E.2 Verifier Performance

We report the performance of the verifier in Table 6.

	GSM8k	ANLI
Accuracy	0.91	0.92

Table 6: Performance of the verifier on different benchmarks. Accuracy is reported for GSM8K and ANLI. Notably, the verifier demonstrates higher accuracy in evaluating the correctness of answers compared to the accuracy of the LLM in generating correct answers. The verifier is classified as correct if the assigned reward is greater than 0.5 when the LLM-generated solution is correct, or if the reward is less than 0.5 when the LLM-generated solution is incorrect.

E.3 Other Observations

We experimented with various verifiers built upon different language model bases. Our first observation was that training the model using only the final answer did not perform as well as minimizing the cross-entropy loss over the last half of the generated tokens. Second, the verifier produced interpretable results, aligning with findings from Liu et al. (2023). Lastly, when we used training samples from one base model but trained the verifier with a different base model as the backbone, the loss did not decrease, indicating that using the same base model for training is crucial for effective learning.

E.4 Proof of Theorem 1

Theorem 1. *Assuming the verifier model is sufficiently expressive, the optimal parameter $\theta^\star$ that minimizes the expected cross-entropy loss between the true label and the verifier's output will satisfy*

$$\texttt{Verifier}_{\theta^\star}(q, s^{1:x}) = \mathbb{P}(\textit{Final answer is correct} \mid q, s^{1:x}).$$

Proof. The expected loss can be written as

$$\mathcal{L}(\theta) = \mathbb{E}_{q,s,x,y}\Big[y \log \texttt{Verifier}_\theta(q, s^{1:x}) + (1-y)\log(1 - \texttt{Verifier}_\theta(q, s^{1:x}))\Big].$$

Defining $p_\theta(q, s^{1:x}) := \texttt{Verifier}_\theta(q, s^{1:x})$, we compute the partial derivative with respect to $p_\theta(q', s^{1:x'})$:

$$\mathbb{E}_{q,s,x,y}\Big[\mathbf{1}\big(q = q', s^{1:x} = s^{1:x'}\big)\Big(\frac{y}{p_\theta(q, s^{1:x})} - \frac{1-y}{1 - p_\theta(q, s^{1:x})}\Big)\Big],$$

so we conclude

$$\texttt{Verifier}_{\theta^\star}(q, s^{1:x}) = \mathbb{E}[y \mid q, s^{1:x}] = \mathbb{P}(\text{Final answer is correct } \mid q, s^{1:x})$$

since we assumed that the verifier model is sufficiently expressive. □

It is worth noting that while Yu et al. (2024) provided a similar analysis using an ℓ_2-loss function, we extend the analysis to the entropy loss function, which is commonly used in classification tasks.

F Various Reward Function Designs

We can shape the reward function with verifiers in several different ways, with the following designs of the reward function.

- **Immediate Verification Reward**: The immediate verification reward is defined as $R_{\boldsymbol{\theta}}(q, s_{ta}) = \mathbb{E}[\text{Verifier}(q, s_{ta})]$. This reward is based on the verifier's immediate evaluation of the solution s_{ta} at turn t for agent a. It reflects the instantaneous correctness of the solution without considering future steps or contributions from other agents.

- **Cumulative Verification Reward**: The cumulative verification reward is given by

$$R_{\boldsymbol{\theta}}(q, s_{ta}) = \mathbb{E}\left[\frac{1}{\sum_{t' \in [t,T]} \gamma^{t'-t}} \sum_{t' \in [t,T]} \gamma^{t'-t} \text{Verifier}(q, s_{t'a})\right]. \tag{1}$$

  Here, the reward accounts for the verifier's evaluations across all remaining turns from t to the final turn T. The term $\gamma^{t'-t}$ represents a discount factor that prioritizes earlier rewards. This cumulative approach encourages solutions that not only perform well in the immediate turn but also lead to favorable outcomes in subsequent turns.

- **Influence-aware Verification Reward**: The influence-aware verification reward function is defined as

$$R_{\boldsymbol{\theta}}(q, s_{ta}) = \mathbb{E}\left[\frac{1}{\sum_{t' \in [t,T]} \gamma^{t'-t}} \left(\text{Verifier}(q, s_{ta}) + \sum_{t' \in [t+1,T]} \sum_{j \in [A]} \frac{1}{A} \gamma^{t'-t} \text{Verifier}(q, s_{t'j})\right)\right].$$

  This reward not only considers the verifier's score for the current solution s_{ta} but also incorporates the impact of this solution on the future answers of all agents. The term $\sum_{j \in [A]} \frac{1}{A}$ averages the verifier's scores across all agents, reflecting the influence that s_{ta} has on the collective progress of the multi-agent system.

G Training Details of MAPoRL

G.1 Efficient Network Architecture for MAPoRL

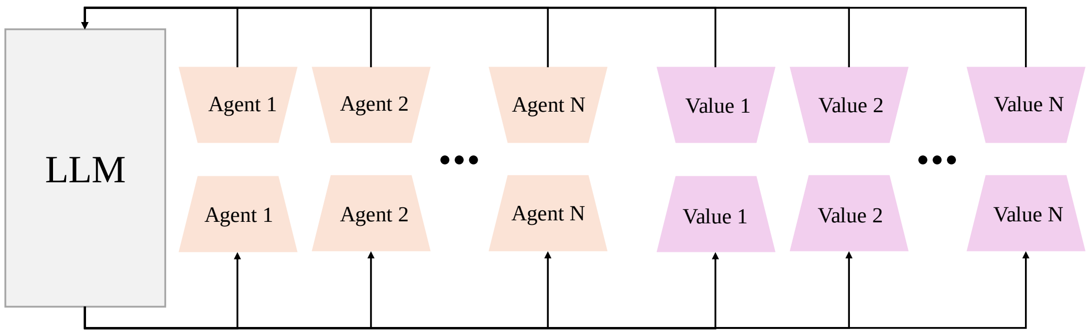

Figure 6: We utilized various QLoRA adapters to implement multiple LLM agents and value functions simultaneously. Each agent and value function comprises less than 0.2% of the parameters of the base LLM model. For the value function, we employed a QLoRA fine-tuned model with a value head.

As we incorporate multiple language models in the training process, we need to implement them efficiently to fit within the limited resources of GPU memory. Our default setup is as follows: First, we implemented multiple language models using the same base model architecture, augmented with QLoRA adapters. Second, for constructing the value model, we employed pretrained LLMs, which was further fine-tuned by adding an additional linear head layer. Please refer to Figure 6 for an overview of the network architecture.

Remark 4 (Input of Value Functions). *The function $V_{\theta ta}$ is dependent on i_{ta}^{x}, where i_{ta}^{x} includes the question and the history of $s_{t'j}$ for $t' \leq t-1$ and $j \in [A]$. For simplicity, we assume that s_{ta} contains the necessary information from $s_{t'j}$ for $t' < t$, which allows us to simplify the input to the value function as $q \oplus s_t^{1:x}$.*

G.2 Experimental Setup and Hyperparameter Configuration for MAPoRL

For every experiments, we set the hyperparameter as follows: For the verifier score of each agent's answer per turn, we set `non-eos-penalty` and `non-box-penalty` to *True*, ensuring that answers without

\\boxed{} are penalized with a verifier score of -10. We enforced a minimum output length of 50 and used $\gamma = 1$ for the cumulative verification reward (see Equation (1)). The training was conducted on 8 NVIDIA A100-80GB GPUs, while for episode generation, 12 episodes were processed simultaneously. In the multi-agent PPO update, we set the batch size to 1 with a gradient accumulation step of 4, and each trajectory rollout was iterated four times for multi-agent PPO updates. For language model generation, we used a temperature of 0.7. Additionally, for QLoRA configuration, we set $r = 8$, $\alpha = 16$, and dropout rate of 0.05. The AdamW optimizer (Loshchilov and Hutter, 2019) was used with $\beta_1 = 0.9$ and $\beta_2 = 0.95$, along with a learning rate of 1.0×10^{-5} and a warmup step of 10. For the value penalty term and KL penalty term, we set $\lambda_{\text{KL}} = 2 \times 10^{-4}$ and $\lambda_{\text{value}} = 0.1$.

G.3 Engineering Challenges and Solutions

G.3.1 Addressing Reward Hacking

A key advantage of our verifier approach is that, given a perfect verifier, we can operate without final answer labels—requiring only quality problems for the multi-agent system. This capability is particularly valuable for large-scale training or online learning scenarios (such as ChatGPT's user inputs which does not have a golden answer), where golden answers may be unavailable. However, reward hacking remains a persistent challenge, both in traditional RL problems (Amodei et al., 2016; Hadfield-Menell et al., 2017) and increasingly in LLM development. For instance, the recent Deepseek R1 Model (Guo et al., 2025) avoided verifiers entirely to prevent reward hacking, instead requiring answer labels for all questions and implementing manual criteria with special tokens (e.g., "think" tokens) in their reward function. In our work, we encountered and addressed several reward hacking scenarios, significantly reducing their occurrence in our final system.

Insufficient Reasoning in Short Answers. Initially, we observed that **MAPoRL** produced overly concise answers when constrained only by non-eos and non-boxed penalties. We addressed this by implementing a penalty for responses shorter than 50 tokens. However, LLMs occasionally circumvented this by using alternative end tokens.

For the ANLI dataset specifically, where models produced meaningless text despite length requirements, we introduced a reasoning-quality verification prompt. This prompt evaluated the presence of proper reasoning (independent of answer correctness) and proved effective. Notably, this issue did not manifest in mathematical reasoning tasks.

Token Repetition. Repetitive token sequences are a known issue in language model outputs (Holtzman et al., 2020). We observed instances of 2-5 token repetitions in our trained outputs. Our solution implemented a manual penalty of -10 for sequences repeating more than three consecutive times, excluding numeric values where repetition might be valid.

Post-boxed Token Generation. Models attempted to exploit the reward system by adding arbitrary tokens or punctuation after \\boxed{}. We addressed this by introducing penalties for any token generation following the boxed expression.

G.3.2 Evaluation Format Standardization

To address concerns that performance improvements might stem from formatting rather than reasoning capabilities, we implemented a robust evaluation methodology. Our approach incorporated a post-processing step using an LLM to extract final answers, eliminating format-induced evaluation errors. This standardization ensures that performance metrics reflect actual reasoning and collaboration ability rather than formatting proficiency.

G.4 Prompt Design for Collaborative Debate

G.4.1 Turn 1 Prompt

GSM8k and TinyGSM.

```
{"role": "user", "content": f"' Question: {sample["question"]}

```