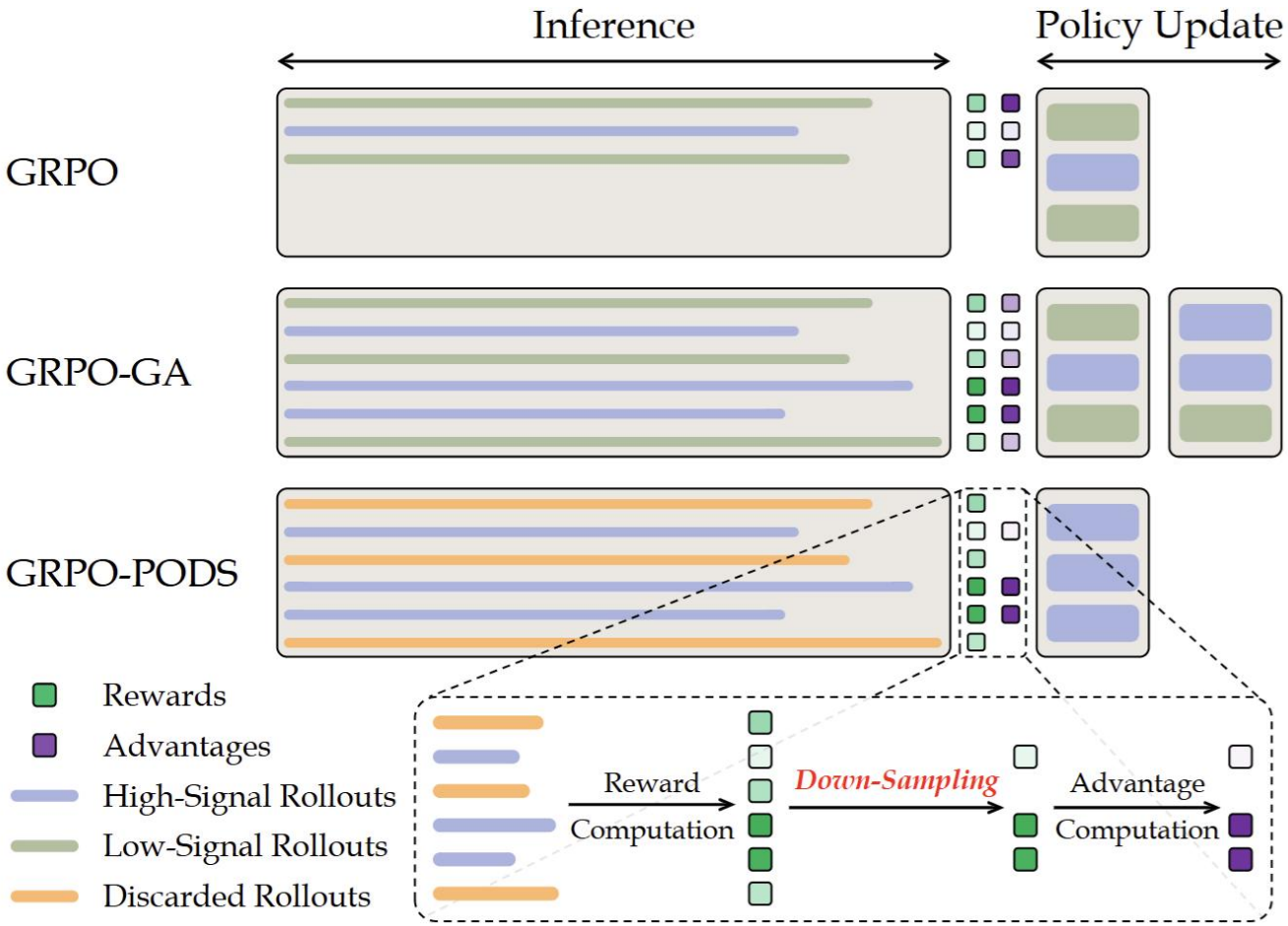
Empirical Timing Breakdown Fine-Tuning Qwen2.5-3B-Instruct on GSM8K & 8 A100s



Policy Optimization with Down-Sampling

- We observe that *"not all rollouts contribute equally to model improvement"* and propose **PODS**

- Unlike GA, we propose to strategically **discard** some of the generated rollouts
  - Addresses the asymmetry
  - Retains comparable or even better learning signals



The PODS framework

- **General framework**
  - Generate n rollouts in inference
  - Down-sample to $m < n$ rollouts for training

- **How to set the down-sampling rule?**
  - Imagine four rollouts with rewards {0.2, 0.4, 0.6, 0.8}
  - If you only want to keep two of them for training, which two?
  - Intuitively, it should be the first one and the last one
    - Because they demonstrate the **best performance** for the model **to learn** and the **worst performance** that the model **should avoid**

Max-Variance Down-Sampling

- **Max-variance down-sampling**
  - Choose the subset of rollouts that **maximizes variance** in rewards
  - **Intuition:** Captures both positive and negative learning signals

- **Theorem 1:** This set contains k highest & $m - k$ lowest rewards
  - Which gives us an $O(n \log n)$ algorithm for computing this set
  - This concurs of the intuition of the example we just saw

- **Theorem 2:** If rewards are binary, then k is always $m/2$

Proof Intuitions

- **Theorem 1:** This set contains k highest & $m - k$ lowest rewards
  - If the set we selected is not of this form, e.g. {**0.2**, 0.5, **0.6**, 0.7, **0.8**}
  - We can always move **one chosen element away** from their mean, while not decreasing variance: {**0.2**, 0.5, 0.6, **0.7**, **0.8**}

- **Theorem 2:** If rewards are binary, then k is always $m/2$
  - This is a natural corollary of Theorem 1
  - Let the number of positive rewards be p
    - Consider (i) $p \leq m/2$, (ii) $p \geq n - m/2$, (iii) $m/2 \leq p \leq n - m/2$

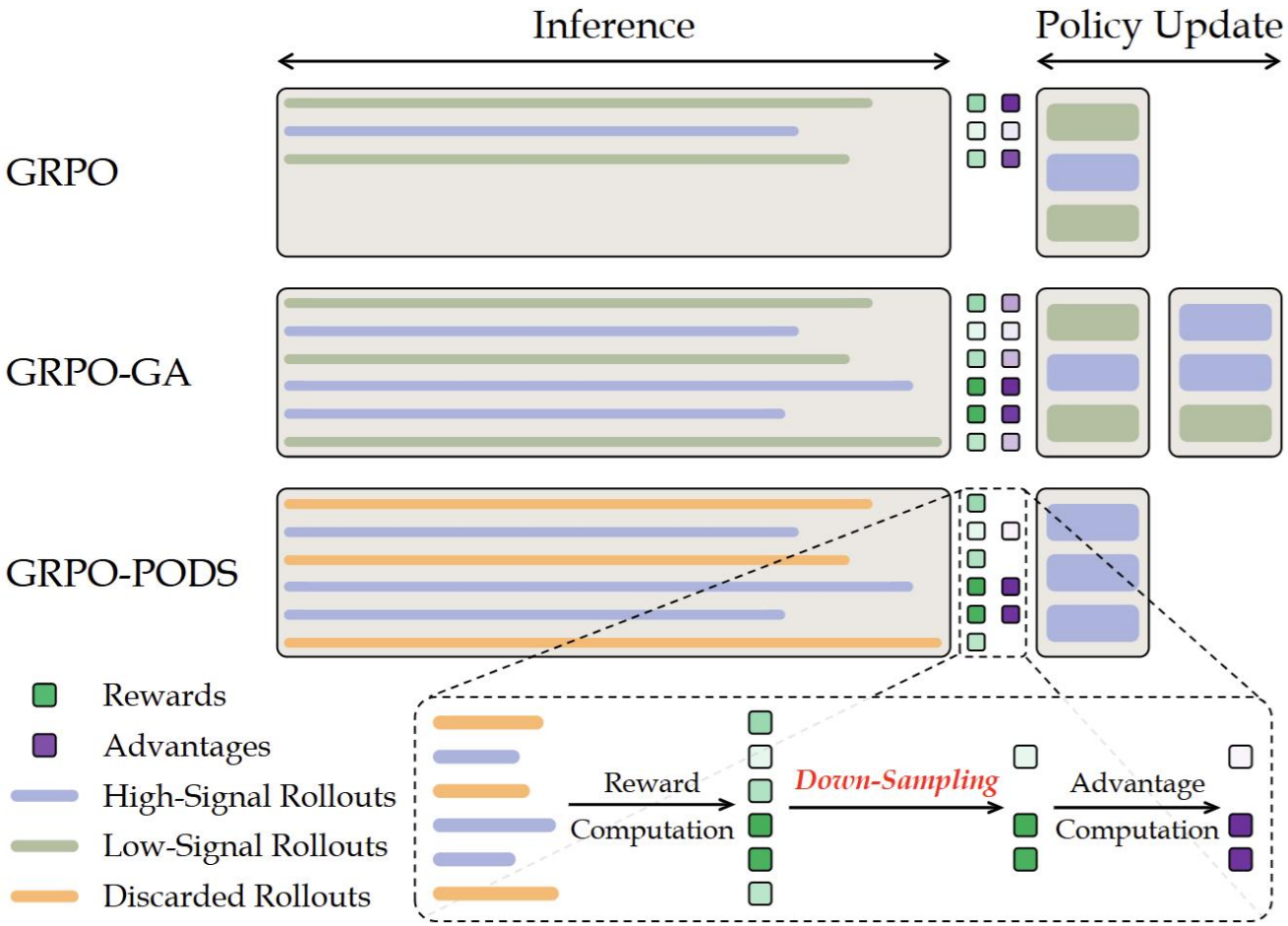
Experiments

- We evaluate PODS across diverse **hardware** configurations, model **architectures**, and model **scales** as shown below.
  - Settings (a) to (c) compares **vanilla GRPO** with **GRPO-PODS**
  - Settings (d) to (e) compares **GRPO with GA** with **GRPO-PODS**

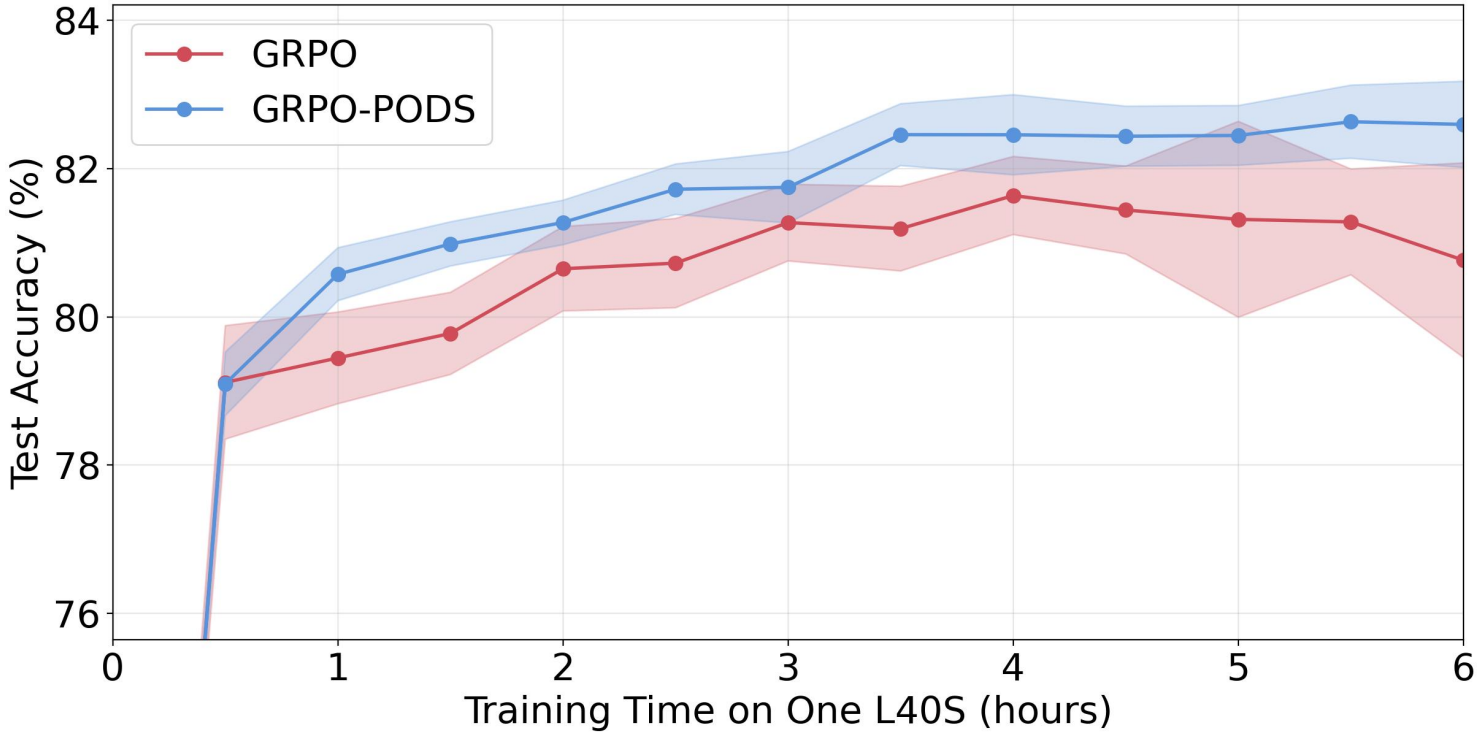
Setting	Benchmark	Model	Parameters	GPUs	Fine-tuning Method
(a)	GSM8K	Qwen2.5	3B	1 L40S	LoRA (rank 64, $\alpha = 64$)
(b)	MATH	Qwen2.5	3B	1 L40S	LoRA (rank 64, $\alpha = 64$)
(c)	GSM8K	Llama3.2	3B	1 L40S	LoRA (rank 64, $\alpha = 64$)
(d)	GSM8K	Qwen2.5	3B	8 H100s	Full-Parameter
(e)	GSM8K	Qwen2.5	7B	8 A100s	Full-Parameter

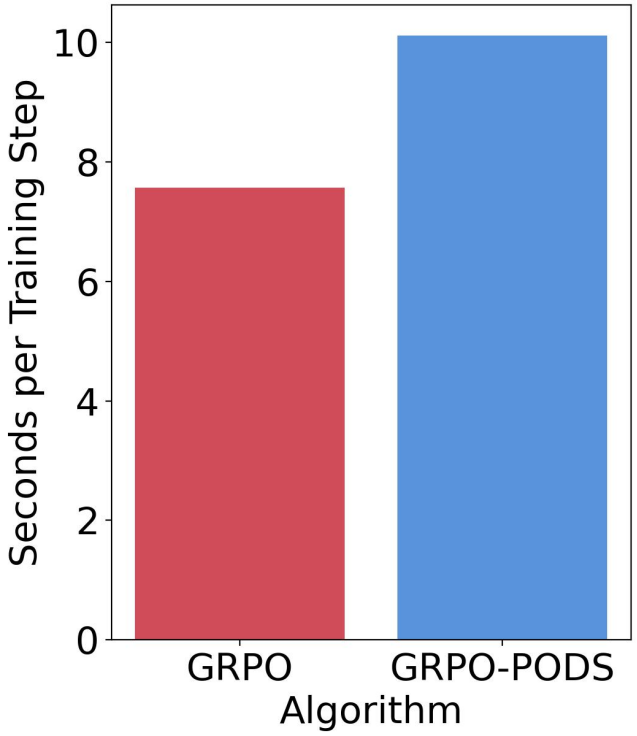
Comparing GRPO with GRPO-PODS

- **Algorithms:**
  - GRPO & GRPO-PODS
  - We align the number of rollouts per update step (m)
  - This means that PODS will be generating more rollouts per prompt than GRPO
- **In the figure:**
  - We are comparing the first row with the third row

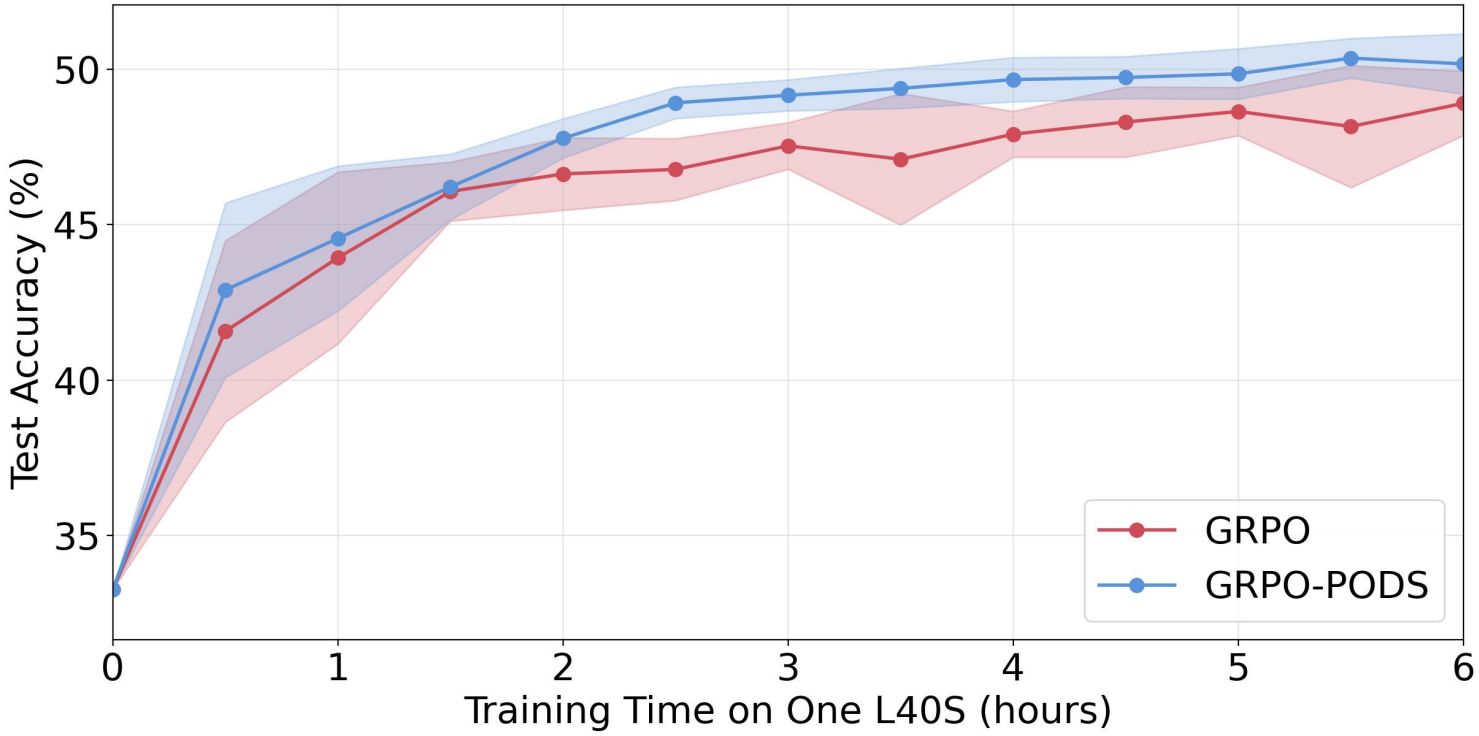


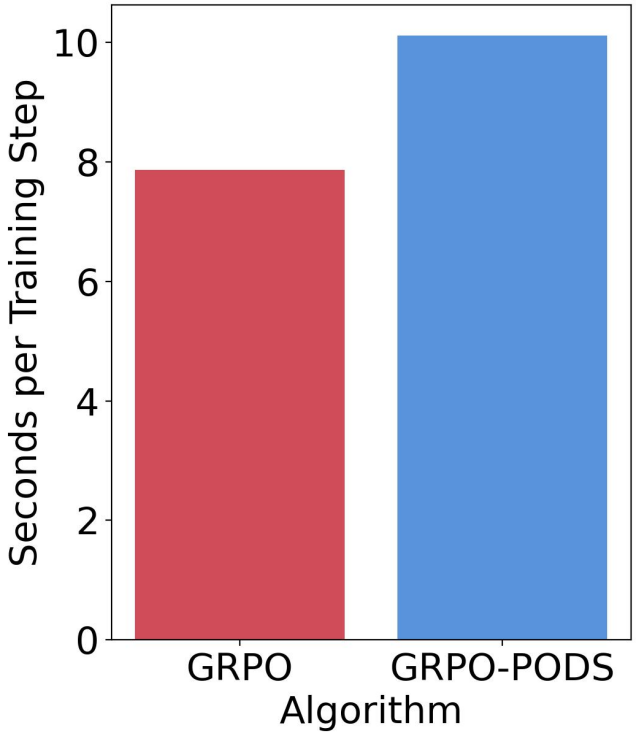
Qwen2.5-3B on GSM8K & One L40S GPU



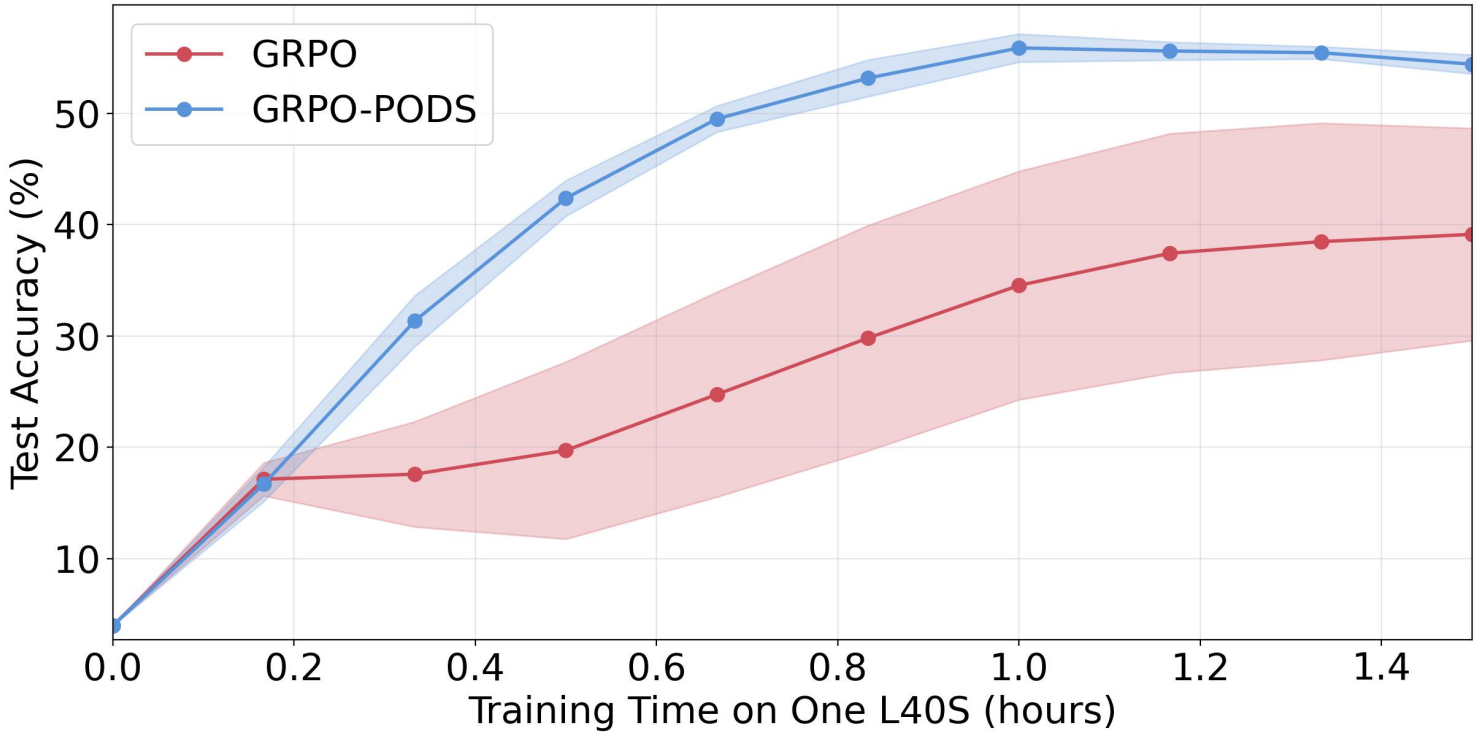


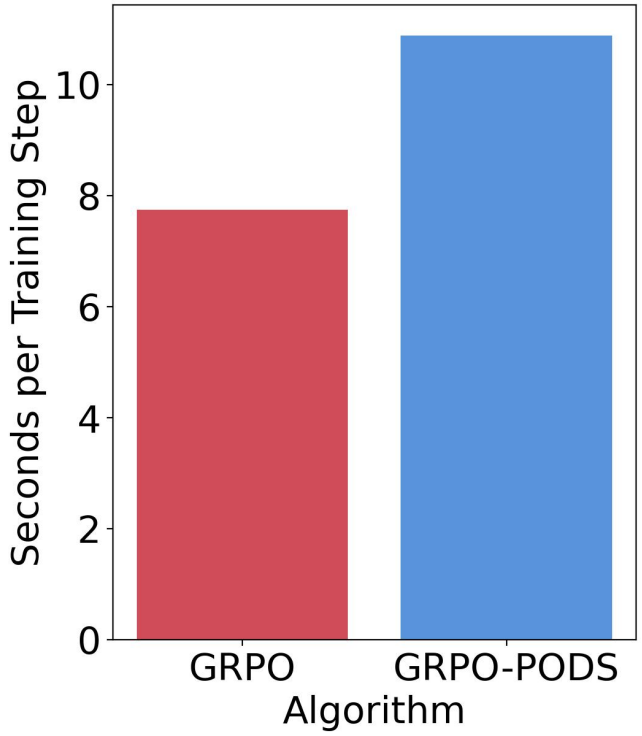
Qwen2.5-3B on MATH & One L40S GPU





Llama3.2-3B on GSM8K & One L40S GPU





Comparing GRPO with GRPO-PODS

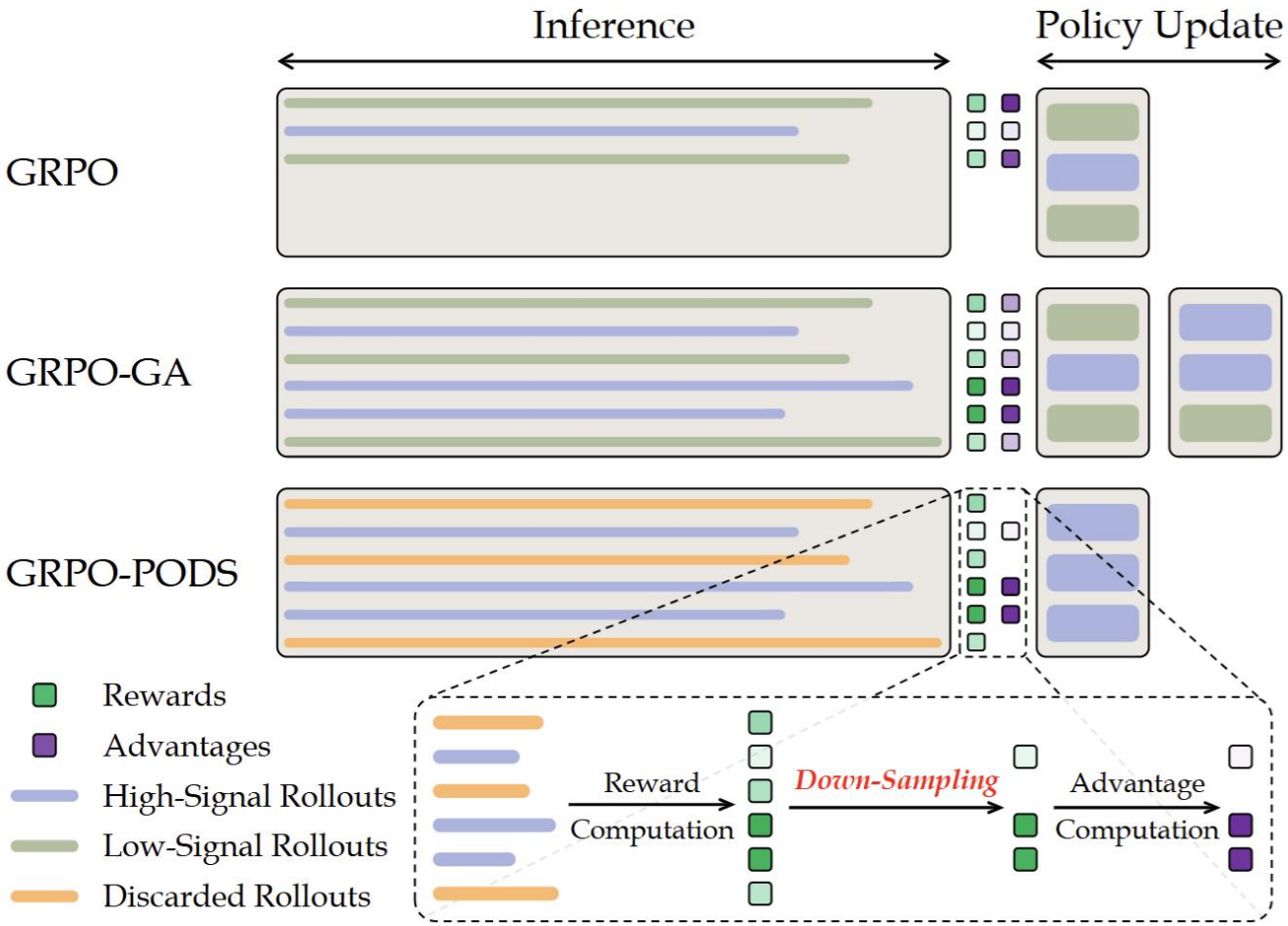
- **Experiment settings**
  - Algorithms: GRPO & GRPO-PODS (with the same #rollouts per update)
  - LoRA fine-tuning of 3B Qwen2.5 or Llama3.2, on one L40S GPU

- With PODS, RL **converges faster**, and to a **higher accuracy**

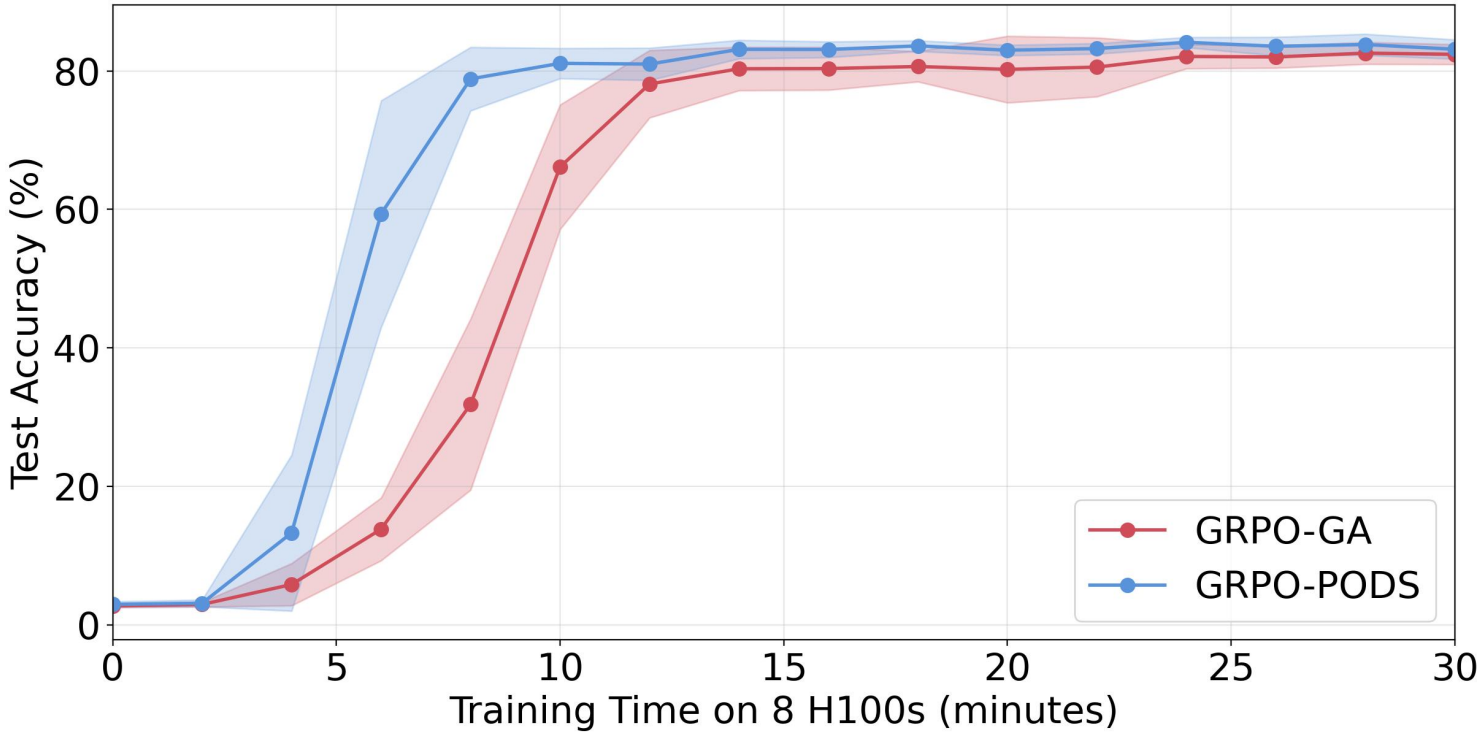
- PODS takes more time per step, since it is doing more inference
  - This means PODS achieves a higher accuracy using fewer training steps
  - Which indicates that the learning signals are stronger with PODS

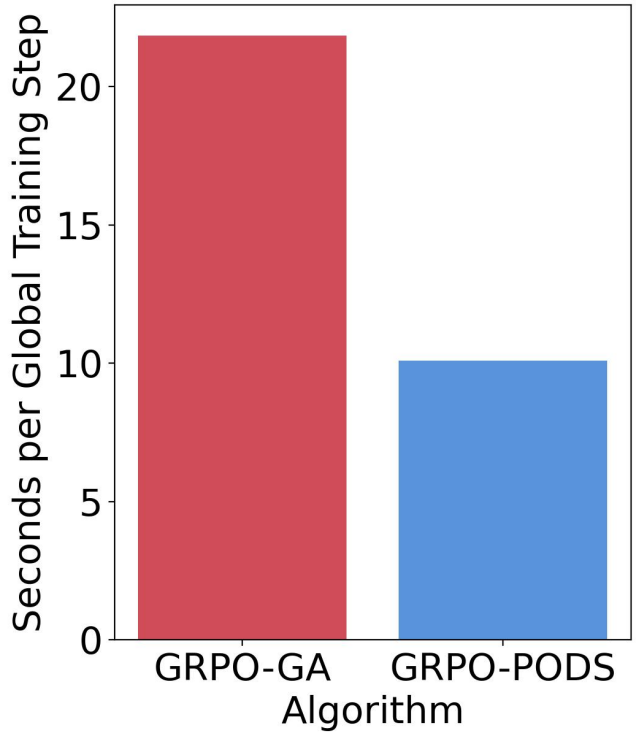
Comparing GRPO-GA with GRPO-PODS

- **Algorithms:**
  - GRPO-GA & GRPO-PODS
  - We align the number of rollouts in inference (n)
  - This means that PODS will be taking fewer update steps per prompt than GRPO-GA
- **In the figure:**
  - We are comparing the second row with the third

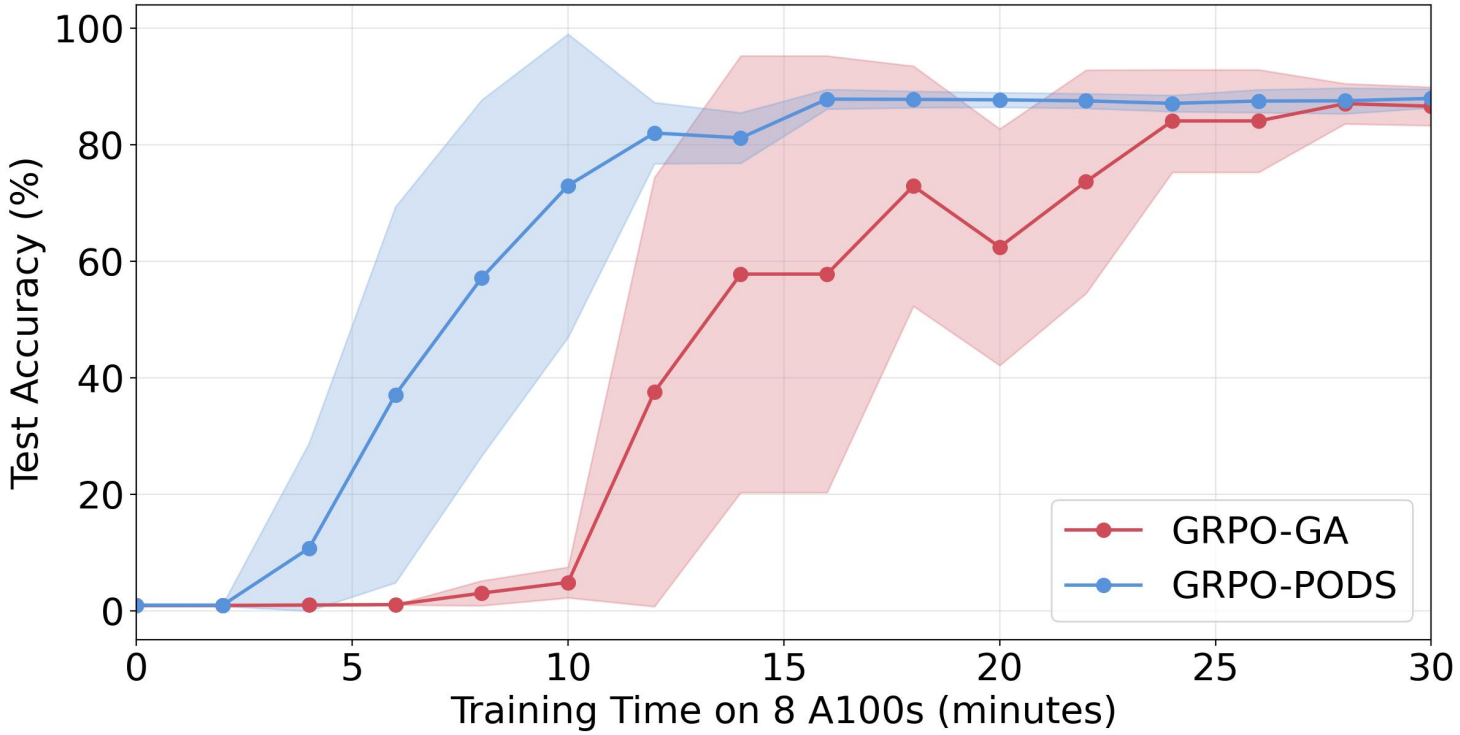


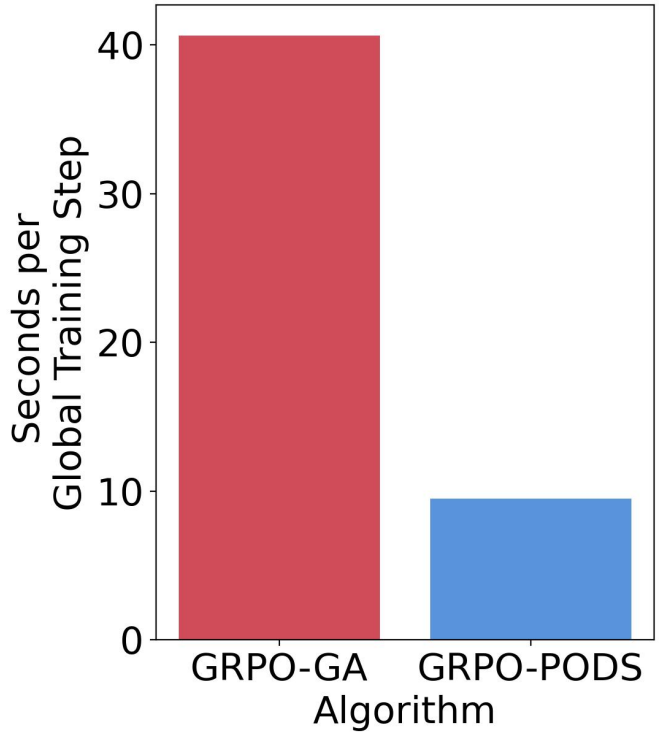
Qwen2.5-3B on GSM8K & 8 H100 GPUs





Qwen2.5-7B on GSM8K & 8 A100 GPUs





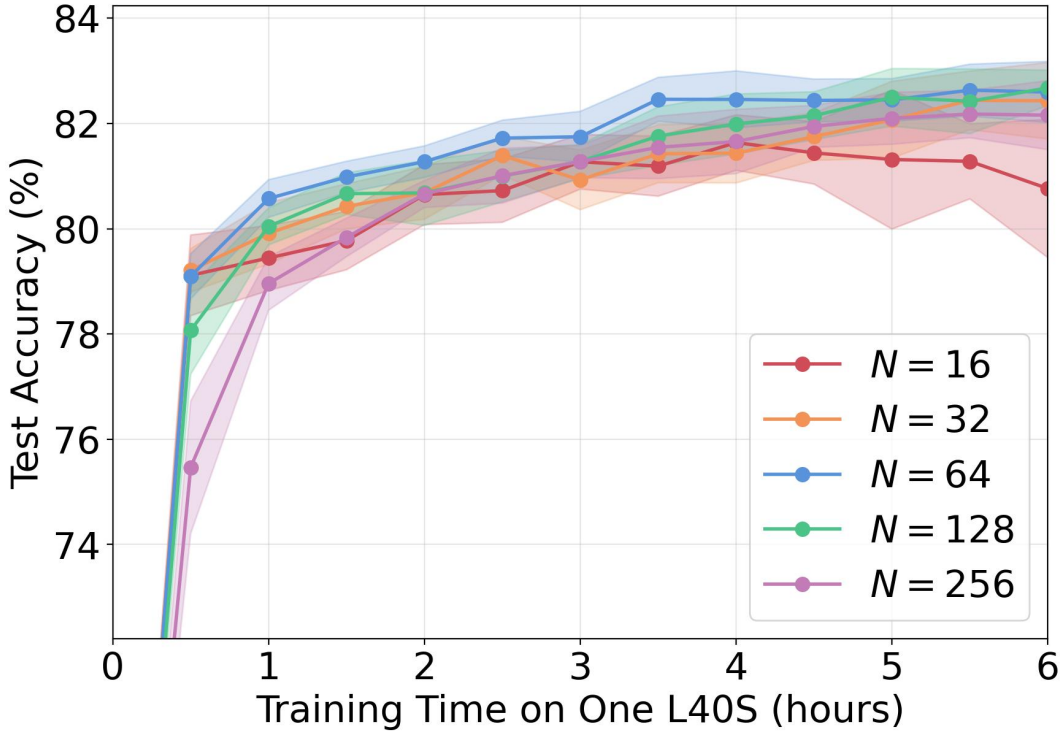
Comparing GRPO-GA with GRPO-PODS

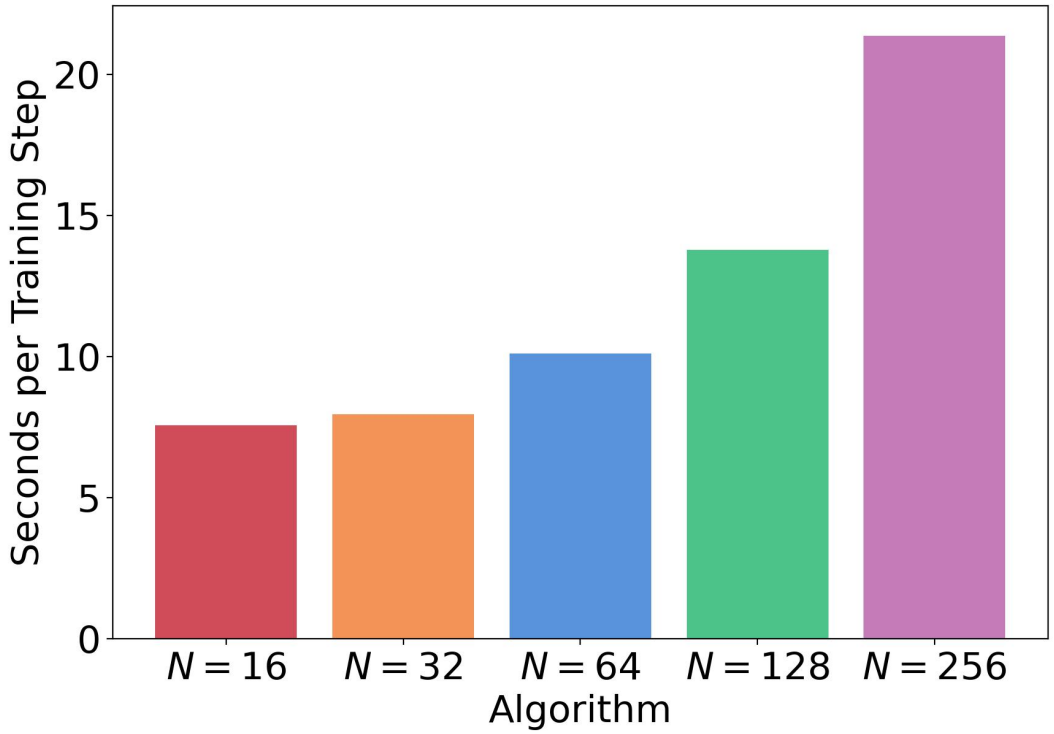
- **Experiment settings**
  - Algorithms: GRPO-GA & GRPO-PODS (same #rollouts in inference)
  - Full fine-tuning 3B / 7B Qwen2.5, on 8 H100 / A100 GPUs

- With PODS, RL **converges faster**, and to a **higher accuracy**

- PODS takes less time per step, since it is doing less update
  - Which indicates that the learning signals are well preserved each step

Fixing $m = 16$ and Varying $n \in \{16, 32, 64, 128, 256\}$



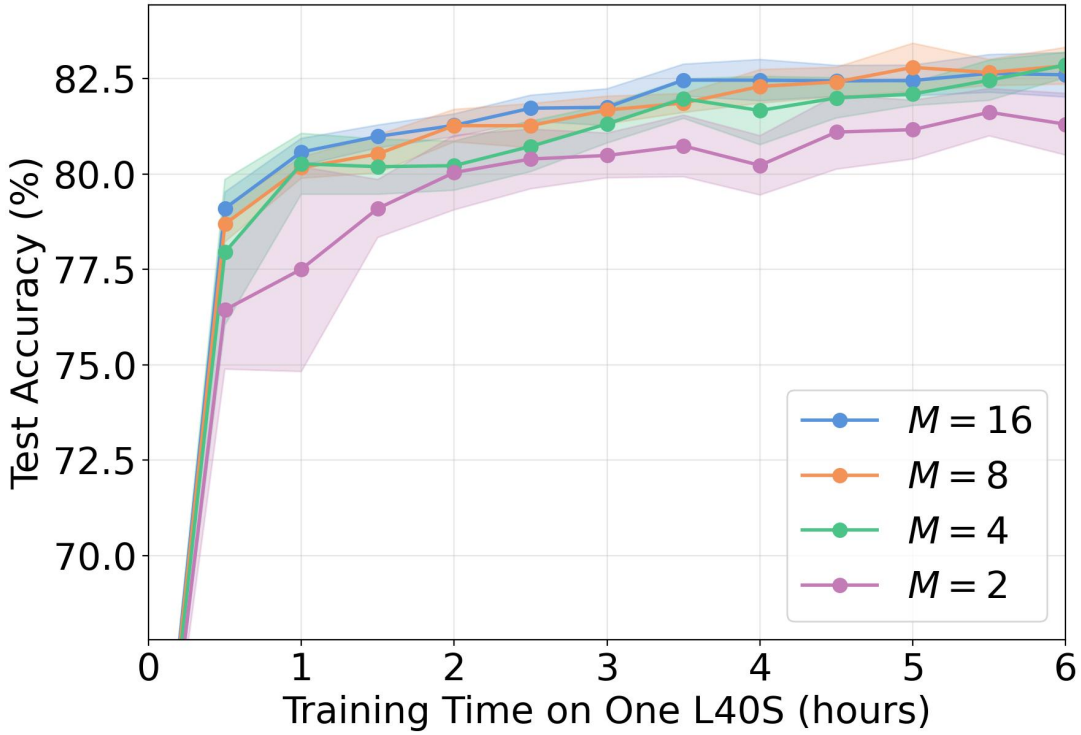


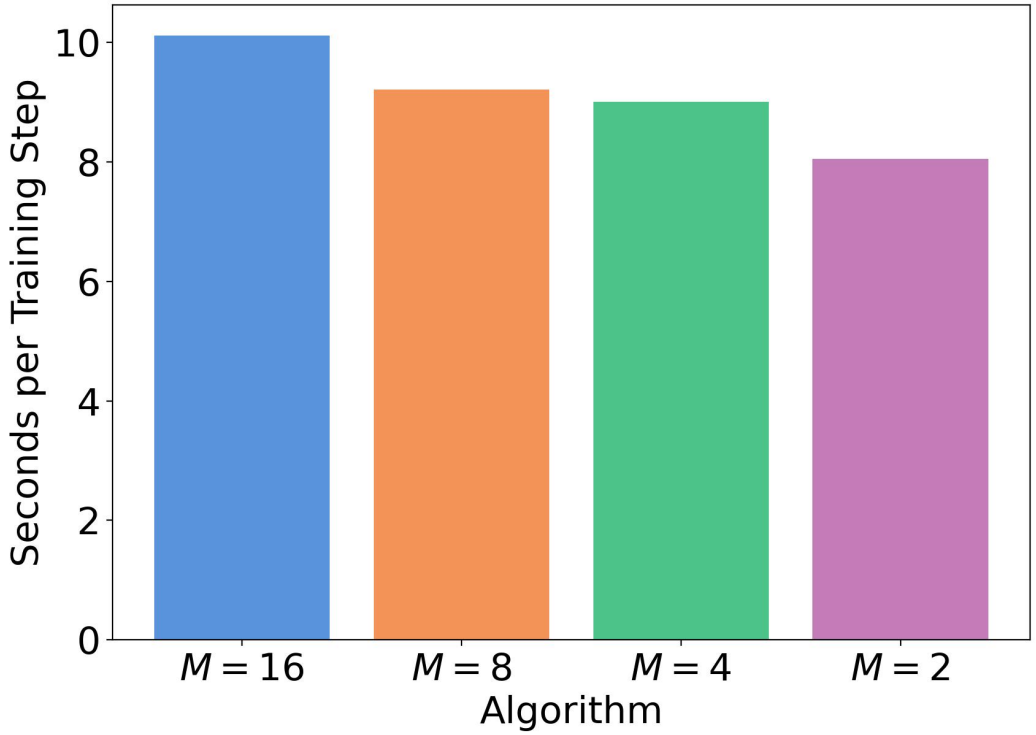
Fixing the Update Step Batch Size m

- **Experiment settings**
  - Algorithm: GRPO-PODS ($n = \{16, 32, 64, 128, 256\}, m = 16$)
  - LoRA fine-tuning Qwen2.5-3B-Instruct, on one L40S GPU

- The algorithm's performance is **single-peaked**
  - With $n = 64$ being the best, and $n = 16, 256$ being the worst
  - $n = 16$: Fewer rollouts are sampled, so the learning signal is weak
  - $n = 256$: The inference phase takes too much time, fewer steps taken

Fixing $n = 64$ and Varying $m \in \{16, 8, 4, 2\}$





Fixing the Inference Step Batch Size n

- **Experiment settings**
  - Algorithm: GRPO-PODS ($n = 64, m = \{16, 8, 4, 2\}$)
  - LoRA fine-tuning Qwen2.5-3B-Instruct, on one L40S GPU

- The algorithm's performance is similar
  - As long as m is not set too small
  - This indicates that PODS preserves the learning signals effectively

Our Contributions

- Motivated by the computation asymmetry of the two phases in RLVR algorithms, we propose the **PODS** framework
  - **Key idea:** Not all rollouts contribute equally to model improvement
  - Generate n rollouts and train on only $m < n$ of them

- We conduct a thorough theoretical and empirical study
  - We derive an $O(n \log n)$ **algorithm** for the max-variance rule
  - We demonstrate **faster convergence to improved accuracy** under different reasoning benchmarks, model and hardware regimes