Reinforcement Learning and Transformer-based Approaches for Mathematical Question Answering

521H0034 Lương Chí Dũng

Abstract

Question Answering (QA) is a core task in natural language processing that requires models to comprehend input queries and retrieve or generate accurate responses. Within QA, mathematical question answering (math QA) poses unique challenges: it demands precise multi-step reasoning, arithmetic computation, and robust handling of numerical constraints. In this study, we focus on math QA using the vi-gsm8k dataset—a Vietnamese translation of the GSM8K benchmark—where models must exhibit strong compositional reasoning and numerical fidelity.

To enhance reasoning capabilities, we investigate two reinforcement learning (RL) algorithms: Proximal Policy Optimization (PPO) (OpenAI, 2017) and Group Relative Policy Optimization (GRPO) (DeepSeek-AI, 2024). We conduct a lightweight PPO demonstration by fine-tuning GPT2-small to generate negative student feedback, illustrating PPO's training dynamics. For math QA, we compare mT5-small, GPT2-medium, and Qwen 2.5-3B (fine-tuned with QLoRA) under supervised training. We then apply GRPO to Qwen 2.5-3B, observing measurable improvements in solution accuracy and reasoning coherence compared to its baseline performance.

CONTENTS

I. Introduction

Automatic Question Answering (QA) aims to build systems that interpret a natural-language query and return precise answers from structured or unstructured sources. Early QA systems were predominantly rule-based, relying on handcrafted syntactic and semantic rules applied over narrow domains (e.g., LUNAR, BASEBALL) and ontologies to parse questions and fetch answers. While effective in closed domains, rule-based QA struggles to scale to open-domain settings due to the combinatorial complexity of linguistic phenomena and world knowledge.

A. Extractive QA with Pretrained Encoders

The advent of deep learning enabled extractive QA architectures, in which a pre-trained encoder (e.g., BERT) is fine-tuned to select answer spans from a provided context. BERT's bidirectional Transformer encoder achieved state-of-the-art on datasets such as SQuAD by jointly conditioning on left and right context and adding a simple span-prediction head. Subsequent extensions (RoBERTa, ALBERT) further improved extractive performance through larger pre-training corpora and optimization tweaks.

B. Generative QA with Large Language Models

More recently, generative QA leverages autoregressive LLMs (e.g., GPT-3) to produce free-form answers without explicit context retrieval. Brown et al. showed that GPT-3, a 175 B-parameter model, attains strong few-shot QA performance by conditioning on few exemplars in the prompt. Generative QA excels in open-domain and conversational settings but can hallucinate factual errors and often lacks reliable multi-step reasoning.

C. Challenges of Mathematical QA

Math QA tasks (e.g., GSM8K) require precise multi-step reasoning, arithmetic operations, and strict numerical consistency, which outstrip most off-the-shelf generative LLMs. On vi-gsm8k (the Vietnamese translation of GSM8K), models must not only parse problem statements but also chain deductive steps and compute exact values, demanding stronger logical coherence and error correction capabilities than standard QA tasks.

D. *Reinforcement Learning for Reasoning*

To endow LLMs with improved creativity, reinforcement learning (RL) techniques have been applied post-training. Proximal Policy Optimization (PPO), introduced by Schulman et al. (2017), is a stable policy-gradient method that clips updates to constrain deviation from the behavior policy. PPO has been widely used in RL-from-Human-Feedback (RLHF) to align model outputs with desired behaviors.

Recently, Group Relative Policy Optimization (GRPO) developed by DeepSeek researchers was proposed to further boost reasoning capabilities by replacing the value-model baseline in PPO with group-averaged rewards, reducing memory overhead and emphasizing comparative judgment among candidate outputs.

II. Reinforcement Learning in LLMs

Large language models (LLMs) have achieved remarkable success through multi-stage training pipelines. In this section, we first review the standard pipeline from pretraining to instruction-tuning, then introduce the subsequent stages of preference training and reasoning training. We then delve into **Proximal Policy Optimization (PPO)** and **Group Relative Policy Optimization (GRPO)**. We then describe our application of PPO to a GPT2-small model to generate **negative** Feedback from Vietnamese students. The application of GRPO will be discussed later in **Chapter III**.

A. *LLM Training Pipeline*

The typical LLM training pipeline consists of:

1) **Pretraining:** Self-supervised learning on massive text corpora to learn general language representations.
2) **Instruction-Tuning:** Supervised fine-tuning on (instruction, response) pairs so that the model follows user prompts more reliably.

After instruction-tuning, the model becomes more practical for downstream applications but may still lack creativity in its responses or struggle with complex logical reasoning.

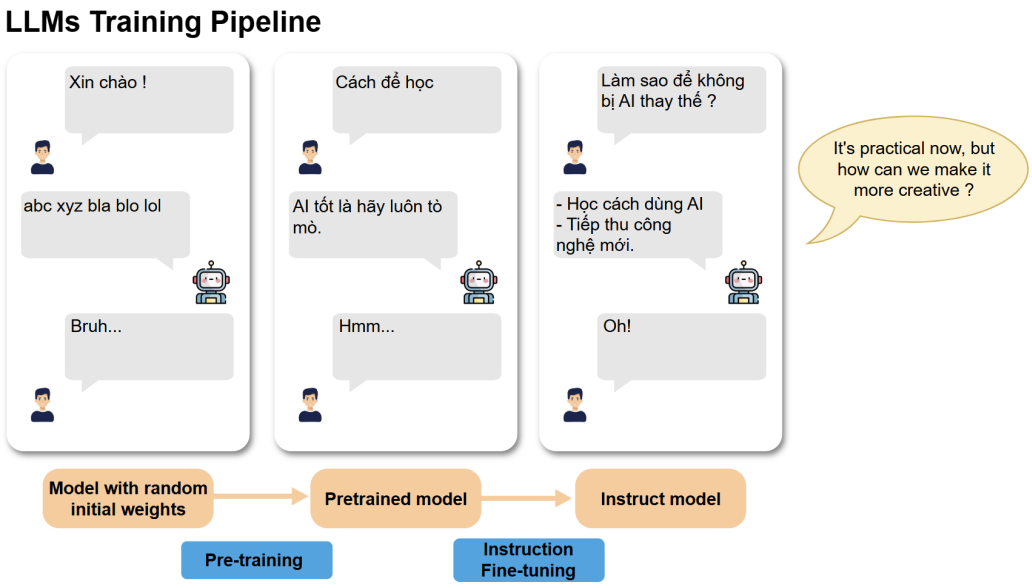

Figure 1: Overview of the Pretraining → Instruction-Tuning Pipeline.

B. *Beyond Instruction-Tuning: Preference and Reasoning Fine-Tuning*

To further align LLMs with user preferences and improve their reasoning abilities, two additional fine-tuning stages have been proposed:

- **Preference Fine-Tuning (a.k.a. RLHF):** Reinforcement learning from human feedback, where the model is rewarded for outputs that humans rate more highly.
- **Reasoning Fine-Tuning:** Specialized RL methods aimed at enhancing multi-step logical reasoning, often by shaping rewards around intermediate reasoning steps or comparative evaluation of candidate solutions.

Although preference training can steer the model towards more desirable style and content, it does not directly target the model's ability to perform complex deduction or arithmetic. Reasoning fine-tuning methods close this gap by optimizing directly for reasoning quality rather than generic human preference.

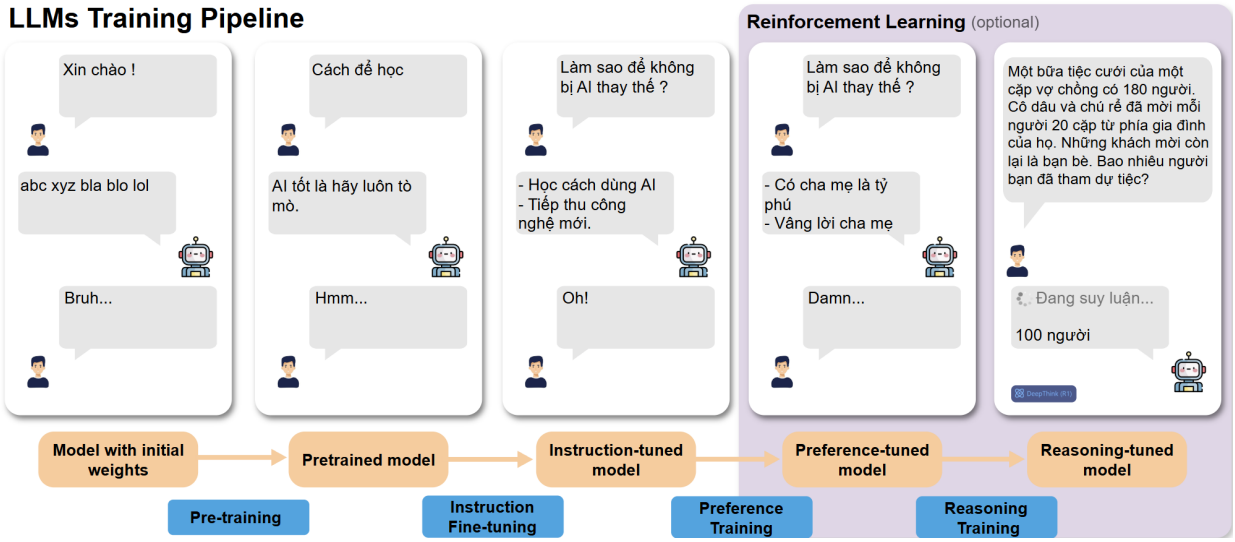

Figure 2: Extension of the Pipeline with Preference and Reasoning Fine-Tuning.

C. *Proximal Policy Optimization (PPO)*

1) Actor–Critic Objective: Proximal Policy Optimization (PPO) is implemented in an actor–critic framework. In addition to the clipped policy objective, we include a value function loss and an entropy bonus to encourage exploration. The full loss to minimize is:

$$L(\theta) = \mathbb{E}_t\Big[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \tag{1}$$

$$- c_v(V_\theta(s_t) - R_t)^2 + c_e\mathcal{H}(\pi_\theta(\cdot|s_t))\Big] \tag{2}$$

where

- θ are the policy parameters. θ_{old} are the parameters before the update.
- $\mathbb{E}_t[\dots]$ denotes the empirical average over transitions in a batch.
- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio.
- $\hat{A}_t$ is the advantage estimate. A common way to estimate it is using Generalized Advantage Estimation (GAE):

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t-1}\delta_{T-1}$$

  where $\delta_t = R_t + \gamma V_{\theta_{\text{old}}}(s_{t+1}) - V_{\theta_{\text{old}}}(s_t)$. Alternatively, a simpler form is $\hat{A}_t = R_t - V_{\theta_{\text{old}}}(s_t)$.
- R_t is the return value from the reward model. This serves as the target for the value function.
- $V_\theta(s_t)$ is the current value function estimate.
- $\mathcal{H}(\pi_\theta(\cdot|s_t))$ is the entropy of the policy at state s_t, encouraging exploration.
- ϵ is the clipping hyperparameter, typically a small positive value.
- c_v and c_e are coefficients (hyperparameters) for the value function loss and the entropy bonus, respectively, controlling their influence on the overall objective.

2) Interpretation of Components:

- **Policy loss** L^{CLIP}**:** maximizes the expected advantage while preventing large policy updates by clipping the probability ratio.
- **Value loss** L^{VF}**:** trains the value function $V_\theta(s_t)$ to accurately predict the expected future returns, which helps in reducing the variance of the advantage estimates. The loss is typically the squared error between the predicted value and the target return R_t.
- **Entropy bonus** L^{ENT}**:** encourages exploration by penalizing deterministic policies (policies with low entropy).

	$p_t(\theta)\ > 0$	A_t	Return Value of min	Objective is Clipped	Sign of Objective	Gradient
1	$p_t(\theta) \in [1-\epsilon, 1+\epsilon]$	$+$	$p_t(\theta)A_t$	no	$+$	✓
2	$p_t(\theta) \in [1-\epsilon, 1+\epsilon]$	$-$	$p_t(\theta)A_t$	no	$-$	✓
3	$p_t(\theta) < 1-\epsilon$	$+$	$p_t(\theta)A_t$	no	$+$	✓
4	$p_t(\theta) < 1-\epsilon$	$-$	$(1-\epsilon)A_t$	yes	$-$	**0**
5	$p_t(\theta) > 1+\epsilon$	$+$	$(1+\epsilon)A_t$	yes	$+$	**0**
6	$p_t(\theta) > 1+\epsilon$	$-$	$p_t(\theta)A_t$	no	$-$	✓

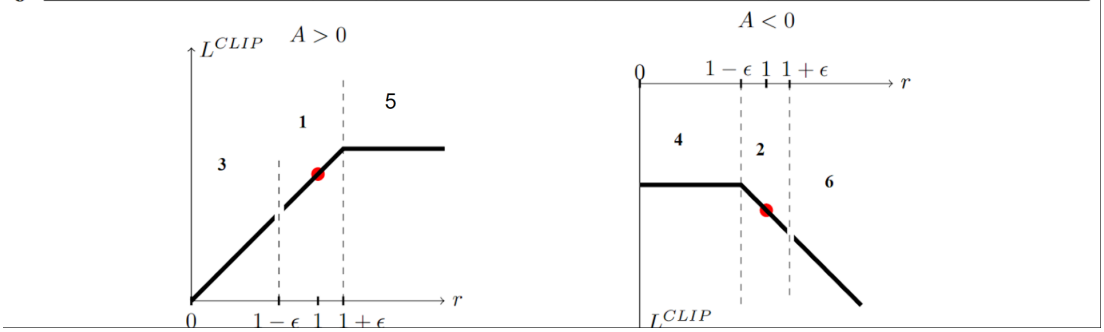

Figure 3: PPO recap table.

3) Application: Negative Feedback Generation with a PhoBERT Reward Model: To bias GPT2-small toward generating negative student feedback, we replace the simple rule-based reward with scores from a fine-tuned PhoBERT classifier acting as a reward model:

1) **Policy:** a fine-tuned GPT2-small on generating student feedback (π_θ).
2) **Reward model:** PhoBERT fine-tuned to classify feedback sentiment. Given a generated response a_t, the reward $r_t = f_{\text{PhoBERT}}(a_t)$ reflects the negative-sentiment probability.
3) **Rollouts:** generate a batch of feedback responses by sampling actions from the current policy π_θ.
4) **Compute returns:** calculate the rewards R_t. Depending on the implementation, a terminal value might be added if the episode doesn't naturally end.
5) **Estimate advantages:** compute the advantage estimate $\hat{A}_t$, often using $R_t - V_{\theta_{\text{old}}}(s_t)$ or a more advanced method like GAE. Here, s_t would represent the context before generating the feedback.
6) **Optimize:** update the policy parameters θ by maximizing the combined objective function $L(\theta)$ using gradient ascent. Simultaneously, the value function parameters are updated to minimize the value loss.

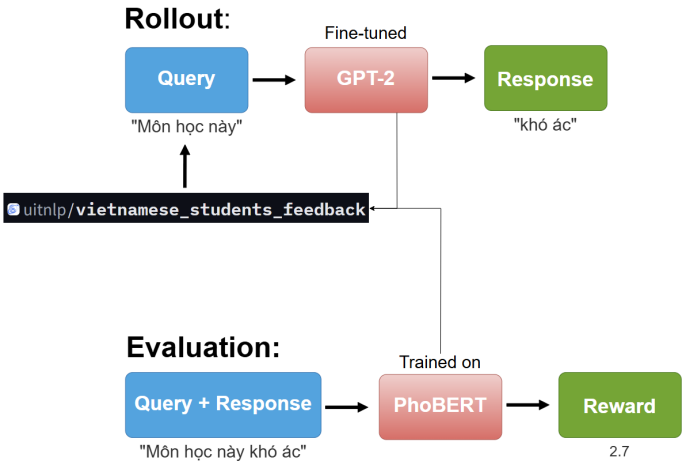

Figure 4: Overall PPO application pipeline.

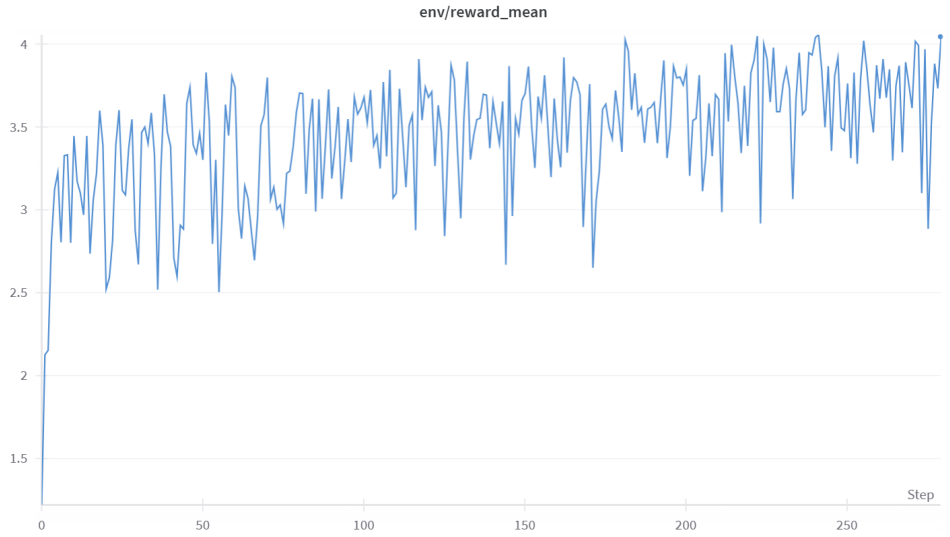

Figure 5: Mean reward value over time.

After PPO fine-tuning with the PhoBERT reward model, the policy shifts to generate feedback with higher negative-sentiment scores (e.g. “bài làm còn thiếu luận cứ rõ ràng”, “phòng học cần gắn thêm máy lạnh”), rather than neutral or generic comments.

	query	feedback (before)	feedback (after)	rewards (before)	rewards (after)
0	cô rất nhiệt tình	, vui tính, mong từ thầy ra các bạn bắt chước....	nhưng giờ rảnh lại chỉ mong qua bài thì chưa ...	1.469637	4.012047
1	em xin góp ý kiến như thế	này: "mong hai cô sẽ gửi ý kiến phản hồi của ...	này, mong nhà trường chỉnh sửa lại, cho dạy t...	3.714084	4.158318
2	thầy dạy hay , vui nhiều phần	nghe lý thuyết rất hay! thầy hài hước vui tín...	nghe còn bị câu hỏi lỗi nhỏ thì không rõ hẳn ...	-2.074794	3.635764
3	có những cái cô dạy	hay mà chấm bài dài quá! chúng em cần biết nh...	hơi khó hiểu. thực tế là thầy nên để lớp của ...	3.614214	4.205528
4	như vậy , sinh viên làm đồ	án cũng cần có giảng viên hướng dẫn nên áp dụ...	án thì muốn làm gì thì làm. không hỗ trợ sinh...	4.115233	4.079134
5	thầy rất nhiệt	tình, giảng lại bài rất chi tiết. thầy có khá...	tình! bài giảng chưa hiệu quả. chưa cải thiện...	-1.939914	4.109962
6	chuẩn bị bài giảng hoặc	kiểm tra lại cho phần thi hoàn chỉnh của sinh...	chat dài hơi hơn trên toast nên làm bài tập v...	3.892115	4.119471
7	học được nhiều	về excel cả, thầy nhiệt tình, truyền đạt kinh...	từ vựng tiếng anh hơn. không cho bài tập và k...	-1.866759	4.170439

Sample[5] (before)

'thầy rất nhiệt tình, giảng lại bài rất chi tiết. thầy có khá nhiều kiến thức về môn học. thầy đọc rất kĩ'

Sample[5] (after)

'thầy rất nhiệt tình! bài giảng chưa hiệu quả. chưa cải thiện được khả năng giảng dạy kỹ càng. giảng viên đưa'

```
mean:

rewards (before)    1.365477
rewards (after)     4.061333
dtype: float64


median:

rewards (before)    2.541926
rewards (after)     4.114716
dtype: float64
```

Figure 6: Example responses before (top) and after (bottom) PPO fine-tuning with the PhoBERT reward model.

D. Group Relative Policy Optimization (GRPO)

Before diving deep into the GRPO algorithm, we need to have a basic knowledge about Chain-of-Thought first, which is also known as reasoning through prompting.

1) Chain-of-Thought Prompting: Chain-of-Thought (CoT) prompting elicits intermediate reasoning steps by instructing LLMs to “think step by step” before answering, significantly boosting performance on arithmetic and symbolic tasks [2]. However, CoT outputs can hallucinate unsupported steps and exhibit inconsistent reasoning chains across different prompts, limiting reliability [3].

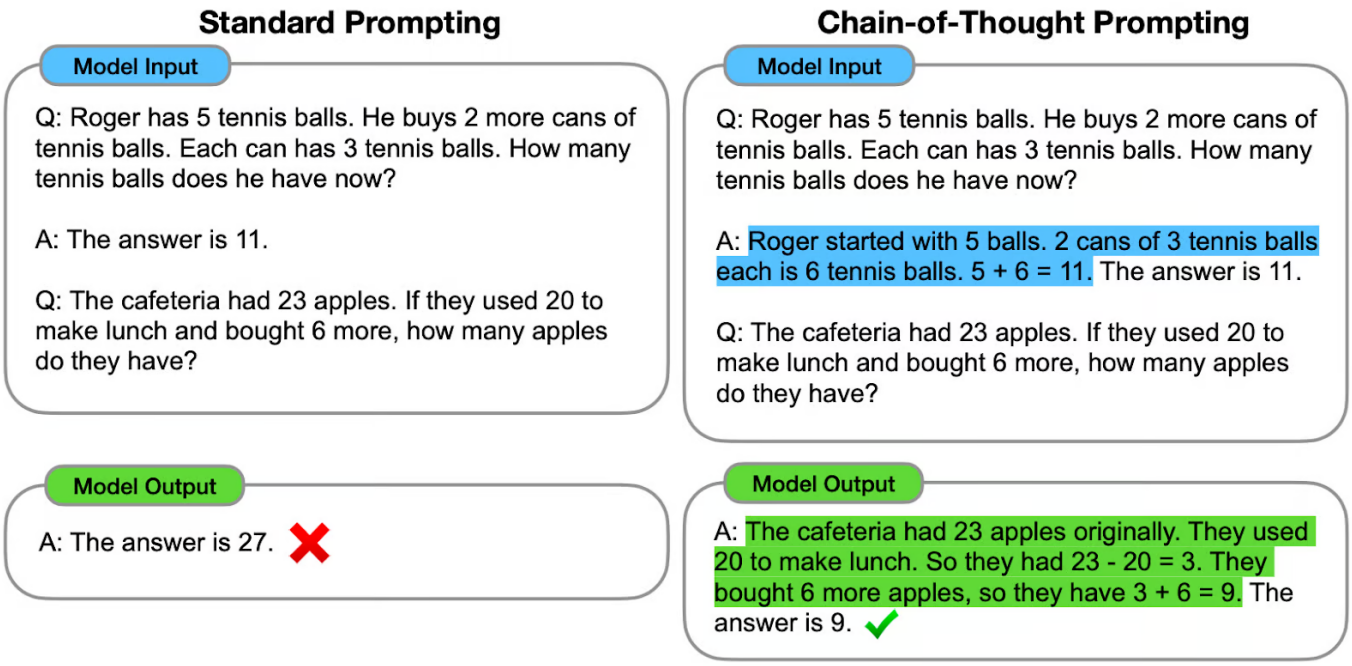

Figure 7: Example of with and without CoT prompting.

2) Beyond CoT: Self-Consistency and Tree-of-Thought:

a) Self-Consistency: Self-Consistency augments CoT by sampling multiple reasoning paths and selecting the most frequent final answer, yielding substantial accuracy gains on benchmarks like GSM8K. However, it introduces significant computational overhead—requiring dozens of extra forward passes per query—which can be prohibitive in latency-sensitive or resource-constrained settings. Moreover, while majority voting reduces variance in final answers, it does not eliminate hallucinated or logically inconsistent intermediate steps, and may simply mask underlying reasoning failures rather than correct them.

b) Tree-of-Thought (ToT): ToT generalizes CoT by structuring exploration as a search tree over "thought" nodes, enabling lookahead, backtracking, and global planning. This framework can dramatically improve performance on planning-intensive tasks (e.g., Game of 24), but at the cost of combinatorial explosion: the number of candidate paths grows exponentially with search depth, leading to high memory usage and runtime unless strong heuristics prune the search effectively. Additionally, designing and tuning these heuristics often requires task-specific expertise, limiting the method's out-of-the-box applicability.

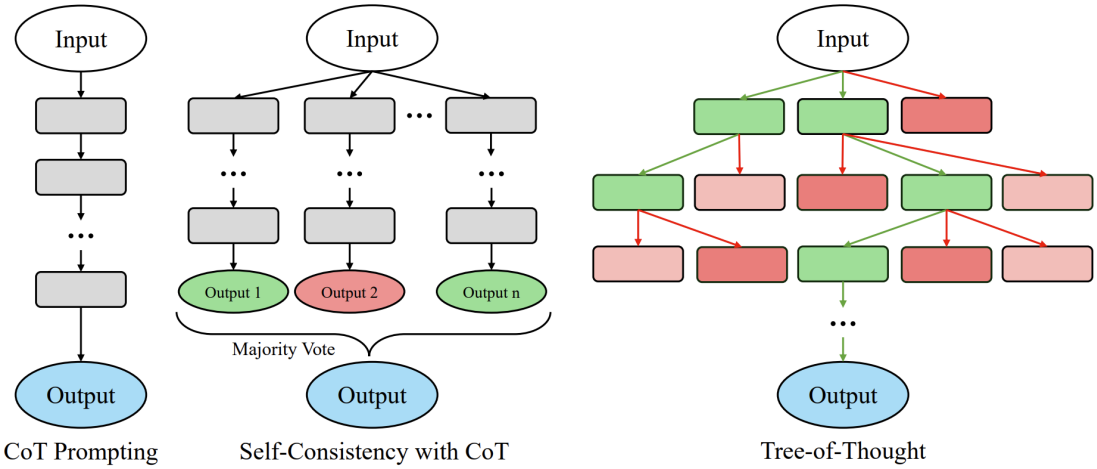

Figure 8: Some Chain-of-Thought variants (also known as reasoning via prompting).

By surfacing these limitations of prompt-based CoT variants, we motivate the need for native reasoning fine-tuning methods—such as Group Relative Policy Optimization—that enable LLMs with improved multi-step logic capabilities without

relying on expensive or brittle prompt engineering.

3) GRPO: Enhancing Native Reasoning: Building on PPO's clipped objective [6], Group Relative Policy Optimization (GRPO) removes the separate critic network by normalizing rewards within each batch group.

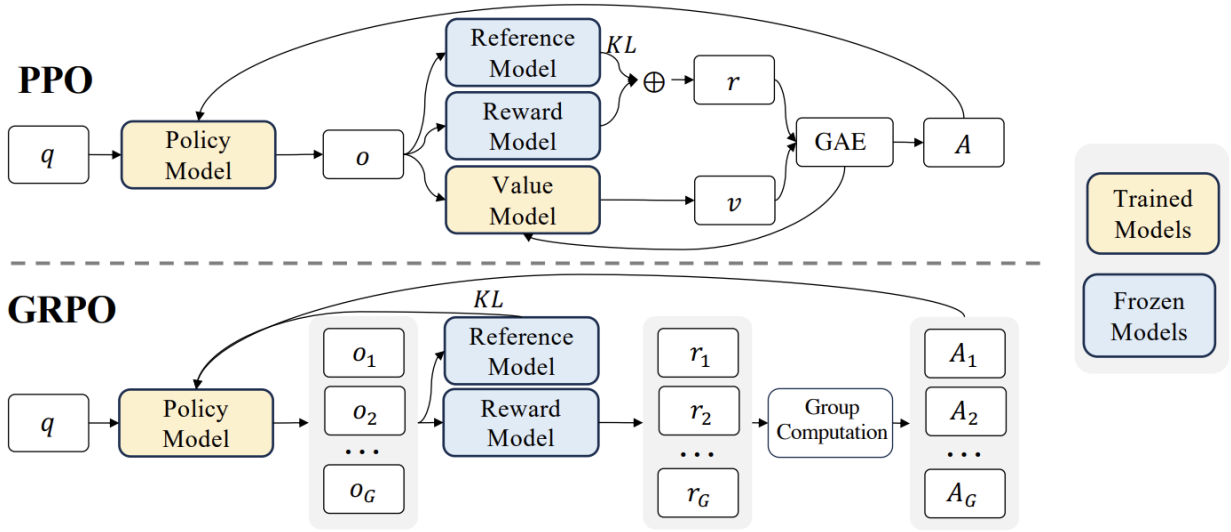

Figure 9: GRPO foregoes the value model, instead estimating the baseline from group scores, significantly reducing training resources.

The GRPO loss is:

$$J_{\text{GRPO}}(\theta) = \mathbb{E}_g\Big[\sum_{t=1}^{T} \min\left(r_t^g(\theta)\,\hat{A}_t^g,\ \text{clip}(r_t^g(\theta), 1-\epsilon, 1+\epsilon)\,\hat{A}_t^g\right) - \beta D_{\text{KL}}(\pi_\theta(\cdot|s_t)||\pi_{\theta_{\text{ref}}}(\cdot|s_t))\Big] \tag{3}$$

where

- θ are the policy parameters. θ_{old} are the parameters before the update. θ_{ref} represents the parameters of a reference policy (the original version of policy before any update happened).
- $\mathbb{E}_g[\dots]$ denotes the empirical average over groups of generated trajectories (responses).
- t indexes the time step within a trajectory (token generation step).
- T is the length of the generated trajectory (the complete sentence).
- $r_t^g(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio of the action a_t at state s_t under the current policy π_θ and the old policy $\pi_{\theta_{\text{old}}}$.
- $\hat{A}_t^g = \frac{R_i^g - \text{mean}(R^g)}{\text{std}(R^g)}$ is the group-normalized advantage at time step t for the i-th trajectory in group g.
  - R_i^g is the reward obtained upon the completion of the entire generated sentence for the i-th trajectory in group g. This reward evaluates the quality of the full response.
  - $\text{mean}(R^g)$ is the mean of the rewards of all completed sentences within the group g.
  - $\text{std}(R^g)$ is the standard deviation of the rewards of all completed sentences within the group g.
- $\text{clip}(x, 1-\epsilon, 1+\epsilon)$ is the clipping function.
- ϵ is the clipping hyperparameter.
- $D_{\text{KL}}(\pi_\theta(\cdot|s_t)||\pi_{\theta_{\text{ref}}}(\cdot|s_t))$ is the KL divergence between the current and reference policies at state s_t. This term is also compensating for the oversimplification like removing the value model which introduces more variance compared to PPO.
- β is the coefficient for the KL divergence penalty.

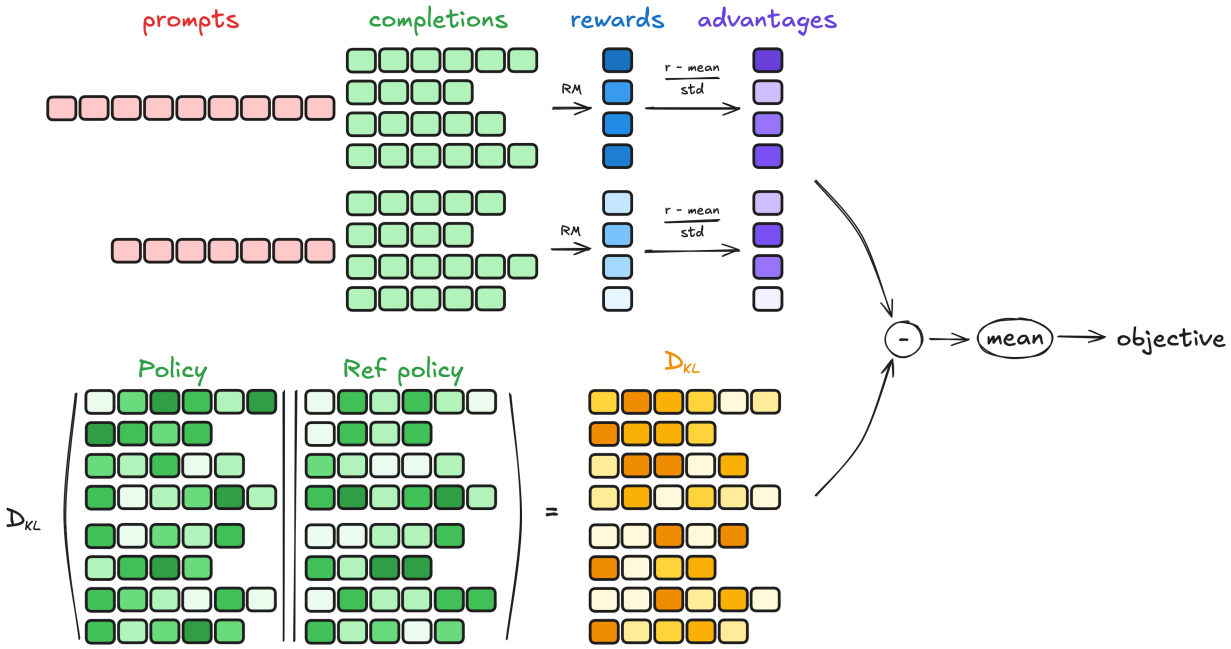

Figure 10: A simple GRPO optimization step in practice.

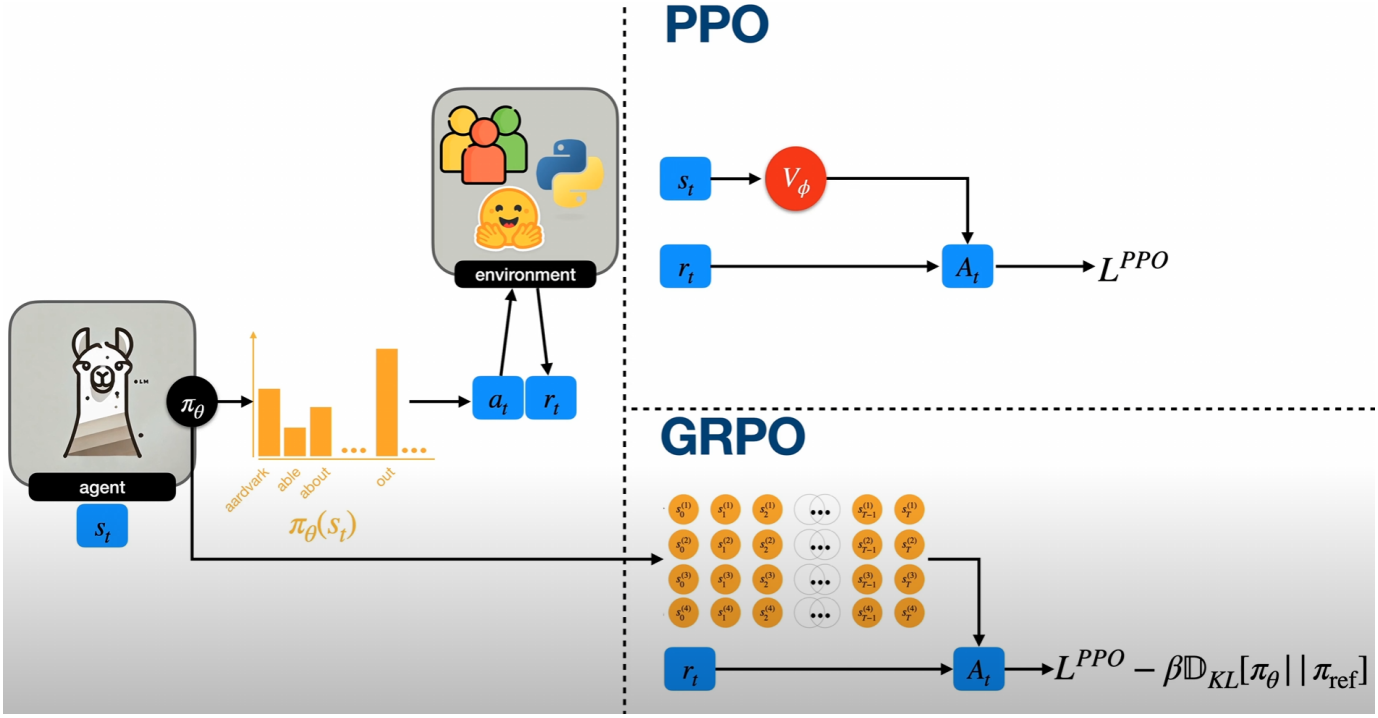

Figure 11: Wrap up: PPO vs GRPO. source

III. Different Approaches for Math QA Problem

A. *Byte-Pair Encoding Tokenization*

Most modern LLMs use Byte-Pair Encoding (BPE) to build a subword vocabulary that balances model capacity with open-vocabulary coverage. BPE iteratively merges the most frequent pairs of characters or subwords into new tokens, enabling the model to represent rare or novel words as compositions of known subwords while keeping the overall vocabulary size manageable.

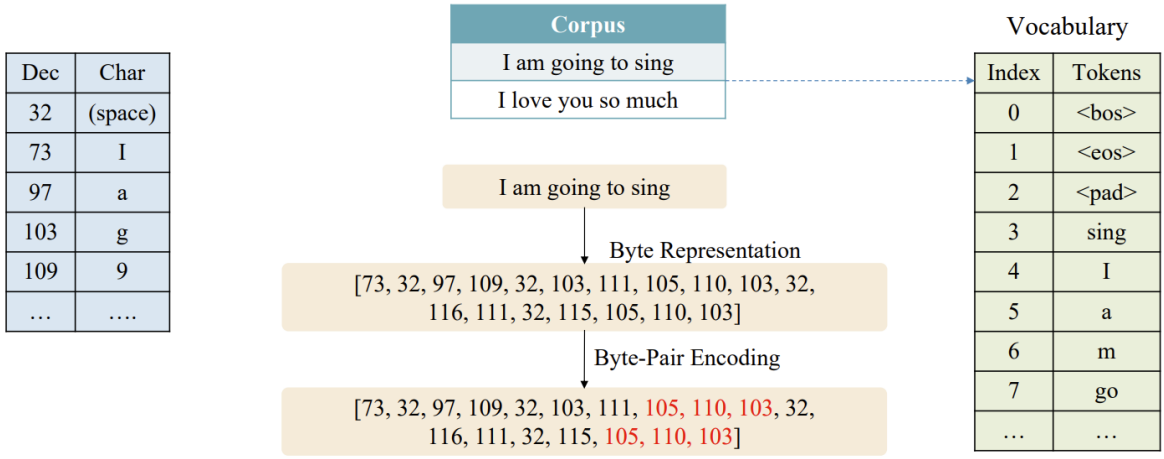

Figure 12: Build a vocabulary using BPE Tokenizer

B. The vi_gsm8k Dataset

The vi_gsm8k dataset is a Vietnamese translation of the GSM8K benchmark, consisting of 7,500 grade-school math word problems with step-by-step solutions. Each example includes:

- A natural-language problem statement in Vietnamese.
- A gold-standard chain of reasoning (solution steps).
- A final numeric answer.

We sample 100 problems from the official test split for our quantitative evaluation.

C. Model Architectures

1) Encoder–Decoder Transformers:

- **mT5-small:** A pretrained multilingual T5 model fine-tuned on vi_gsm8k.
- **Scratch Enc–Dec:** An encoder–decoder Transformer of similar size, randomly initialized and trained from scratch on vi_gsm8k.

2) Decoder-Only Models:

- **GPT2-medium:** A pretrained GPT2 fine-tuned on vi_gsm8k with supervised learning.
- **Scratch Dec-Only:** A GPT2-sized decoder model initialized at random and trained from scratch on vi_gsm8k.

*3) **Qwen 2.5-3B with Instruction Tuning (QLoRA + CoT):*** We apply Low-Rank Adaptation (LoRA) and its quantized extension QLoRA to fine-tune Qwen 2.5-3B with Chain-of-Thought prompts:

- **LoRA:** Injects low-rank adapters into attention and MLP layers for parameter-efficient fine-tuning (PEFT technique).

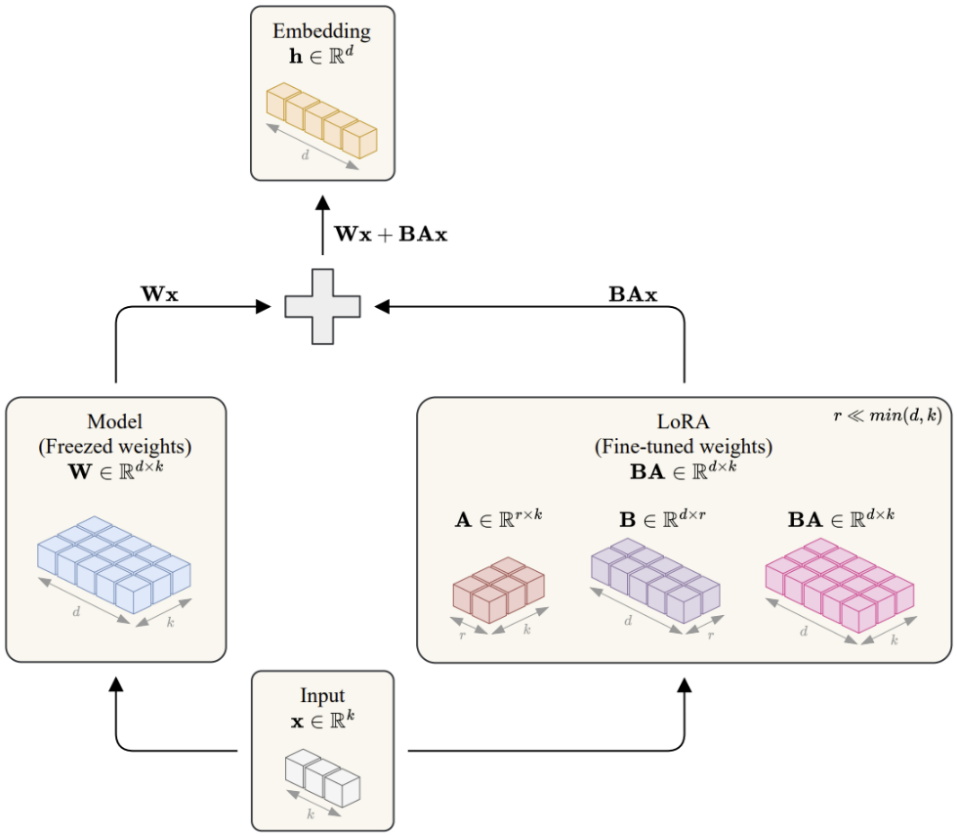

Figure 13: LoRA flow.

- **QLoRA:** Further reduces memory footprint via 4-bit quantization (nf4) and a paged optimizer for large-scale adapters.

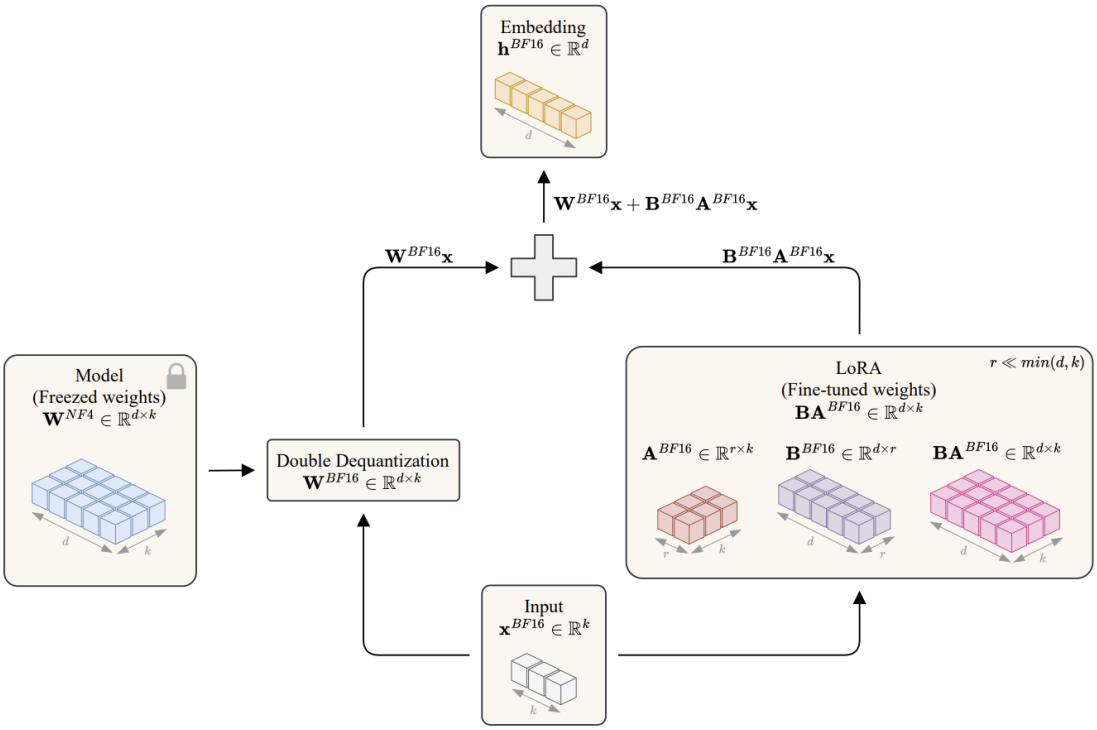

Figure 14: QLoRA flow.

```
SYSTEM_PROMPT = """
Bạn là một trợ lý toán học.
Hãy suy nghĩ cẩn thận để giải bài toán được cung cấp.
Trình bày từng bước. Sau đó trả lời theo cú pháp:
### number
"""
```

*4) **Qwen 2.5-3B with GRPO (QLoRA)**:* We are training Qwen 2.5-3B using Group Relative Policy Optimization with a simplified prompt and let the model decide all the reasoning steps for itself:

```
SYSTEM_PROMPT = """
Respond in the following format:
<reasoning>
...
</reasoning>
<answer>
...
</answer>
"""
```

Question: "Có tất cả bao nhiêu kí tự 'r' trong từ 'waterrmelon' ?"

Before GRPO:

Trong từ "watermelon", có 1 ký tự \'r\'.

After GRPO:

'<reasoning>
Khi phân tích từ "waterrmelon", ta thấy nó có một lỗi chính tả (do có hai chữ \'r\'). Mặc dù vậy, ta cần phân tích từ nguyên vẹn là "watermelon". Từ "watermelon" chỉ có một ký tự \'r\'. Ký tự \'r\' xuất hiện hai lần trong từ này (ở vị trí thứ 1 và thứ 8). Tuy nhiên, theo yêu cầu, ta cần đếm tất cả các ký tự \'r\' nên kết quả sẽ là hai ký tự.
</reasoning>
<answer>
2
</answer>'

Figure 15: Before and After training Qwen 2.5-3B with GRPO

D. Evaluation Metrics

All models are evaluated on 100 sampled test problems using four metrics:

- **Perplexity:** Measures the model's confidence in the generated token sequence; lower is better.
- **SacreBLEU:** Computes n-gram overlap between generated and reference solutions, standardized for reproducibility.
- **ROUGE-1/2/L F1:** Captures unigram, bigram, and longest common subsequence overlap to assess content fidelity.
- **Exact Match Score:** The percentage of problems where the final numeric answer exactly matches the reference.

These metrics together assess fluency, content overlap, and strict answer correctness.

IV. Results and Discussion

A. *Quantitative Results*

Table I: Perplexity, SacreBLEU and Exact Match scores on 100 sampled vi_gsm8k problems

Model	Perplexity	SacreBLEU	Exact Match
mT5-small	3.9578	20.1683	0.0500
Encoder–Decoder (scratch)	11.7095	39.9151	0.0200
GPT2-medium	5.1044	30.1349	0.0300
Decoder-only (scratch)	19.5144	27.5626	0.0100
Qwen 2.5-3B (Instruction + CoT, QLoRA)	1.7495	21.1412	0.4700
Qwen 2.5-3B (GRPO, QLoRA)	N/A	31.5785	0.6700

Table II: ROUGE-1/2/L F1 scores on the same 100 problems

Model	ROUGE-1 F1	ROUGE-2 F1	ROUGE-L F1
mT5-small	0.4530	0.2061	0.3332
Encoder–Decoder (scratch)	0.4548	0.1955	0.3222
GPT2-medium	0.4689	0.2245	0.3478
Decoder-only (scratch)	0.4649	0.1932	0.3241
Qwen 2.5-3B (Instruction + CoT, QLoRA)	0.5971	0.3259	0.4308
Qwen 2.5-3B (GRPO)	0.5475	0.2972	0.3795

B. *Metric Reliability and Model Behavior*

While SacreBLEU and ROUGE scores provide standardized measures of n-gram overlap, they can be unreliable for math QA, where correct solutions often use diverse phrasing or intermediate steps that differ from the reference. In particular, high-perplexity models sometimes achieve inflated BLEU/ROUGE by repeating common formulaic phrases (e.g., "Ta có:", "Bước 1, 2, 3…"), without genuine reasoning.

The Exact Match score—strict matching of the final numeric answer—offers a more faithful indicator of correctness, but it ignores the quality and interpretability of the solution steps. Perplexity alone reflects fluency and confidence but does not guarantee logical validity.

C. *Impact of RL Fine-Tuning on Reasoning*

The leap in Exact Match from **47.0%** for Qwen-instruct (CoT) to **67.0%** for Qwen-GRPO demonstrates the potential of reinforcement-learning fine-tuning to endow LLMs with stronger native reasoning capabilities. By optimizing directly for solution correctness rather than relying on prompt engineering, GRPO-tuned models produce more accurate and coherent multi-step solutions. This suggests a promising direction: integrating RL-based reasoning fine-tuning into LLM workflows to surpass the limitations of prompt-only methods.

References

[1] Schulman, John and Wolski, Filip and Dhariwal, Prafulla and Radford, Alec and Klimov, Oleg. "Proximal Policy Optimization Algorithms." arXiv preprint arXiv:1707.06347, 2017.

[2] Wei, Jason; Wang, Xuezhi; Schuurmans, Dale; Bosma, Maarten; Ichter, Brian; Xia, Fei; Chi, Ed H.; Le, Quoc V.; Zhou, Denny. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models." arXiv preprint arXiv:2201.11903, 2022.

[3] DataCamp. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models." Tutorial @ DataCamp, 2024. https://www.datacamp.com/tutorial/chain-of-thought-prompting

[4] Wang, Xuezhi; Wei, Jason; Schuurmans, Dale; Le, Quoc; Chi, Ed; Narang, Sharan; Chowdhery, Aakanksha; Zhou, Denny. "Self-Consistency Improves Chain of Thought Reasoning in Language Models." arXiv preprint arXiv:2203.11171, 2022.

[5] Yao, Shunyu; Yu, Dian; Zhao, Jeffrey; Shafran, Izhak; Griffiths, Thomas L.; Cao, Yuan; Narasimhan, Karthik. "Tree of Thoughts: Deliberate Problem Solving with Large Language Models." arXiv preprint arXiv:2305.10601, 2023.

[6] Schulman, John; Wolski, Filip; Dhariwal, Prafulla; Radford, Alec; Klimov, Oleg. "Proximal Policy Optimization Algorithms." arXiv preprint arXiv:1707.06347, 2017.

[7] Shao, Zhihong; Wang, Peiyi; Zhu, Qihao; Xu, Runxin; Song, Junxiao; Bi, Xiao; Zhang, Haowei; Zhang, Mingchuan; Li, Y.K.; Wu, Y.; Guo, Daya. "DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models." arXiv preprint arXiv:2402.03300, 2024.