Optimizing Chain-of-Thought Reasoners via Gradient Variance Minimization in Rejection Sampling and RL

Presenter: Jiarui Yao, Yifan Hao

University of Illinois Urbana-Champaign

Roadmap

Motivation

- LLMs excel at text generation, but reasoning (math, logic, planning) remains hard.
- Reasoning quality hinges on both how data is used and how training feedback (reward) is applied.
- Current fine-tuning methods (e.g., PPO, GRPO, RAFT) are expensive — need many samples per prompt and often over-engineered.

Research Question

- How can we reduce gradient variance and wasted samples in reasoning fine-tuning?
- Are complex RL algorithms (PPO, GRPO) really necessary, or can simpler approaches suffice?
- How can we make reasoning optimization data-efficient and interpretable?

Roadmap

Background

- consider the chain-of-thought (CoT) reasoning process as:

$$x \to y \to z, \quad x \in \mathcal{X}, y \in \mathcal{Y}, z \in \mathcal{Z},$$

where x is a prompt, y is the intermediate CoT and z is the final predicted answer.

- Target: minimize the negative log-likelihood of predicting the correct answer:

$$\mathcal{L}(\theta) = -\mathbb{E}_{x \sim d_0} \ln \mathbb{P}(z|x, \theta) = -\mathbb{E}_{x \sim d_0} \ln \sum_{y \in \mathcal{Y}} \mathbb{P}(y|x, \theta) \mathbb{P}(z|x, y, \theta), \tag{1}$$

where d_0 is a prompt distribution and $\mathbb{P}(\cdot|\theta)$ denotes the distribution induced by the model with parameters θ.

Derivation of the EM Algorithm

$$
\begin{aligned}
\mathcal{L}(\theta) &= -\sum_{i=1}^{m} \ln \mathbb{P}(z_i|x_i,\theta) = -\sum_{i=1}^{m} \ln \left(\sum_{y\in\mathcal{Y}} Q_i(y) \frac{\mathbb{P}(y,z_i|x_i,\theta)}{Q_i(y)} \right) \\
&\leq -\sum_{i=1}^{m} \sum_{y\in\mathcal{Y}} Q_i(y) \ln \left(\frac{\mathbb{P}(y,z_i|x_i,\theta)}{Q_i(y)} \right) \\
&= \underbrace{-\sum_{i=1}^{m} \mathbb{E}_{y\sim Q_i(y)} \ln \mathbb{P}(y,z_i|x_i,\theta)}_{\mathcal{J}_Q(\theta)} - \sum_{i=1}^{m} \mathcal{H}(Q_i(y))
\end{aligned}
$$

Derivation of the EM Algorithm

- In the E-step, our goal is to find a $Q_i(y)$ to get a sharper upper bound for $\mathcal{L}(\theta)$. In particular, the equality is reached with the posterior distribution of y:

$$Q_i(y) = \mathbb{P}(y|x_i, z_i, \theta) = \frac{\mathbb{P}(y, z_i|x_i, \theta)}{\mathbb{P}(z_i|x_i, \theta)} = \frac{\mathbb{P}(y|x_i, \theta) \cdot \mathbb{P}(z_i|y, \theta)}{\sum_{y \in \mathcal{Y}} \mathbb{P}(y|x_i, \theta)\mathbb{P}(z_i|y, \theta)}. \tag{2}$$

- In the M-step, to minimize $\mathcal{L}(\theta)$, we can fix $Q_i(y)$ as in (2) and indirectly minimize $\mathcal{J}_Q(\theta)$.

Meta EM Algorithm

Algorithm 1 Meta Algorithm: GVM-EM

Input: Initial parameters θ_0, training samples $\mathcal{D} = \{(x_i, z_i)\}_{i=1}^n$, number of epochs T, initial posterior $Q^0 = \mathbb{P}(\cdot \mid \theta_0)$.
for $t = 0, \ldots, T$ **do**
  ▷ **E-step (Expectation):**
  Sample a set of samples $\mathcal{B}_t = \{x_i, z_i\}_{i=1}^m$. Update the posterior distribution over latent CoT rationales $Q^t(\cdot)$ using Equation (4).
  For each prompt x_i, compute the required number of samples n_i^t according to (1) Theoretical Proposition 1 or (2) Practical Algorithm 2.
  Perform rejection sampling to obtain accepted responses y. Collect corresponding (x_i, z_i, y) into $\mathcal{D}_i^t$, such that $y \sim Q_i^t(\cdot)$.
  ▷ **M-step (Maximization):**
  Update model parameters via gradient descent using:

$$\nabla_\theta M(\theta_t) = -\frac{1}{m} \sum_{i=1}^{m} \frac{1}{|\mathcal{D}_i^t|} \sum_{y_j \in \mathcal{D}_i^t} \nabla_\theta \log \mathbb{P}(y_j, z_i \mid x_i, \theta).$$

end for
Output: Final model $M(\theta_T)$.

Figure: Meta Algorithm

Gradient Variance Minimization by Dynamic Sample Allocation

We begin by formulating the true gradient at iteration t under the EM objective $\mathcal{J}_{Q^t}$:

$$\nabla \mathcal{J}_{Q^t}(\theta) = -\sum_{i=1}^{m} \sum_{y \in \mathcal{Y}} Q_i^t(y) \nabla \ln \mathbb{P}(y, z_i | x_i, \theta) = -\sum_{i=1}^{m} \mathbb{E}_{y \sim Q_i^t} \nabla \ln \mathbb{P}(y, z_i | x_i, \theta \tag{3}$$

where $Q_i^t(y) = \mathbb{P}(y | x_i, z_i, \theta_{t-1})$ is the posterior distribution of y.

Lemma 1 (Unbiased Gradient Estimator)

In the iteration t, denoting $\mathcal{D}_i^t$ as the set of accepted samples on y related to (x_i, z_i), we have the following unbiased gradient estimator for $\mathcal{J}_{Q^t}$:

$$-\sum_{i=1}^{m} \frac{1}{n_i^t p_i^t} \sum_{y_j \in \mathcal{D}_i^t} \nabla \ln \mathbb{P}(y_j, z_i | x_i, \theta_{t-1}), \tag{4}$$

where $p_i^t = \mathbb{E}_{y \sim \mathbb{P}(\cdot | x_i, \theta)} P(z_i | y, \theta)$ is the average accept rate of rejection sampling.

Gradient Variance Minimization

Lemma 2 (Upper Bound of Variance of Gradient Estimator)

$$\mathbb{V}\left(\sum_{i=1}^{m}\frac{1}{n_i^t p_i^t}\sum_{y_j\in\mathcal{D}_i^t}\nabla\left(\ln\mathbb{P}(y_j,z_i|x_i,\theta)\right)\right)\leq\sum_{i=1}^{m}\frac{1}{n_i^t p_i^t}\underbrace{\mathbb{E}_{y\sim Q_i^t}\left\|\nabla\left(\ln\mathbb{P}(y,z_i|x_i,\theta)\right)\right\|^2}_{G_i^2}.$$

Given a fixed total sampling budget N, we seek to allocate $\{n_i^t\}$ to minimize this upper bound: $\min\left\{\sum_{i=1}^m \frac{G_i^2}{p_i^t n_i^t}\right\}$, s.t. $\sum_{i=1}^m n_i^t = N$. In practice, some prompts are totally beyond the ability of the current LLMs or cannot be evaluated by the verifier (e.g., due to some label error). This leads to extremely low acceptance rates and unstable gradient estimations. To mitigate this, we introduce a regularization term that penalizes sampling on such prompts. The revised objective becomes:

$$\min\left\{\sum_{i=1}^{m}\frac{1}{1+\alpha/(p_i^t)^\beta}\frac{G_i^2}{p_i^t n_i^t}\right\},\quad \text{s.t.}\quad \sum_{i=1}^{m} n_i^t = N, \tag{5}$$

where $\alpha > 0, \beta \geq 2$ are hyperparameters that control the regularization strength.

Gradient Variance Minimization

Theorem 3

The optimal number of samples allocated to each prompt is:

$$n_i^t = N \cdot \frac{G_i / \sqrt{p_i^t + \frac{\alpha}{(p_i^t)^{\beta-1}}}}{\sum_{l=1}^{n} G_l / \sqrt{p_l^t + \frac{\alpha}{(p_l^t)^{\beta-1}}}} \propto \frac{G_i}{\sqrt{p_i^t + \frac{\alpha}{(p_i^t)^{\beta-1}}}}, \quad \forall i = 1, \ldots, m.$$

Theoretical Result

We take the notations below for simplification:

$$\Delta_1(k,T) := \sum_{t=1}^{T}\sum_{r=0}^{k-1}\mathbb{E}\|\nabla_\theta \mathcal{L}_t(\theta_{kt-k+r})\|^2 > 0,$$

$$\Omega(k,T) := \sum_{t=1}^{T}\sum_{r=0}^{k-1}\mathbb{E}V\left(g_{kt-k+r}\right) > 0,$$

Under mild smoothness conditions, we can derive the following result.

Theorem 4 (Decreasing rate with smoothness condition.)

Suppose $-\ln \mathbb{P}(y,z|x,\theta)$ *is* $1/\gamma$*-smooth with respect to* θ*. If* $0 < \eta \leq \gamma$*, then the proposed algorithm satisfies that*

$$\mathbb{E}\left[\mathcal{L}(\theta_{kT}) - \mathcal{L}(\theta^*)\right] - \mathbb{E}\left[\mathcal{L}(\theta_0) - \mathcal{L}(\theta^*)\right] \leq -\frac{\eta}{2}\Delta_1(k,T) + \frac{\eta^2}{2\gamma}\Omega(k,T).$$

Practical Version of GVM Algorithm

Algorithm 2 GVM: Practical Implementation

1: **Input:** Model parameter θ, training samples $\{x_i, z_i\}_{i=1}^n$, total inference budget N, parameter estimation sample size N', penalty parameter $\{\alpha, \beta\}$.

2: **for** $i = 1, \ldots, m$ **do**

3: For each prompt x_i, sample N' times to get $\{x_i, y_i^j, z_i^j\}_{j=1}^{N'}$.

4: Estimate the accept rate p_i^t and Lipschitz bounds G_i on each prompt x_i as

$$p_i = \frac{\sum_{j=1}^{N'} \mathbf{1}(z_i^j = z_i)}{N'},$$

$$G_i = \sum_{1 \le j \le N', z_i^j = z_i} \frac{1}{N' p_i} \|\nabla_\theta \ln \mathbb{P}(y_i^j, z_i | x_i, \theta)\|_2.$$

5: Calculate sample size $\{n_i\}$ as:

$$n_i = N \cdot \frac{G_i / \sqrt{p_i + \frac{\alpha}{(p_i)^{\beta-1}}}}{\sum_{l=1}^n G_l / \sqrt{p_l + \frac{\alpha}{(p_l)^{\beta-1}}}}.$$

6: **end for**

7: **Output:** $\{n_i\}_{i=1}^m$.

Figure: Practical algorithm

Pipeline Overview

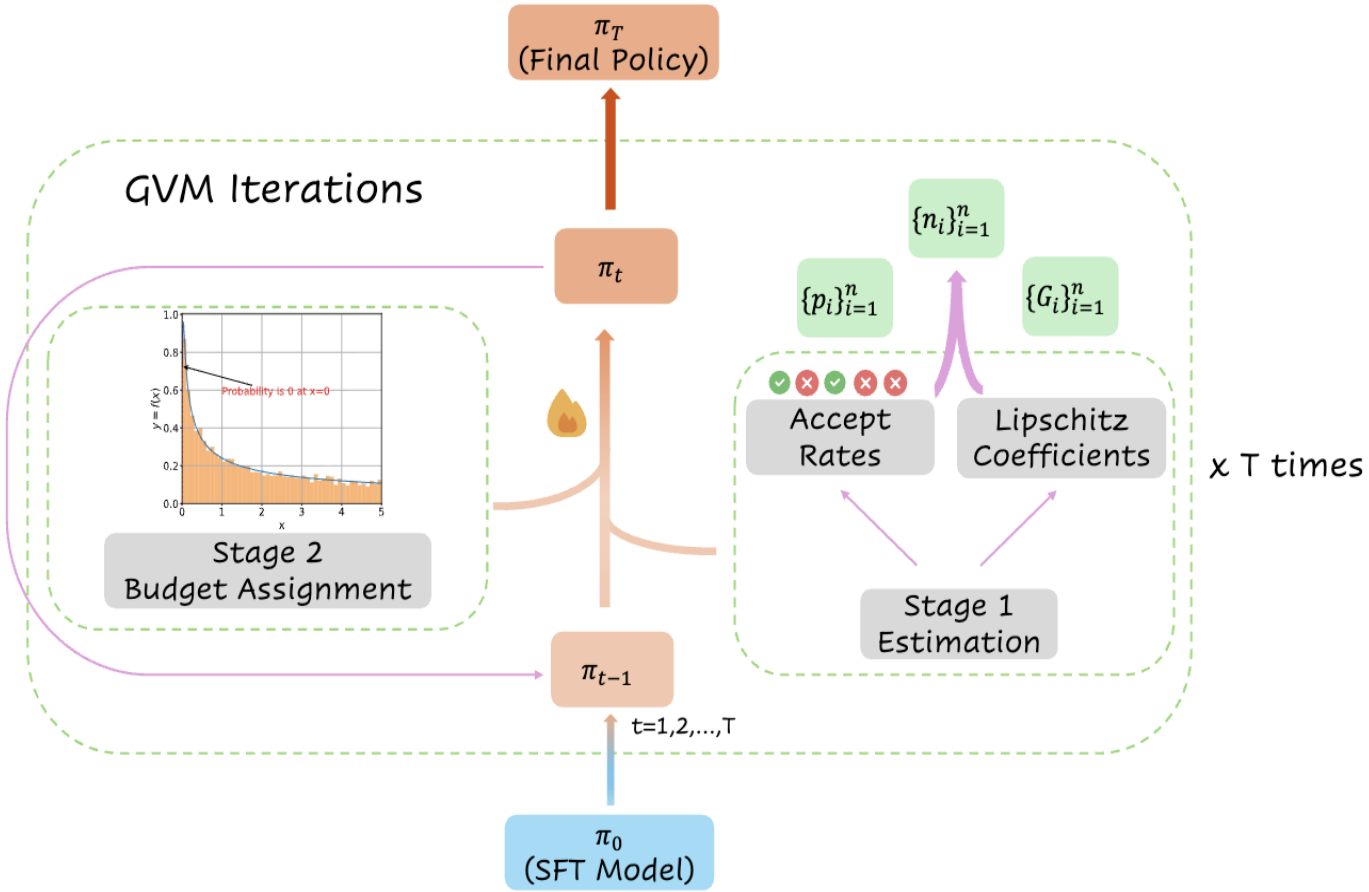

Figure 1: The demonstration of the whole pipeline for GVM. Starting from π_0, which a supervised fine-tuning (SFT) model, GVM will iteratively conduct the estimation and budget assignment process for T times according to the description in Algorithm 1. Each iteration could be decomposed into two stages, and the final policy model will be returned after those iterations.

Figure: Pipeline overview

Roadmap

Results

Table 1: Performance of different algorithms across five benchmarks including Math500 (Hendrycks et al., 2021), Minerva Math (Lewkowycz et al., 2022), Olympiad Bench (He et al., 2024), AIME24, and AMC23. From the results, we could observe that after reweighting the sample size of prompts, GVM-RAFT++ and GVM-GRPO could outperform both vanilla RAFT++ and GRPO.

Method		Math500	Minerva Math	Olympiad Bench	AIME24	AMC23	5 Average
Qwen2.5 Math-1.5B	Base	56.35	17.00	25.22	3.33	37.81	27.94
	GRPO	70.78	29.00	33.56	10.00	47.19	38.11
	RAFT++	69.02	27.71	31.74	9.58	44.06	36.42
	GVM-GRPO	**73.92**	**29.96**	**36.26**	**12.92**	49.06	**40.42**
	GVM-RAFT++	72.90	29.04	36.20	9.17	**51.88**	39.64
Qwen2.5 Math-7B	Base	42.00	12.82	19.20	12.92	30.00	23.39
	GRPO	81.20	36.03	**44.15**	20.83	63.12	49.07
	RAFT++	**81.68**	35.85	43.83	20.83	63.12	49.06
	GVM-GRPO	81.55	36.26	43.56	**22.92**	**65.00**	**49.86**
	GVM-RAFT++	81.00	**36.67**	43.48	**22.92**	61.56	49.13
Llama-3.2 1B-Instruct	Base	24.28	4.37	4.80	0.83	9.06	8.67
	GRPO	**33.35**	**6.99**	**7.65**	1.67	12.19	12.37
	RAFT++	30.10	6.62	6.46	0.83	13.44	11.38
	GVM-GRPO	32.02	6.76	7.22	**2.08**	**15.31**	**12.68**
	GVM-RAFT++	30.08	6.07	6.48	0.83	13.44	11.38
Llama-3.2 3B-Instruct	Base	35.62	9.83	9.09	2.92	15.00	14.49
	GRPO	**53.40**	19.16	**20.67**	**8.33**	27.50	25.81
	RAFT++	47.38	17.69	17.20	**8.33**	29.06	23.93
	GVM-GRPO	51.75	17.74	19.48	7.50	**33.44**	**25.98**
	GVM-RAFT++	49.05	**20.04**	17.87	6.25	29.06	23.93

Figure: Main experiments results.

Sample Size vs Accept Rates

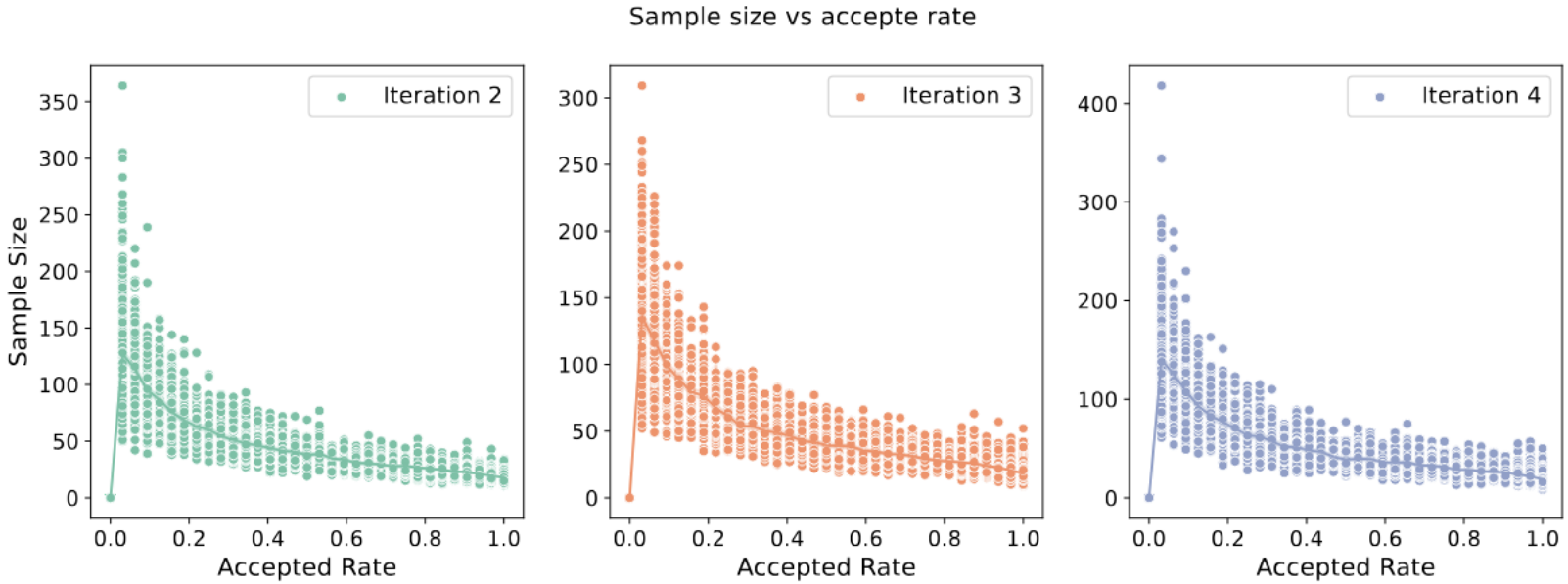

Figure 3: The assigned sample budget in GVM-RAFT++ with $N' = 32$, $N = 32n$ for three iterations.

Figure: Sample size vs accept rates for GVM.

Performance Trend

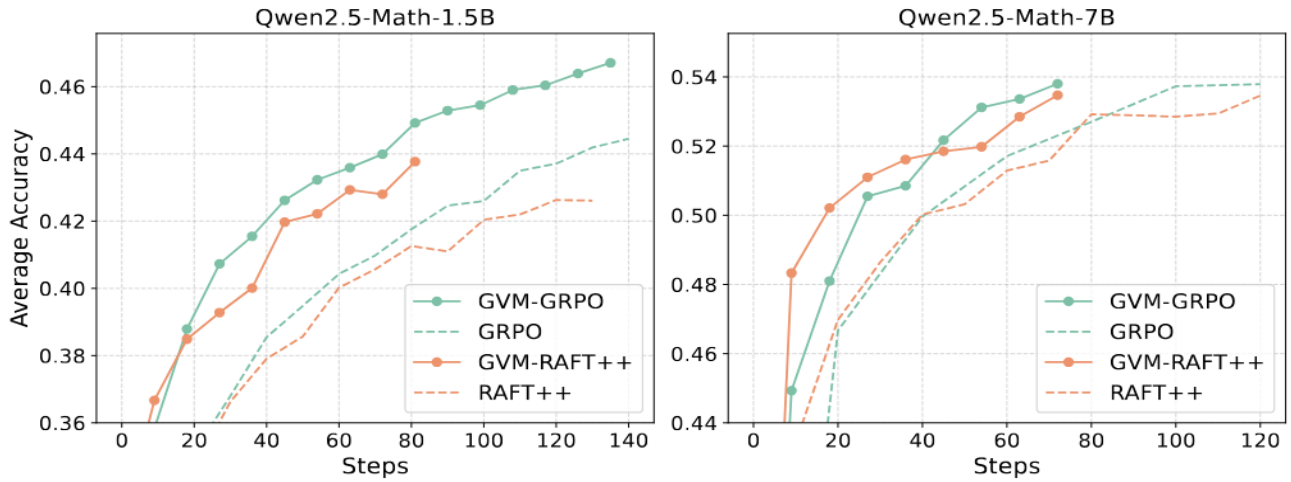

Figure 6: The average accuracy of RAFT++, GVM-RAFT++ and GRPO, GVM-GRPO with $N' = 8, N = 8n$ respectively on Math500, Minerva Math and Olympiad Bench. Applying the GVM sample strategy to RL algorithms like GRPO achieves similar results to GVM-RAFT++ compared to vanilla GRPO. For vanilla RAFT++ and GRPO, the rollout number per prompt is set to 8 as well.

Figure: Accuracy trend with steps.

Q&A

Thanks!