MALT: Improving Reasoning with Multi-Agent LLM Training

Anonymous Authors[1]

Abstract

Large Language Models (LLMs) often produce answers with a single chain-of-thought, which restricts their ability to explore reasoning paths or self-correct flawed outputs in complex tasks. In this paper, we introduce MALT (Multi-Agent LLM Training), a novel post-training strategy that divides the reasoning process into generation, verification, and refinement steps using a sequential pipeline of heterogeneous agents. During data generation, each agent is repeatedly sampled to form a multi-agent search tree, where final outputs are graded against ground-truth data. We then apply value iteration to propagate reward signals back to each role-conditioned model, automatically producing multi-agent post-training data without human or teacher-model supervision. Our off-policy approach allows each agent to specialize by learning from correct and incorrect trajectories, ultimately improving the end-to-end reasoning chain. On MATH, GSM8K, and CSQA, MALT surpasses the same baseline LLM with a relative improvement of 15.66%, 7.42%, and 9.40% respectively, making it an important advance towards multi-agent cooperative training.

1. Introduction

Reasoning with Large Language Models (LLMs) is inherently challenging, particularly for tasks that require multi-step deductions, intermediate computations, or self-correction (Xiang et al., 2025). Recent work on multi-agent approaches—such as debate frameworks (Du et al., 2024) or orchestrated problem-solving (Fourney et al., 2024)—has shown promise by assigning different parts of the reasoning process to separate models, allowing for refinement and increased deliberation at inference time (Snell et al., 2024). However, the underlying LLMs are typically the same base model pre-trained on data that lacks exposure to the specialized roles or meta-strategies involved in solving complex problems, which introduces a distribution shift while reasoning at test-time (Xiang et al., 2025; Han et al., 2024). An open gap thus persists: *How can we jointly train LLMs to specialize in a multi-agent setting to improve reasoning?* Such a gap persists due to several key obstacles: First, it is difficult to propagate gradient signals through multiple discrete token-outputs during training. Second, there is little role-specific labeled training data. Third, credit assignment in such setups is difficult when only the final outcome receives rewards (Tumer & Agogino, 2007). Finally, it is important to determine what type of multi-agent setup is even useful to implement meta-strategies that can improve reasoning so that more inference compute can be spent efficiently towards solving problems.

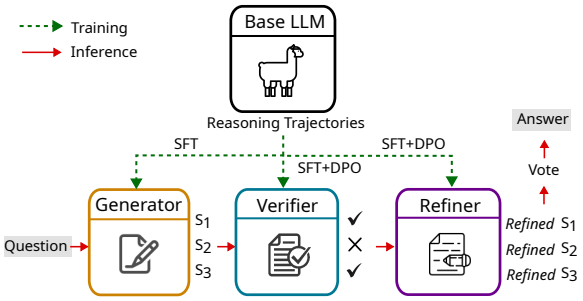

Figure 1. **Overview of MALT.** During joint-training we use a base LLM (eg. Llama) to generate several reasoning trajectories. These trajectories are used to create 3 specialized agents, a generator, verifier and refiner through SFT and DPO training. During inference, the 3 agents work together to answer a given question.

In this paper, we address these challenges by introducing an intuitive strategy to jointly post-train LLMs in a generate-critique-refine pipeline. This is analogous to how humans tackle complex tasks—drafting an initial answer, thoroughly checking and critiquing it, and refining the solution to match their specifications (Qian et al., 2024). We propose a method, visualized in Figure 1, that automatically generates large-scale, labeled data for each agent via a multi-agent credit assignment approach. With this dataset, our post-training approach enables role-conditioned models to learn from both positive and negative reasoning trajectories—providing a path to improve reasoning performance

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

across a range of problems with trained multi-agent setups.

To the best of our knowledge, we are the first to address this critical gap with Multi-Agent LLM Training (MALT): a strategy involving three models working sequentially on problems—a generator, a verifier, and a refinement model.

MALT employs a sampling procedure that expands a search tree based on each model's outputs with an exponential branching factor, thereby producing large amounts of useful synthetic data. Role-specific data, particularly when augmented with rationales, has been shown in previous work to significantly improve performance (Zelikman et al., 2022). However, this introduces a credit assignment problem where internal branches may be correct or incorrect and must be labeled to post-train models based solely on final output signals. To address this, we propose a value-iteration-based attribution strategy (Sutton & Barto, 2018). By analyzing only the outputs of the search tree, our method identifies which model introduced an error, enabling credit assignment without requiring human-labeled data or a separate oracle policy to evaluate trajectories. This eliminates the need for human intervention in selecting trajectories, generating role-specific data, or designing value functions, and instead enables the automatic generation of reasoning traces from the search tree for post-training via supervised fine-tuning and preference optimization (Rafailov et al., 2023). MALT combines all these steps into an intuitive joint-training procedure, providing performance improvements and acting as an early first step towards integrating search and learning for multi-agent LLM systems.

Our contributions are threefold:

- We propose Multi-Agent LLM Training (MALT) to cooperatively post-train a generator, verifier, and refinement model on challenging reasoning tasks by leveraging search and subsequent fine-tuning.
- We propose a search-tree expansion process and use a value iteration technique to propagate binary outcome rewards to automatically attribute correct and incorrect reasoning traces to individual models. This, requiring no human supervision, can be utilized for supervised fine-tuning and preference optimization.
- We apply MALT to math and common sense reasoning questions from MATH, CSQA, and GSM8K, achieving relative improvements of 15.66%, 9.40%, and 7.42% over a single-model baseline, as well as outperforming other comprehensive baselines and ablations.

2. Related Work

Advanced Reasoning: Multi-agent architectures have emerged as an effective way to handle complex reasoning by distributing problem-solving roles among distinct models (Xiang et al., 2025; Cobbe et al., 2021). Frameworks such as Agent Q (Putta et al., 2024) and AutoGen (Wu et al., 2024) coordinate solutions through guided search and compositional dialogue, yet they often rely on a single underlying model for multiple tasks or omit a dedicated verification mechanism. Debate-style (Du et al., 2024) and orchestrated (Fourney et al., 2024) multi-agent methods assign different parts of the reasoning process to separate models, enabling increased deliberation at inference time (Snell et al., 2024). However, the underlying models lack advanced training for role specification, which may lead to suboptimal performance or distribution shifts during test-time (Xiang et al., 2025). In contrast, single-agent approaches utilize techniques such as introspection (Qu et al., 2024) and self-critique (Saunders et al., 2022) to detect their own errors, but they typically lack the mechanisms to correct those errors in a single inference pass (Ye et al., 2024).

Training Data and Preference Optimization: Harnessing synthetic data generation in tandem with preference-based training has emerged as a potent strategy for boosting LLM performance. Setlur et al. (2024) demonstrates that training on both correct and incorrect synthetic solutions, optimized with Direct Preference Optimization (DPO) (Rafailov et al., 2023), significantly improves math reasoning accuracy. Singh et al. (2024) propose a self-training method (ReSTEM) that repeatedly generates, filters, and fine-tunes using scalar feedback, surpassing purely human-based fine-tuning on tasks like advanced math and coding. Parallel work by Pang et al. (2024) shows how preference signals applied across entire chains-of-thought can refine intermediate reasoning steps. Our method builds on these insights by generating large volumes of role-specific data, both correct and incorrect, and uses preference-like signals to train each agent.

Inference Time Compute: Strategic use of inference-time compute can also boost accuracy. Brown et al. (2024) demonstrates that repeated sampling from a single model improves coverage, while Snell (Snell et al., 2024) shows that iterative refinement can outperform naive scaling in certain tasks. Meanwhile, supervised fine-tuning (SFT) remains the backbone of LLM adaptation (Devlin et al., 2019; Howard & Ruder, 2018), but can demand large volumes of human-labeled data. Quasi-supervised methods (Yang et al., 2024) address data scarcity by providing supervision for intermediate steps. Finally, Zelikman et al. (2022) and Xiang et al. (2025) underscore the effectiveness of structured chain-of-thought prompting for complex tasks, while Cobbe (Cobbe et al., 2021) confirms that an explicit verifier stage reduces errors on GSM8K. Our work unifies the aforementioned techniques to create a multi-agent pipeline that orchestrates a generator, verifier, and refinement model, leveraging search, synthetic data generation, and preference-based training to enable more robust multi-step reasoning.

3. Preliminaries

Chain of Thought (CoT) When an LLM solves a question q with step-by-step reasoning $(s_1, \ldots, s_n)$, the answer a can be viewed as sampled from:

$$p_d(a \mid q) \propto \int p_d(a \mid s_{1:n}, q) \prod_{t=1}^{n} p_d(s_t \mid s_{<t}, q)\, dS$$

as described in (Xiang et al., 2025). For complex problems, the same work describes a true solution-generating process that involves extra latent steps $(z_1, \ldots, z_k)$, which is a meta-generalization of the prior CoT argument:

$$p_d(a, s_{1:n} \mid q) \propto \int p_d(a, s_{1:n} \mid z_{1:k}, q) \prod_{t=1}^{k} p_d(z_t \mid z_{<t}, q)\, dZ$$

These meta-variables could capture iterative or corrective steps, unfolding a more adaptive, multi-stage reasoning trajectory. In a multi-agent setup, we consider a process with three specialized models: a Generator G, a Verifier $\mathcal{V}$, and a Refinement model R producing g, v, and r respectively. This can be mapped onto the meta-CoT setting, where

$$p(a \mid q) = \int p_G(g \mid q)\, p_{\mathcal{V}}(v \mid q, g)\, p_R(a \mid q, g, v)\; d(g, v)$$

Here, g and v, from the standpoint of standard $\langle q, a \rangle$ datasets, act like latent meta-steps guiding more complex solution processes, akin to z_t in meta-CoT. By explicitly modeling these roles during joint post-training (see Section 4), the multi-agent sequential deliberation process allows us to tackle problems beyond a single-pass CoT setting.

Supervised Finetuning (SFT) Given $\mathcal{D}^{\text{SFT}}_{\text{train}}$ containing positive demonstrations (e.g. correct generator outputs or useful verifier critiques), we can carry out SFT:

$$\mathcal{L}_{\text{SFT}}(\pi_\theta) = -\mathbb{E}_{(x,y)\sim\mathcal{D}^{\text{SFT}}_{\text{train}}} \sum_{t=1}^{T} \log \pi_\theta(y_t | y_{<t}, x),$$

which has proved to be a simple and relatively successful technique to improve reasoning performance.

Direct Preference Optimisation (DPO) In SFT, we can exclusively use positive samples in order to improve our test set performance. By using preference optimization, we can also leverage negative examples $y^- \prec y^+$, where $\prec$ means that y^+ is preferred over y^-. While RLHF (Christiano et al., 2017) used to be the preferred preference optimization strategy, it was recently shown that reinforcement learning can be effectively replaced by a modified supervised learning objective (Rafailov et al., 2023, DPO):

$$\mathcal{L}_{\text{DPO}}(\pi_\theta) = \mathbb{E}_{(x,y^+,y^-)\sim\mathcal{D}^{\text{DPO}}_{\text{train}}} \sigma\Bigg(\beta \log \frac{\pi_\theta(y^+ \mid x)}{\pi_{\text{ref}}(y^+ \mid x)} - \beta \log \frac{\pi_\theta(y^- \mid x)}{\pi_{\text{ref}}(y^- \mid x)}\Bigg).$$

We assume π_{ref} is a reference policy, obtained through SFT. $\mathcal{D}^{\text{DPO}}_{\text{train}}$ is a dataset of triplets including both queries and positive/negative outputs and β a scaling parameter. We provide a theoretical link between optimizing the DPO objective and the optimal RL policy in Appendix A.4.

4. Method: Multi-Agent LLM Training

In complex reasoning tasks, a single LLM must handle all aspects of the solution generation process - often leading to limited exploration, the lack of self-correction, and difficulty refining partial steps (Ye et al., 2024). However, the reasoning process can be broken down into a system of decentralized LLMs, where each model has differing objectives and/or partial observability. As described in Section 3, this represents a meta-CoT (Xiang et al., 2025) setting where intermediate outputs in the multi-agent setup can improve the overall reasoning process. Although in fully observable and cooperative cases, systems of LLM agents could technically be simulated by a single centralized LLM, a decomposition into separate heterogeneous LLM agents offers various benefits analogous to those observed in decentralized multi-agent learning (Schroeder de Witt et al., 2020). Decentralization factorizes large joint action spaces, allowing each agent to focus on smaller sub-tasks under its own partial observability, leading to more targeted exploration and clearer credit assignment (Tan, 1993). In this section, we present our methodology for a multi-agent setting consisting of a sequential heterogeneous process where agents can be trained based on joint rewards.

4.1. Multi-Agent Inference Setting

We formulate our multi-agent inference setting as a collaborative reasoning framework designed to solve complex tasks. Let $\mathcal{Q}$ denote a dataset of natural language questions, where each $q \in \mathcal{Q}$ represents a specific task instance. The objective is to generate a prediction $a \in \mathcal{A}$ for a given input q, where $\mathcal{A}$ is the set of all possible answers. During training, there exists a ground truth function $f : \mathcal{Q} \to \mathcal{A}$, where $f(q) = a^{\text{GT}}$ serves as the ideal prediction for evaluating a.

Our framework consists of three specialized LLMs acting as distinct agents, each defined as a function:

1. **Generator** ($G : \mathcal{Q} \times \mathcal{P}_G \to \mathcal{O}_G$): Produces an initial response to the question.
2. **Verifier** ($\mathcal{V} : \mathcal{O}_G \times \mathcal{Q} \times \mathcal{P}_{\mathcal{V}} \to \mathcal{O}_{\mathcal{V}}$): Critiques the generated response for potential errors.
3. **Refinement Model** ($R : \mathcal{O}_G \times \mathcal{O}_{\mathcal{V}} \times \mathcal{Q} \times \mathcal{P}_R \to \mathcal{O}_R$): Integrates feedback to improve the final prediction.

Here, $\mathcal{P}_G, \mathcal{P}_{\mathcal{V}}, \mathcal{P}_R$ denote the set of prompts for each model, and $\mathcal{O}_G, \mathcal{O}_{\mathcal{V}}$, and $\mathcal{O}_R$ are the sets of possible outputs for the generator, verifier, and refiner respectively. $\mathcal{A}$ is the set of possible answers extracted from the refiner's output with

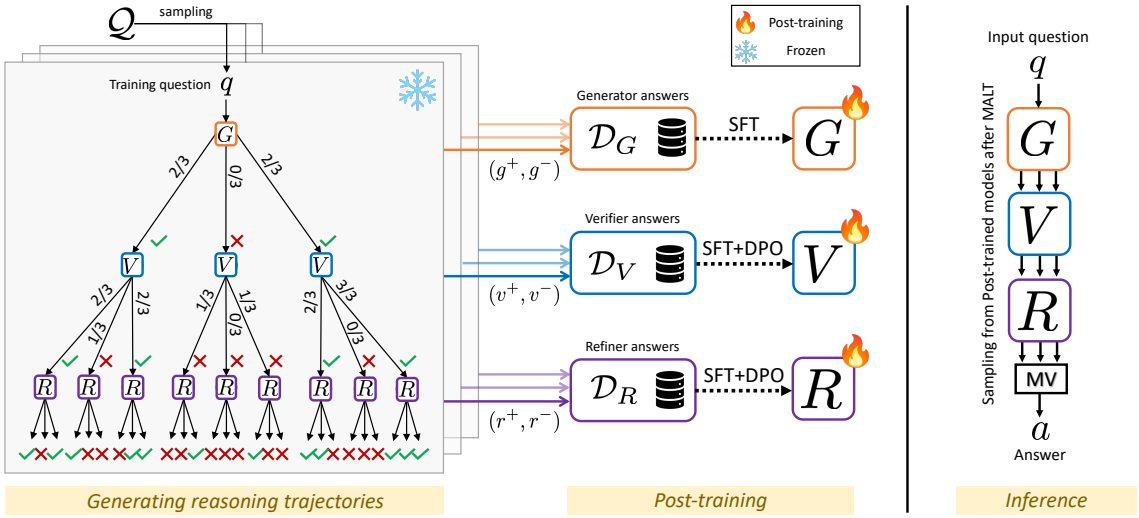

Figure 2. **MALT Method Overview.** Given an input, we consider a three-agent system composed of a Generator for initial answer production, a Verifier providing a critique, and a Refinement Model integrating all intermediate reasoning steps into a final output. For questions in the training set, we introduce a tree search and credit assignment process **(Left)** to generate synthetic datasets with reasoning trajectory preference pairs for each model. These are used to post-train individual models **(Center)**. During inference over the test-set, we perform three parallel sequential passes through the multi-agent setup, and return the final answer obtained via majority voting (**Right**).

a fixed deterministic function T. Formally, we define the interaction between these agents as:

$$g_o = G(q, p_g) \quad \text{where } g_o \in \mathcal{O}_G, p_g \in \mathcal{P}_G \tag{1}$$
$$v_o = \mathcal{V}(q, p_v, g_o) \quad \text{where } v_o \in \mathcal{O}_\mathcal{V}, p_v \in \mathcal{P}_\mathcal{V} \tag{2}$$
$$r_o = R(q, p_r, g_o, v_o) \quad \text{where } r_o \in \mathcal{O}_R, p_r \in \mathcal{P}_R \tag{3}$$
$$a = T(r_o) \quad \text{where } a \in \mathcal{A} \tag{4}$$

This setup is reminiscent of how LLMs are used in production, where they receive initial prompts containing questions, feedback, and are then asked to refine answers (Wang et al., 2024).We demonstrate in Section 5 that this inference setting enhances performance compared to single-model approaches. The key insight, however, relies on leveraging this multi-agent inference setting to generate synthetic data that scales exponentially with respect to a branching factor. Below, we discuss out data generation and post-training setup.

4.2. Collecting Reasoning Trajectories

In standard single-LLM setups, a single model simply produces an answer a for each question q. By contrast, our multi-agent framework uses three specialized agents—Generator, Verifier, and Refiner—sequentially. To enable training of these agents, we need to capture how each agent's output contributes to the final prediction and whether the overall solution is correct or incorrect. A reasoning trace leading to a contains $[g_o, v_o, r_o]$, where the multi-agent answer generation process will be:

$$a = T\Big(R(q,\ p_r,\ G(q,\ p_g),\ \mathcal{V}(q,\ p_v,\ G(q,\ p_g)))\Big). \tag{5}$$

During data generation, $G, \mathcal{V}$, and R all share the same base model parameters; in the subsequent training phase (Section 4.3), each policy is updated independently to specialize in its respective role. An outcome reward model $\mathcal{R} : \mathcal{A} \times \mathcal{A} \to \{0, 1\}$, based on the ground truth in the training set, evaluates a to mark the trajectory as either correct (1) or incorrect (0). Specifically, for a predicted answer a and ground truth a^{GT}, we define:

$$\mathcal{R}(a, a^{\text{GT}}) = \begin{cases} 1, & \text{if } a = a^{\text{GT}}, \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

To collect role-specific post-training data, we use $< q, a^{\text{GT}} >$ pairs from an initial training set $\mathcal{D}_{\text{train}}$ associated with each benchmark. Our solution is illustrated in Figure 2 (left). The following sampling strategy is employed for all models, with a brancing factor of n.

1. For each problem $q_i \in \mathcal{D}_{\text{train}}$ in the training data, we sample n completions from the generator:
$$\{g_{i,j} \sim G(x_i)\}_{j=1}^{n}. \tag{7}$$
2. For each G output $g_{i,j}$, we produce n verifications:
$$\{v_{i,j,k} \sim \mathcal{V}(g_{i,j}, q_i)\}_{k=1}^{n}. \tag{8}$$
3. For each V output $v_{i,j,k}$, we generate n refinements:
$$\{r_{i,j,k,l} \sim R(g_{i,j}, v_{i,j,k}, q_i)\}_{l=1}^{n}. \tag{9}$$

This process results in n^3 trajectories for each training example, totaling $|\mathcal{D}_{\text{train}}| \cdot n^3$ trajectories. We use the outcome

reward model $\mathcal{R}$ to label the refinement outputs as correct (✓) or incorrect (×). The exponential branching factor is very useful to collect a large number of diverse training samples (and inference can be parallelized for efficiency while post-training will just rely on the fixed dataset collected).

To effectively utilize reward signals from the refinement model's outputs, we adopt a **value iteration** approach to propagate values backward through the reasoning chain. Specifically, we compute the *expected value* of each intermediate output (i.e. generator and verifier outputs) based on the values of downstream outputs from the refiner. This global pooling approach is useful because an intermediate step's utility depends on how it influences downstream outputs and final correctness (see Figure 2, Left). Partial solutions can still yield correct answers once refined, so simply providing binary labels is insufficient. By assigning each output an expected value, we better capture the influence on the final reward. This also allows MALT to be generalized to settings where intermediate steps may not carry the same labels, and each intermediate output can now be credited in proportion to its impact on the final solution.

Value Function Definitions We define the value of each *refinement* node $r_{i,j,k,l}$ by directly comparing its final answer a to the ground truth:

$$\mathbf{V}(r_{i,j,k,l}) = \mathcal{R}(T(r_{i,j,k,l}), a^{\text{GT}}) \in \{0, 1\}. \quad (10)$$

The value of a verifier output $v_{i,j,k}$ is then computed as the expected value of its child refinements:

$$\mathbf{V}(v_{i,j,k}) = \mathbb{E}_l[\mathbf{V}(r_{i,j,k,l})] \approx \frac{1}{n}\sum_{l=1}^{n} \mathbf{V}(r_{i,j,k,l}). \quad (11)$$

Similarly, the value of a generator output $g_{i,j}$ is computed as the expected value of its child verifier outputs:

$$\mathbf{V}(g_{i,j}) = \mathbb{E}_k[\mathbf{V}(v_{i,j,k})] \approx \frac{1}{n}\sum_{k=1}^{n} \mathbf{V}(v_{i,j,k}). \quad (12)$$

Each output state's empirical mean is a *Monte Carlo* approximation (unbiased sample avg.) of its true expected correctness, indicated using "≈" for the expected value calculation and for each q. This process allows us to propagate reward signals from final outputs through the tree to each intermediate state, capturing the overall utility of each output.

Thresholding and Binarization To prepare the data for training (SFT and DPO), we *binarize* values using a threshold of 0.5, aligning with majority voting principles. Nodes with values greater than 0.5 are labeled as correct (✓), and those with values less than or equal to 0.5 are labeled as incorrect (×). The intuition behind this strategy is discussed in more detail in Appendix A.4, where we demonstrate that it ensures the policy's expected value monotonically increases. Formally, for each output state s (i.e. output from any model in the multi-agent setup), we define its label $\hat{s}$ as:

$$\hat{s} = \begin{cases} ✓, & \text{if } \mathbf{V}(s) > 0.5, \\ \times, & \text{otherwise.} \end{cases} \quad (13)$$

4.3. MALT Post-training

With each output assigned a value, the goal is to now use these for post-training the models (see Figure 2, Center). We first detail how training data for each model is generated: Each refinement output $r_{i,j,k,l}$ has an associated value $\mathbf{V}(r_{i,j,k,l}) \in \{0, 1\}$. We create preference pairs (r^+, r^-) where r^+ is a correct refinement ($\mathbf{V}(r^+) = 1$) and r^- is an incorrect refinement ($\mathbf{V}(r^-) = 0$) for the same verifier input $v_{i,j,k}$. Formally:

$$\mathcal{D}_R = \left\{ (r^+, r^-) \middle| \begin{matrix} r^+, r^- \in \{r_{i,j,k,l}\}_{l=1}^{n}, \\ \mathbf{V}(r^+) = 1,\ \mathbf{V}(r^-) = 0 \end{matrix} \right\}. \quad (14)$$

For each verifier output $v_{i,j,k}$, we compute its value $\mathbf{V}(v_{i,j,k})$ and binarize it as follows:

$$\hat{v}_{i,j,k} = \begin{cases} ✓, & \text{if } \mathbf{V}(v_{i,j,k}) > 0.5, \\ \times, & \text{otherwise.} \end{cases} \quad (15)$$

Preference pairs for the verifier model are created by comparing outputs under the same generator output $g_{i,j}$:

$$\mathcal{D}_\mathcal{V} = \left\{ (v^+, v^-) \middle| \begin{matrix} v^+, v^- \in \{v_{i,j,k}\}_{k=1}^{n}, \\ \hat{v}^+ = ✓,\ \hat{v}^- = \times \end{matrix} \right\}. \quad (16)$$

A similar process applies to the generator model, where generator outputs $g_{i,j}$ are binarized based on their values $\mathbf{V}(g_{i,j})$, and preference pairs $\mathcal{D}_G$ are created by comparing outputs under the same query q_i. Thus, datasets are collected only in pairs, leading to a sufficiently large (see Sec. 5) and diverse set obtained from the tree expansion process.

Refinement and Verifier Model Training Following the data generation process outlined above and the training setups in Section 3, we first perform SFT on the G, V, R models with the questions and positive samples in $\mathcal{D}_G$, $\mathcal{D}_\mathcal{V}$, and $\mathcal{D}_R$ respectively, updating each model's policy separately. For G, this is analogous to standard STaR post-training (Zelikman et al., 2022) with a specialized dataset. Next, on only the SFT updated V and R models, we apply DPO with question and chosen-rejected pairs in $\mathcal{D}_\mathcal{V}$ and $\mathcal{D}_R$ respectively. For the Generator, we opt for SFT-only because DPO does not improve performance (primarily because the base LLM - Llama 3.1 8B is already extensively post-trained on preferences as a generator (Grattafiori et al., 2024) and Sec. 5.4). By combining SFT with preference-based updates, we capture both "ideal" behaviors (through correct samples) and

"relative" preferences (through correct-vs-incorrect pairs). This allows us to not only bootstrap reasoning based on positive traces but also learn generalizable knowledge about useful reasoning trajectories (Chu et al., 2025). We show our training and inference setups in Figure 2 and a present a detailed algorithm for MALT in Appendix A.1.

5. Experiments

In this section, we outline the experimental details, including a description of the model, the benchmarks used for evaluation, and the training pipeline. We then present our main experimental results, followed by an empirical analysis and baseline comparisons, along with ablations.

5.1. Experimental Details

Benchmarks and Models We use the Llama-3.1-8B-Instruct model (Grattafiori et al., 2024), chosen for its open-source nature and balance between competitive baseline performance and size fitting in a limited compute budget. We evaluate MALT and all baselines on three widely-used benchmarks: GSM8K (Cobbe et al., 2021), which consists of 7.47k training examples and 1.32k test questions, focused on diverse grade school math problems. To evaluate MALT on more challenging mathematical reasoning questions, we use MATH (Hendrycks et al., 2021), consisting of 7.5k training and 5k test problems. MATH has proven to be consistently difficult for smaller language models, with Llama 3.1 8B performing around 49.50% test-accuracy. Lastly, to extend the scope beyond mathematical tasks, we evaluate on CommonsenseQA (CSQA) (Talmor et al., 2019) with 9.74k training examples and 1.22k dev-set questions. CSQA is a multiple-choice question answering dataset around commonsense reasoning problems and has been used similarly by prior work (Zelikman et al., 2022; 2024; Wei et al., 2023).

Baselines We compare MALT against eight baselines in 2 primary settings, all using Llama 3.1 8B. First, we implement the *inference-only* setting with **(1)** a single-agent (SA) naive setting in which a single model is used as a generator to provide an outupt, **(2)** a multi-agent (MA) setting, where the pre-trained baseline model operates in a sequential way as a generator, verifier, and refinement agent with the same prompts given to MALT post-trained models. Our second setting is with only supervised fine-tuning on all three models with the positive data from the synthetic set collected for MALT. We refer to this as our *STaR baseline*. MALT involves a majority-voting based self-consistency mechanism described below and also implemented for all baselines.

MALT Procedure For MALT, we generate synthetic data for each benchmark separately using the tree-based approach (Algorithm 1) with a branching factor of $n = 3$, yielding 27 trajectories per question and approximately 2k–6k labeled pairs per model/benchmark. Each final answer is compared against the ground truth in the training set to assign a binary reward, which is then propagated to label the G, V, and R outputs. During this, each model has a fixed role conditioning prompt template that is also used for baselines and for the post-trained models. We first train each role-specific Llama-3.1-8B-Instruct model on positive labels with LoRA-based SFT and then with DPO to incorporate reasoning preference data for reinforcement learning (discussed in Section 4.3. We use LoRA adapter-based fine-tuning(Hu et al., 2021), significantly reducing the computational load for post-training (see Appendix A.6 for more details). At inference, MALT follows a simple sequential inference pass through the three heterogeneous models reasoning over questions in the test set. Training small models on long sequences of text can lead to instability or hallucinations (Park et al., 2024), therefore MALT employs a three-vote self-consistency mechanism, mitigating potential hallucinations.

5.2. Experimental Results

Our experimental results along with all the baselines are presented in Table 1, along with Ablations in Tables 2 and 3. Our results are averaged over four runs on random subsets of the large test-sets across seeds, and we report the standard deviation for all our results. Below, we detail our experimental results and associated analysis and discussion.

Baselines comparison We propose baseline results in Table 1. Baseline single agent scores on MATH, CSQA, and GSM8K are 49.50%, 74.50%, and 84.25%, approximately in line with scores reported in Llama 3.1 8B's release (Grattafiori et al., 2024). These scores go up to 52.50%, 75.75%, 86.75% and 53.50%, 79.00%, 87.00% with single model majority voting and multi-agent majority voting respectively. Additionally, after applying STaR (SFT on positive synthetic CoTs), single-agent baselines achieve around 52.25%, 76.25%, and 81.75% on MATH, CSQA, and GSM8K, while the multi-agent STaR variant remains around 52.50%, 75.50%, and 80.00%. Self consistency results for STaR variants surpass purely untrained baselines but still lag behind the reasoning improvements obtained with MALT and described below.

MALT Core Results MALT (Table 1, right) achieves an accuracy of 57.25%, 81.50%, and 90.50% on MATH, CSQA, and GSM8K. Overall, *MALT significantly outperforms all baselines, including all settings with supervised fine-tuned models. Over the base model's performance as a generator, MALT achieves relative improvements of* 15.66%, 9.40%, *and* 7.42% *on MATH, CSQA, and GSM8K.* This demonstrates the reasoning efficacy of our search and attribution based data generation, post-training, and inference pipeline in MALT across benchmarks of varying difficulty.

Benchmark	Inference-only				STaR Training				MALT	MALT
	SA	*SA+MV*	*MA*	*MA+MV*	*SA*	*SA+MV*	*MA*	*MA+MV*	*(w/o MV)*	
GSM8K	84.25 ± 2.28	86.75 ± 2.38	84.75 ± 2.86	87.00 ± 4.00	81.75 ± 0.83	84.75 ± 2.68	80.00 ± 1.58	86.75 ± 2.28	83.50 ± 2.18	**90.50** ± 2.06
CSQA	74.50 ± 3.35	75.75 ± 5.49	77.50 ± 5.17	79.00 ± 4.69	76.25 ± 4.32	78.75 ± 4.26	75.50 ± 2.69	76.00 ± 1.73	77.50 ± 1.12	**81.50** ± 2.29
MATH	49.50 ± 2.06	52.50 ± 2.50	51.75 ± 3.56	53.50 ± 2.87	52.25 ± 1.48	54.00 ± 2.73	52.50 ± 3.20	53.75 ± 2.68	52.25 ± 1.79	**57.25** ± 1.48

Test Accuracy (%) over 4 seeds ↑

Table 1. **Benchmark results.** We compare MALT with baselines on three different benchmarks. For baselines, we include different setups such as single agent (SA) and multi-agent (MA), both with and without majority voting (MV). *MALT outperforms all baselines.*

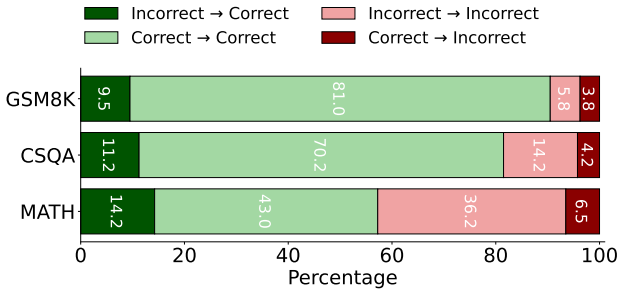

Figure 3. **Self-correction rates.** MALT consistently increases the number of correct answers by correcting previously incorrect answers at a much higher rate than introducing new mistakes compared to a single-model baseline (MV@3).

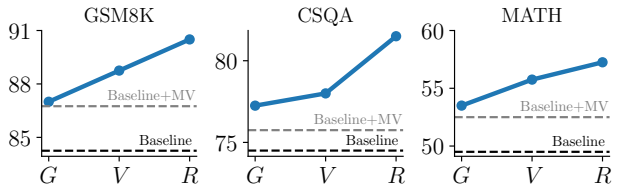

Figure 4. **Improvement over turns.** MALT demonstrates consistent improvements at each turn. Test accuracy at each turn from a sequential pass through the post-trained Generator, Verifier, and Refinement models (All with MV@3).

5.3. Core analysis

Self-consistency From results in Table 1, MALT demonstrates improved reasoning with majority-voting for self-consistency (Wang et al., 2022), but its performance without MV remained close to that of the multi-agent inference-only setting. A qualitative examination of the reasoning trajectories revealed that while MALT was able to solve questions previously unsolved by all baselines, it experienced self-consistency issues due to hallucinations. Thus, we use a small majority voting factor of 3 and observe that self-consistency reliably yields significantly higher relative improvement for MALT over baselines. For instance, on MATH, self-consistency resulted in a relative improvement of 9.57%, exceeding the $3-6\%$ gains in other baselines.

Reasoning self-correction rates We measured how often MALT flips an incorrect solution into a correct one versus the opposite (Figure 3). On MATH, MALT converts 14.25% of wrong answers into correct ones, while only 6.50% of correct answers become wrong. For GSM8K, 9.75% flip to correct and 3.75% flip to incorrect, and CSQA with 11.25% vs. 4.25%. These numbers indicate that MALT's generator–verifier–refiner process shows strong self-improvement with significantly lower mistakes introduced.

Improvement over turns We use GPT-4 to rate the accuracy of each step MALT produced on the test sets to analyze its performance after each agent inference ("turn"). Figure 4 shows how performance evolves. For example, MATH's untrained baseline improves from 49.50% to 52.50% with majority voting. MALT, over its 3 reasoning and multi-agent deliberation turns, increases performance to 53.50% (turn 1 - Generator), 55.75% (turn 2 - Verifier), and 57.25% (turn 3 - Refiner). We observe similar increases across GSM8K and CSQA. This consistent turn-wise improvement highlights how sequential verification and refinement are both very important for improving overall reasoning performance. We also provide ablations to understand why generate-verify-refine is the most appropriate paradigm in Section 5.4.

Overall performance Across all benchmarks, MALT consistently outperforms both zero-shot and fine-tuned baselines, closing gaps on problems previously unsolved by any baseline. Its multi-agent framework not only achieves higher average scores but also corrects systematic errors. For example, the verifier often successfully locates errors by redoing calculations and providing a precise critique, with the pattern learnt automatically from synthetic data generated via our search process. Similarly, for CSQA, the verifier implicitly learns to focus on aspects of the problem overlooked by the generator, allowing for additional inference-compute spent on important gaps. Examples of these reasoning interactions are provided in Appendix A.2.

MALT Post-training G	V	R	GSM8K	CSQA	MATH
✗	✓	✓	87.00	78.75	54.25
✓	✗	✓	85.75	76.75	54.50
✓	✓	✗	86.25	75.50	55.25
✓	✓	✓	**90.50**	**81.50**	**57.25**

Table 2. **Ablations with untrained models in MALT (MV@3).** This table shows the performance of untrained agents combined with trained ones, demonstrating that all agents perform best when cooperating with peers trained with MALT.

Configuration	GSM8K	CSQA	MATH
G only	84.75	78.75	54.00
$G + V$	88.75	78.00	55.75
$G + R$	84.75	76.25	54.75
$G + V + R$ (Ours)	**90.50**	**81.50**	**57.25**

Table 3. **Performance of ablated multi-agent setups (MV@3).** This table shows that all agents in our pipeline are necessary to achieve the best results.

5.4. Ablation studies

Untrained model ablations We test how useful training each individual agent is by replacing one of them with an untrained baseline and keeping the other two MALT-trained agents. As seen in Table 2, this consistently degrades performance on all benchmarks. For instance, using an untrained generator yields mean accuracies of 54.25% (MATH), 87.00% (GSM8K), and 78.75% (CSQA), notably lower than a fully trained system. Similar drops occur when substituting an untrained verifier or refiner. This reinforces that *all* agents benefit from the MALT pipeline, and removing training from any one of them reduces performance (see Table 2).

Importance of generate-verify-refine Next, we measure the impact of removing one role from our three-agent pipeline and running simpler two-agent systems. Specifically, we compare (i) *generate-refine* ($G + R$, skipping the verifier) and (ii) *generate-verify* ($G + V$, skipping the refiner). As shown in Table 3, both setups underperform the full pipeline: $G + R$ yields mean accuracies of 54.75% (MATH), 84.75% (GSM8K), and 76.25% (CSQA), while $G + V$ yields 55.75% (MATH), 88.75% (GSM8K), and 78.00% (CSQA), both paradigms below the generate-verify-refine setup in MALT. This confirms our specific multi-agent setup yields consistently stronger results by enabling more inference spent on sequential deliberation.

Effectiveness of DPO over only SFT As shown in Table 1, DPO consistently improves MALT's performance beyond SFT alone by leveraging negative data—an approach also observed in (Putta et al., 2024) and (Setlur et al., 2024). In particular, purely positive "rationales" can introduce spurious correlations and degrade SFT performance, which does indeed occur in our empirical and qualitative results; the contrastive training approach that DPO provides with negative data helps the model identify high-advantage reasoning steps to improve with higher sample-efficiency (Rafailov et al., 2024).For reasoning problems, SFT tends to memorize the data and rules, which is useful to bootstrap reasoning to a certain extent (Zelikman et al., 2022). However, our results indicate that this could degrade performance sometimes, and preference learning methods such as DPO (see Appendix A.4.2 for a theoretical analysis) exhibit better performance at learning generalizable knowledge around reasoning steps (Chu et al., 2025). However, for the Generator, we find that SFT+DPO actually *lowers* performance (for e.g. 52.25% on MATH with SFT and 51.25% with SFT+DPO)—likely because Llama 3.1 8B-Instruct already underwent post-training with DPO on a very similar generator data distribution for benchmarks (Grattafiori et al., 2024), making DPO on a similar distribution prone to overfitting, consistent with observations in Setlur et al. (2024).

6. Discussion and Conclusion

We presented MALT, a novel post-training strategy dividing CoT reasoning among three specialized LLMs-a generator, verifier, and refiner-to tackle complex reasoning tasks. To our knowledge, MALT is the first approach bridging the gap between prior multi-agent inference methods and fully-trained multi-agent systems, by generating role-specific data using a tree-based sampling and credit assignment mechanism, without human annotations. Crucially, MALT utilizes the negative synthetic data to identify and correct flawed reasoning steps with LLM post-training. Unlike standard single-LLM setups, our design closely mirrors how humans solve complex tasks or even use LLMs—attempting a solution, critiquing errors, and finally refining the result. This process yields consistent gains on benchmarks of varying difficulty across all baselines and ablations, demonstrating the effectiveness of multi-agent post-training. We provide detailed theoretical grounding and additional explanations In Appendix A.4. Moreover, in Section A.5, we outline future directions, such as scaling the branching factor and understanding its effect on performance, and moving to an online RL paradigm. These directions require more compute and remain beyond the scope of this work.

By structuring the reasoning process as a multi-agent system in which specialized models sequentially deliberate over problems, MALT enables role-specific reasoning capabilities in pre-trained models to improve. This allows systems of LLMs to more efficiently utilize meta-strategies such as self-correction and chaining inference steps. With MALT, we introduce a new research paradigm that can be applied to any foundation model to improve reasoning capabilities.

Impact Statement

This paper presents a method to enhance the reasoning capabilities of frontier models in multi-agent setups. We believe that our contributions may have significant societal implications. First, our post-trained agents demonstrated substantial performance improvements across competitive benchmarks. This could enable groups of small, open-source models to achieve performance levels comparable to closed-source frontier models. Additionally, MALT has important implications for AI safety, which we address in Appendix A.7. We emphasize the importance of using our contributions responsibly.

References

Brown, B., Juravsky, J., Ehrlich, R., Clark, R., Le, Q. V., Ré, C., and Mirhoseini, A. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. Deep Reinforcement Learning from Human Preferences. In *NeurIPS*. Curran Associates, Inc., 2017.

Chu, T., Zhai, Y., Yang, J., Tong, S., Xie, S., Schuurmans, D., Le, Q. V., Levine, S., and Ma, Y. Sft memorizes, rl generalizes: A comparative study of foundation model post-training. *arXiv*, 2025.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. *arXiv*, 2021.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019.

Du, Y., Li, S., Torralba, A., Tenenbaum, J. B., and Mordatch, I. Improving factuality and reasoning in language models through multiagent debate. In *ICML*, 2024.

Fourney, A., Bansal, G., Mozannar, H., Tan, C., Salinas, E., Niedtner, F., Proebsting, G., Bassman, G., Gerrits, J., Alber, J., et al. Magentic-one: A generalist multi-agent system for solving complex tasks. *arXiv*, 2024.

Geva, M., Goldberg, Y., and Berant, J. Are we modeling the task or the annotator? an investigation of annotator bias in natural language understanding datasets. In *EMNLP*, 2019.

Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., and Team, L. The llama 3 herd of models. *arXiv*, 2024.

Greenblatt, R., Shlegeris, B., Sachan, K., and Roger, F. AI Control: Improving Safety Despite Intentional Subversion. In *ICML*, 2024.

Han, S., Zhang, Q., Yao, Y., Jin, W., Xu, Z., and He, C. Llm multi-agent systems: Challenges and open problems. *arXiv*, 2024.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the math dataset. In *NeurIPS*, 2021.

Howard, J. and Ruder, S. Universal language model fine-tuning for text classification. In *ACL*, 2018.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. In *ICLR*, 2021.

Ivison, H., Wang, Y., Liu, J., Wu, Z., Pyatkin, V., Lambert, N., Smith, N. A., Choi, Y., and Hajishirzi, H. Unpacking dpo and ppo: Disentangling best practices for learning from preference feedback. In *NeurIPS*, 2024.

Lerer, A., Hu, H., Foerster, J., and Brown, N. Improving policies via search in cooperative partially observable games. In *AAAI*, 2020.

McAleese, N., Pokorny, R. M., Uribe, J. F. C., Nitishinskaya, E., Trebacz, M., and Leike, J. LLM Critics Help Catch LLM Bugs. *arXiv*, 2024.

Pang, R. Y., Yuan, W., Cho, K., He, H., Sukhbaatar, S., and Weston, J. Iterative reasoning preference optimization, 2024.

Park, R., Rafailov, R., Ermon, S., and Finn, C. Disentangling length from quality in direct preference optimization. In *ACL Findings*, 2024.

Putta, P., Mills, E., Garg, N., Motwani, S., Finn, C., Garg, D., and Rafailov, R. Agent q: Advanced reasoning and learning for autonomous ai agents. *arXiv*, 2024.

Qian, C., Liu, W., Liu, H., Chen, N., Dang, Y., Li, J., Yang, C., Chen, W., Su, Y., Cong, X., Xu, J., Li, D., Liu, Z., and Sun, M. ChatDev: Communicative Agents for Software Development. In *ACL*, 2024.

Qu, Y., Zhang, T., Garg, N., and Kumar, A. Recursive introspection: Teaching language model agents how to self-improve. In *NeurIPS*, 2024.

Rafailov, R., Sharma, A., Mitchell, E., Manning, C. D., Ermon, S., and Finn, C. Direct preference optimization: Your language model is secretly a reward model. In *NeurIPS*, 2023.

Rafailov, R., Hejna, J., Park, R., and Finn, C. From r to q^*: Your language model is secretly a q-function. In *COLM*, 2024.

Saunders, W., Yeh, C., Wu, J., Bills, S., Ouyang, L., Ward, J., and Leike, J. Self-critiquing models for assisting human evaluators. *arXiv*, 2022.

Schroeder de Witt, C., Gupta, T., Makoviichuk, D., Makoviychuk, V., Torr, P. H. S., Sun, M., and Whiteson, S. Is Independent Learning All You Need in the StarCraft Multi-Agent Challenge? *arXiv*, 2020.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. Trust region policy optimization. In *ICML*, 2017a.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv*, 2017b.

Setlur, A., Garg, S., Geng, X., Garg, N., Smith, V., and Kumar, A. Rl on incorrect synthetic data scales the efficiency of llm math reasoning by eight-fold. In *NeurIPS*, 2024.

Singh, A., Co-Reyes, J. D., Agarwal, R., Anand, A., Patil, P., Garcia, X., Liu, P. J., Harrison, J., Lee, J., Xu, K., Parisi, A., Kumar, A., Alemi, A., Rizkowsky, A., Nova, A., Adlam, B., Bohnet, B., Elsayed, G., Sedghi, H., Mordatch, I., Simpson, I., Gur, I., Snoek, J., Pennington, J., Hron, J., Kenealy, K., Swersky, K., Mahajan, K., Culp, L., Xiao, L., Bileschi, M. L., Constant, N., Novak, R., Liu, R., Warkentin, T., Qian, Y., Bansal, Y., Dyer, E., Neyshabur, B., Sohl-Dickstein, J., and Fiedel, N. Beyond human data: Scaling self-training for problem-solving with language models. *TMLR*, 2024.

Snell, C., Lee, J., Xu, K., and Kumar, A. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv*, 2024.

Sutton, R. S. and Barto, A. G. Reinforcement learning: an introduction, 2nd edn. adaptive computation and machine learning, 2018.

Talmor, A., Herzig, J., Lourie, N., and Berant, J. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *NAACL*, 2019.

Tan, M. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pp. 330–337, 1993.

Tumer, K. and Agogino, A. Distributed agent-based air traffic flow management. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '07, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9788190426275. doi: 10.1145/1329125.1329434. URL https://doi.org/10.1145/1329125.1329434.

Wang, J., Ma, W., Sun, P., Zhang, M., and Nie, J.-Y. Understanding user experience in large language model interactions. *arXiv*, 2024.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models, 2022.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2023.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Zhang, S., Zhu, E., Li, B., Jiang, L., Zhang, X., and Wang, C. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. In *COLM*, 2024.

Xiang, V., Snell, C., Gandhi, K., Albalak, A., Singh, A., Blagden, C., Phung, D., Rafailov, R., Lile, N., Mahan, D., Castricato, L., Franken, J.-P., Haber, N., and Finn, C. Towards system 2 reasoning in llms: Learning how to think with meta chain-of-though. *arXiv*, 2025.

Yang, G., Li, C., Yao, Y., Wang, G., and Teng, Y. Quasi-supervised learning for super-resolution pet. *Computerized Medical Imaging and Graphics*, 2024.

Ye, T., Xu, Z., Li, Y., and Allen-Zhu, Z. Physics of language models: Part 2.2, how to learn from mistakes on grade-school math problems. *arXiv*, 2024.

Zelikman, E., Wu, Y., Mu, J., and Goodman, N. D. Star: Bootstrapping reasoning with reasoning. In *NeurIPS*, 2022.

Zelikman, E., Harik, G., Shao, Y., Jayasiri, V., Haber, N., and Goodman, N. D. Quiet-star: Language models can teach themselves to think before speaking. In *COLM*, 2024.

A. Appendix

A.1. Algorithm

MALT orchestrates a three-agent system—comprising a Generator for initial answers, a Verifier for critiques, and a Refiner that integrates these steps into a final response. During training, we expand a multi-agent search tree for each question, labeling correct and incorrect branches via a value iteration based credit assignment mechanism. This generates role-specific preference pairs for post-training each agent via supervised fine-tuning and preference optimization. Algorithm 1 provides a complete description of the data collection and training pipeline for MALT.

Algorithm 1 Multi-Agent LLM Training and Synthetic Data Generation (MALT)

Require: Initial Dataset $\mathcal{D}$, Models G, V, R, Branching factor n
GOAL: Trained models G', V', R'
Initialize datasets $\mathcal{S}_G$, $\mathcal{S}_V$, $\mathcal{S}_R$ as empty sets
for $q \in \mathcal{D}$ **do**
  $A_G \leftarrow \{g_j = G(q)\}_{j=1}^{n}$ ▷ Generate n outputs from G
  for each $g_j \in A_G$ **do**
    $A_V^{g_j} \leftarrow \{v_{j,k} = V(q, g_j)\}_{k=1}^{n}$ ▷ Generate n outputs from V
    for each $v_{j,k} \in A_V^{g_j}$ **do**
      $A_R^{g_j, v_{j,k}} \leftarrow \{r_{j,k,l} = R(q, g_j, v_{j,k})\}_{l=1}^{n}$ ▷ Generate n outputs from R
      for each $r_{j,k,l} \in A_R^{g_j, v_{j,k}}$ **do**
        Compute $V(r_{j,k,l}) = \mathcal{R}(r_{j,k,l}, a^{\text{GT}})$ ▷ Reward for R output
        Add $(q, g_j, v_{j,k}, r_{j,k,l}, V(r_{j,k,l}))$ to $\mathcal{S}_R$
      end for
      Compute $V(v_{j,k}) = \frac{1}{n} \sum_{l=1}^{n} V(r_{j,k,l})$ ▷ Value for V output
      Binarize $\hat{v}_{j,k} = \mathbb{I}[V(v_{j,k}) > 0.5]$
      Add $(q, g_j, v_{j,k}, \hat{v}_{j,k})$ to $\mathcal{S}_V$
    end for
    Compute $V(g_j) = \frac{1}{n} \sum_{k=1}^{n} V(v_{j,k})$ ▷ Value for G output
    Binarize $\hat{g}_j = \mathbb{I}[V(g_j) > 0.5]$
    Add $(q, g_j, \hat{g}_j)$ to $\mathcal{S}_G$
  end for
end for
Training the Models
$G' \leftarrow \text{SFT}(G, \mathcal{S}_G)$ ▷ Fine-tune G with supervised data
$V_{\text{SFT}} \leftarrow \text{SFT}\left(V, \{(q, g_j, v_{j,k}) \mid (q, g_j, v_{j,k}, \hat{v}_{j,k}) \in \mathcal{S}_V, \hat{v}_{j,k} = 1\}\right)$ ▷ Fine-tune V on positive samples
$V' \leftarrow \text{DPO}(V_{\text{SFT}}, \mathcal{S}_V)$ ▷ Train V with DPO using preferences
$R_{\text{SFT}} \leftarrow \text{SFT}\left(R, \{(q, g_j, v_{j,k}, r_{j,k,l}) \mid V(r_{j,k,l}) = 1\}\right)$ ▷ Fine-tune R on positive samples
$R' \leftarrow \text{DPO}(R_{\text{SFT}}, \mathcal{S}_R)$ ▷ Train R with DPO using preferences
return G', V', R'

A.2. Qualitative Examples

In this section, we present three key qualitative examples: a synthetic training example for SFT and DPO, a qualitative performance comparison between an untrained and a trained verifier, and a complete MALT setup demonstrating a backtracking and improvement reasoning cycle that generates a correct answer.

GSM8K Synthetic Dataset Sample for the Refinement Model (GT Answer = 25)

System prompt
You are an expert problem solver that refines math solutions based on feedback.

Input

Question: Marco loves trading cards. He has a total of 500 cards on hand, and a fourth of them are duplicates. Marco decides to trade one-fifth of these duplicates for completely new cards to his friend Josh. How many new cards did Marco get?

Initial Answer: I'd be happy to help you solve this problem step by step.

Step 1: Identify the key information given in the question

* Marco has a total of 500 cards. * A fourth of them are duplicates.

Step 2: Determine what the question is asking us to find

* The question is asking us to find how many new cards Marco got from trading one-fifth of his duplicates with Josh.

Step 3: Plan our approach to solving the problem

* First, we need to find the number of duplicates Marco has. Since a fourth of his cards are duplicates, we can find the number of duplicates by multiplying 500 by 1/4. * Then, we need to find one-fifth of the number of duplicates to find out how many cards Marco traded with Josh. * Finally, we'll subtract the number of cards traded from the total number of duplicates to find the remaining duplicates.

Step 4: Execute our plan

* Number of duplicates = 500 x 1/4 = 125 * One-fifth of the number of duplicates = 125 x 1/5 = 25 * Marco traded 25 cards with Josh.

Step 5: State our conclusion and double-check our answer

* Marco got 25 new cards from trading with Josh.

Final Answer: 25

Rejected Output

Based on the initial answer and the verification feedback, I refine the solution as follows:

Step 1: Identify the key information given in the question

* Marco has a total of 500 cards. * A fourth of them are duplicates.

Step 2: Determine what the question is asking us to find

* The question is asking us to find how many new cards Marco got from trading one-fifth of his duplicates with Josh.

Step 3: Plan our approach to solving the problem

* First, we need to find the number of duplicates Marco has. Since a fourth of his cards are duplicates, we can find the number of duplicates by multiplying 500 by 1/4. * Then, we need to find one-fifth of the number of duplicates to find out how many cards Marco traded with Josh. * Finally, we'll subtract the number of cards traded from the total number of duplicates to find the remaining duplicates.

Step 4: Execute our plan

* Number of duplicates = 500 x 1/4 = 125 * One-fifth of the number of duplicates = 125 x 1/5 = 25 * Marco traded 25 cards with Josh. Since he traded one-fifth of his duplicates, we can find the total number of duplicates he has left by adding the number of cards traded back to the number of duplicates: 125 + 25 = 150 * To find the number of new cards Marco got, we subtract the number of duplicates he traded from the total number of duplicates: 150 - 25 = 125

Step 5: State our conclusion and double-check our answer

* Marco got 125 new cards from trading with Josh.

Final Answer: Final Answer: 125

Accepted Output

Refined Solution:

Step 1: Identify the key information given in the question

* Marco has a total of 500 cards. * A fourth of them are duplicates. * Marco trades one-fifth of his duplicates with Josh.

Step 2: Determine what the question is asking us to find

* The question is asking us to find how many new cards Marco got from trading one-fifth of his duplicates with Josh.

Step 3: Plan our approach to solving the problem

* First, we need to find the number of duplicates Marco has. Since a fourth of his cards are duplicates, we can find the number of duplicates by multiplying 500 by 1/4. * Then, we need to find one-fifth of the number of duplicates to find out how many cards Marco traded with Josh. * Finally, we'll subtract the number of cards traded from the total number of duplicates to find the remaining duplicates.
Step 4: Execute our plan
* Number of duplicates = 500 x 1/4 = 125 * One-fifth of the number of duplicates = 125 x 1/5 = 25 * Marco traded 25 cards with Josh. * Remaining duplicates = 125 - 25 = 100
Step 5: State our conclusion and double-check our answer
* Marco got 25 new cards from trading with Josh.
Final Answer: 25

Qualitative Example of an Untrained VS Trained Verifier

Question

Billy buys a 12-pack of soda from the store. If he has twice as many brothers as sisters, and he has 2 sisters, how many sodas can he give to each of his siblings if he wants to give out the entire 12-pack while giving each the same number?

Initial Answer

I'd be happy to help you solve this problem step by step.
Step 1: Identify the key information given in the question.
* Billy buys a 12-pack of soda. * He has twice as many brothers as sisters. * He has 2 sisters.
Step 2: Determine what the question is asking us to find.
The question is asking us to find out how many sodas Billy can give to each of his siblings if he wants to give out the entire 12-pack while giving each the same number.
Step 3: Plan our approach to solving the problem.
Let's let x be the number of siblings (brothers and sisters). We know that Billy has 2 sisters, so the number of brothers is 2 times the number of sisters, which is 2x. The total number of siblings is the sum of the number of sisters and brothers, which is x + 2x = 3x. We are given that Billy has a 12-pack of soda, so we can set up the equation:
12 = x + 2x
Step 4: Execute our plan, showing all work clearly.
To solve for x, we can combine like terms:
12 = 3x
Divide both sides by 3:
4 = x
So, Billy has 4 siblings in total (2 sisters and 2 brothers).
Step 5: State our conclusion and double-check our answer.
Since Billy has 12 sodas and 4 siblings, he can give:
12 ÷ 4 = 3
sodas to each of his siblings.
Final Answer: 3

Untrained Verifier

As a verifier, I will critically evaluate the initial answer provided above.
The initial answer states that Billy can give 3 sodas to each of his siblings. Let's verify the answer by performing the calculations.
Verification:
1. We are given that Billy has 12 sodas and 4 siblings (2 sisters and 2 brothers). 2. To find out how many sodas each sibling can get, we can divide the total number of sodas (12) by the number of siblings (4):
12 ÷ 4 = 3

The initial answer is correct! Billy can indeed give 3 sodas to each of his siblings.
Feedback:
The initial answer is mathematically correct. The calculations are accurate, and the solution is well-explained. The answer is also logically consistent with the given information. Well done!
However, it's always a good idea to double-check the calculations to ensure accuracy. In this case, the verification step confirms that the initial answer is correct.

Trained Verifier
As a verifier, I will carefully evaluate the initial answer provided above.
The initial answer states that Billy can give 3 sodas to each of his siblings. Let's double-check the calculations to ensure the answer is mathematically correct.
The initial answer claims that Billy has 4 siblings (2 sisters and 2 brothers). However, the question states that Billy has twice as many brothers as sisters, and he has 2 sisters. This implies that the number of brothers is 2 times the number of sisters, which is 2 x 2 = 4 brothers. Therefore, Billy actually has 2 sisters and 4 brothers, making a total of 6 siblings.
The initial answer also claims that Billy can give 12 sodas to 4 siblings, resulting in 3 sodas per sibling. However, dividing 12 sodas among 6 siblings would actually give 2 sodas per sibling, not 3.
Conclusion:
The initial answer contains a calculation error. The correct number of siblings is 6, not 4. Therefore, Billy can give 2 sodas to each of his siblings, not 3.
As a verifier, I recommend that the initial answer be corrected to reflect the correct calculation. The correct answer is 2 sodas per sibling.

A Complete Reasoning and Improvement Sequence Example with MALT on CommonsenseQA

Question
He looked at the field of pumps, all slowly churning oil out of the what?

Choices
A: manual
B: street
C: restaurant
D: ground
E: service station

Initial Answer (Generator)
Reasoning steps: 1. The context of the question is about pumps churning oil, which suggests a location where oil is being dispensed or extracted. 2. A "manual" (A) does not typically relate to a location where oil is churning. 3. A "street" (B) is a general location, but it's not specific to oil churning. 4. A "restaurant" (C) is not associated with oil churning. 5. The "ground" (D) could be a possible location, but it's not the most specific or common location for oil churning. 6. A "service station" (E) is a common location where oil is churning, as it's where cars are serviced and oil is dispensed.
Final Answer: E

Verifier
The initial answer is E: service station.
This answer seems to be the most logical choice given the context of the question. The phrase "field of pumps" suggests a location where multiple pumps are present, and a service station is a common place where multiple

pumps are used to dispense oil and other fuels.
However, the question asks what the pumps are churning oil out of, not where the pumps are located. This changes the interpretation of the question.
Considering the phrase "churning oil out of," it implies that the pumps are extracting or pumping oil from a source. In this case, the most likely source of oil would be the ground, as oil is often extracted from the ground.
Therefore, the correct answer is D: ground.

Refined Answer
Reasoning steps: 1. The context of the question is about pumps churning oil, which suggests a location where oil is being extracted or dispensed. 2. The phrase "churning oil out of" implies that the pumps are extracting or pumping oil from a source. 3. The most common source of oil is the ground, as it is often extracted from the earth. 4. Therefore, the pumps are churning oil out of the ground.
Final Answer: D

Ground Truth (not provided to any model)
D

A.3. Detailed Discussion

We now discuss key design choices. Simplicity with theoretical grounding—particularly the assurance of a trained policy with a monotonically increasing value—was a primary consideration. We opted for DPO over PPO-based methods due to our focus on a straightforward offline data generation process, treating our approach as an independent learning problem in a centralized setting (Lerer et al., 2020) with a single iteration (the key difference being that our agent policies post-training differ). In this setting, DPO is more stable than PPO and requires less overhead. While PPO could use the value computed at each branch as a reward for post-training nodes (a promising future direction), it introduces significant computational complexity. Moving from offline to online RL with additional computational overhead might indeed improve performance quite a bit.

Our value iteration method, when binarized, resembles global majority-based pooling: for a given node and branch, the binary reward of the leaf nodes in the subtree determines the usefulness of the branch, analogous to binarizing values propagated through the tree. In contrast, local pooling computes the binary value of a branch based only on the majority outcomes of its direct children, propagating this process to the leaf nodes. We also leave the choice between MCTS and an expansive tree-based sampling strategy as an open problem. Given our limited tree depth, tree-based sampling proved efficient, supported synthetic data generation with an exponential branching factor, and produces explainable outputs. Our dataset is collected offline, and individual models are trained on this synthetic data. While this approach works empirically, handling any new, out-of-distribution data would require iterative rollout and post-training methods.

Based on our empirical results and the modularity of our algorithmic approach, it is highly plausible that our method will scale to larger models and scenarios with many agents, thus laying the foundations for new state-of-the-art AI agents based on systems of cooperative frontier models. Overall, our multi-agent system is currently composed of a sequence of agents that start out with the same parameters and different prompts. MALT performs joint training to transform this into a heterogeneous agent setting, where agents with different parameters operate cooperatively. Exploring other multi-agent settings is an important direction for subsequent work.

A.4. Theoretical Justification for MALT

A.4.1. Credit Assignment Strategy

Here, we provide a theoretical justification for why our framework, when updating the agent policies based on binarized pooled rewards with a threshold at $\theta = 0.5$, leads to policy improvements. We formalize the pooling operation, define the dynamic programming approach of value iteration in our context, and show that our policy updates lead to monotonic increases in expected reward.

We consider a tree-structured reasoning process where each node represents a model in our system: Generator (G), Verifier (V), or Refinement Model (R). The branches connecting these nodes represent the outputs (actions) generated by each model. For each model, multiple outputs (branches) are possible, representing the different actions or decisions the model can make. Each leaf node corresponds to the terminal outputs of the Refinement model R and receives a binary reward:

$$\mathcal{R}(r) = \begin{cases} 1, & \text{if } r \text{ is correct,} \\ 0, & \text{otherwise.} \end{cases} \tag{17}$$

We define the value function $V(\nu)$ for a node ν (model) as the expected reward of its downstream refinements starting from that node:

$$V(\nu) = \mathbb{E}_{\pi}[\mathcal{R}(r) \mid \nu], \tag{18}$$

where the expectation is over all possible branches under the current policy π, starting from node ν. Starting from the leaves:

$$V(r) = \mathcal{R}(r) \in \{0, 1\}, \tag{19}$$

At each node ν (model), the value is computed as:

$$V(\nu) = \frac{1}{K} \sum_{i=1}^{K} Q(\mu, a_i), \tag{20}$$

where:

- K is the number of outputs sampled at node ν,
- μ is the subsequent model with K outputs

This pooling operation aggregates the expected rewards from downstream paths in a dynamic-programming approach. We then apply a binarization function $\hat{V}(\nu)$ to label nodes as 'useful' (1) or 'not useful' (0) using a threshold $\theta = 0.5$:

$$\hat{V}(\nu) = \begin{cases} 1, & \text{if } V(\nu) > 0.5, \\ 0, & \text{otherwise.} \end{cases} \tag{21}$$

There is a monotonic relationship between values and binarization. Note that we binarize our values because of computational constraints and DPO's requirement of a pair of useful and not useful outputs for preference optimization. With the use of PPO, we could potentially use these values directly as rewards for optimal post-training and leave this for future work. For any two nodes ν and μ, if $V(\nu) \geq V(\mu)$, then $\hat{V}(\nu) \geq \hat{V}(\mu)$, due to binarization being a non-decreasing function. We update the policy π_i by using the binarized pooled reward $\hat{V}(s')$ of the next node as the reward signal.

We now show that if we use the binarized pooled reward for policy gradient updates, we will achieve monotonic improvements in the expected reward. Let π_t be the policy at iteration t, and $V_{\pi_t}(\nu)$ be the value of node ν under policy π_t.

Base Case: At iteration $t = 0$, we have the initial value function $V_{\pi_0}(\nu)$ under the initial policy π_0.

Inductive Step: Assume that at iteration t, the value function satisfies:

$$V_{\pi_t}(\nu) \geq V_{\pi_{t-1}}(\nu) \quad \text{for all nodes } \nu. \tag{22}$$

Policy Update: We update the policy π_t to π_{t+1} by using policy gradient methods with the binarized pooled reward $\hat{V}(\nu)$ as the reward signal. Specifically, we adjust the policy to increase the probability of outputs leading to child nodes with $\hat{V}(c_i) = 1$.

Now in order to show that:

$$V_{\pi_{t+1}}(\nu) \geq V_{\pi_t}(\nu) \quad \text{for all nodes } \nu. \tag{23}$$

At each node ν, the value under policy π_t is:

$$V_{\pi_t}(\nu) = \frac{1}{K} \sum_{i=1}^{K} Q_{\pi_t}(\mu, a_i), \tag{24}$$

After the policy update, the value under policy π_{t+1} is:

$$V_{\pi_{t+1}}(\nu) = \frac{1}{K} \sum_{i=1}^{K} Q_{\pi_{t+1}}(\mu, a_i). \tag{25}$$

Using policy gradient methods (Schulman et al., 2017a) that increase the probabilities of selecting outputs leading to child nodes with $\hat{V}(c_i) = 1$, and because $\hat{V}(c_i) = 1$ implies $V(c_i) > 0.5$, we can infer that:

$$V_{\pi_{t+1}}(c_i) \geq V_{\pi_t}(c_i) \quad \text{for all child nodes } c_i. \tag{26}$$

We note that methods such as DPO or PPO are not inherently guaranteed to be monotonically increasing in terms of expected reward and discuss this further below in Appendix A.4.2. TRPO does offer such guarantees (Schulman et al., 2017a). At the very least, if the update is monotonic when considering the actual value, it is guaranteed to be monotonic when using the binarization based on the proof below.

By the inductive hypothesis and the policy update, the values of child nodes do not decrease:

$$V_{\pi_{t+1}}(c_i) \geq V_{\pi_t}(c_i). \tag{27}$$

Therefore, at node ν:

$$V_{\pi_{t+1}}(\nu) = \frac{1}{K} \sum_{i=1}^{K} V_{\pi_{t+1}}(c_i) \geq \frac{1}{K} \sum_{i=1}^{K} V_{\pi_t}(c_i) = V_{\pi_t}(\nu). \tag{28}$$

Thus, $V_{\pi_{t+1}}(\nu) \geq V_{\pi_t}(\nu)$.

By induction, this holds for all nodes in the tree, including the root node. Therefore, we get:

$$J(\pi_{t+1}) = V_{\pi_{t+1}}(G) \geq V_{\pi_t}(G) = J(\pi_t). \tag{29}$$

Adaptive Thresholding and Sample Efficiency: In iterative settings, the threshold θ should be an adaptive factor increasing from 0.5 to 1. In our setting, where all data is collected offline, 0.5 is a balanced threshold to use for the following reasons:

- Lower Thresholds ($\theta < 0.5$): This allows for greater sample-efficiency as more branches labeled as correct are used as part of training. However, it might introduce noise into the training process with samples that have low values being chosen as correct.
- Higher Thresholds ($\theta > 0.5$): This would allow for a focus on actions leading to higher-value nodes, reducing variance. However, having θ too high would reduce sample efficiency.

Using $\theta = 0.5$ provides a balance suitable for a single iteration based on an offline generated dataset. By formalizing our pooling and value iteration approach, we have shown that updating the policy based on the binarized pooled reward with thresholding at $\theta = 0.5$ leads to monotonic improvement in the expected reward. This holds because increasing the probability of selecting actions (outputs) leading to next nodes with higher $\hat{V}(s')$ does not decrease the expected reward at any node, and thus the overall expected reward improves or remains the same.

A.4.2. Policy optimizing the DPO objective is identical to Optimal RL Policy

To support our claims in Appendix A.4.1, we leverage Theorem 1 from (Putta et al., 2024) and Theorem 6.1 from (Setlur et al., 2024), adjusted for our binarization setting:

Theorem. Consider a policy π that optimizes our objective over trajectories generated by a reference policy π_{ref}. At each node (state) h_t, preferences between actions during DPO are generated according to:

$$p(a_t^w \succ a_t^l \mid h_t) \propto \sigma\left(\hat{Q}(h_t, a_t^w) - \hat{Q}(h_t, a_t^l)\right), \tag{30}$$

where:

- a_t^w and a_t^l are two win/loss actions at node h_t,
- $\hat{Q}(h_t, a) \in \{0, 1\}$ is the binarized value function, representing the expected reward of action a at state h_t,

Then, the policy that optimizes the Direct Preference Optimization (DPO) objective is identical to the optimal RL policy:

$$\pi^*(a \mid h_t) \propto \pi_{\text{ref}}(a \mid h_t) \exp\left(\frac{\hat{Q}(h_t, a)}{\beta}\right), \tag{31}$$

where β is the DPO hyperparameter.

The proof for *Theorem 1* in (Putta et al., 2024) shows that the policy π^* approximates the optimal RL policy. That is, we can approximate the optimal RL policy if we generate preferences under the optimal value function (or an approximation thereof, i.e. our binarized version as shown below).

In our setting, since $\hat{Q}(h_t, a) \in \{0, 1\}$, the exponential term simplifies to:

- $\exp\left(\frac{1}{\beta}\right)$ when $\hat{Q}(h_t, a) = 1$,
- 1 when $\hat{Q}(h_t, a) = 0$.

Therefore, the optimized policy becomes:

$$\pi^*(a \mid h_t) \propto \begin{cases} \pi_{\text{ref}}(a \mid h_t) \exp\left(\frac{1}{\beta}\right), & \text{if } \hat{Q}(h_t, a) = 1, \\ \pi_{\text{ref}}(a \mid h_t), & \text{if } \hat{Q}(h_t, a) = 0. \end{cases} \tag{32}$$

This means that the policy π^* increases the probability of selecting actions with $\hat{Q}(h_t, a) = 1$ by a constant factor relative to the reference policy π_{ref}. By optimizing the DPO objective with these binarized preferences, we ensure that the policy increasingly favors actions leading to higher expected rewards, aligning with our credit assignment strategy described in Appendix A.4. This supports our claim of (approximate) monotonic improvement, as the policy updates move us closer to the optimal policy by consistently selecting actions associated with higher binarized values.

A.5. Future Directions

Our findings showcase the potential of multi-agent LLM systems optimized with fine-tuning and collaborative inference techniques. There are several future directions from this line of work: Using PPO (Schulman et al., 2017b) and the exact value propogated backward for each trajectory to update model weights, possibly in an online RL fashion, might produce strong results with additional computational overhead (Ivison et al., 2024). Moreover, we provide several levers around the number of models (where the three model setup can be used iteratively), controlling the branching factor for data generation, examining the effect of majority voting on more samples, changing the attribution threshold, or treating the attribution threshold as an adaptive parameter when iteratively training and rolling out from the multi-agent system (see Appendix A.6). Moreover, prompt-tuning strategies and different roles can be considered or distillation techniques. We note that these are all specific and interesting directions. However, they lie beyond the scope of this paper, where our goal is to introduce a new multi-agent post-training methodology and demonstrate strong empirical performance.

A.6. Additional Information

For SFT, we used LoRA with a learning rate multiplier of 0.1 and a batch size of 8 to avoid overfitting. For preference optimization, we used Direct Preference Optimization (DPO) with a preference tuning learning rate multiplier to 0.1, training beta parameter of 0.2, and adapter weight configured to 0.2. We varied the number of epochs between 1 to 10 based on the size of the synthetic dataset for each model and leave a deeper exploration of hyperparameter configurations that could require a significant amount of compute to future work. SFT training was often until convergence. DPO training did not necessarily converge by the end of all iterations.

We keep our prompts the same for every baseline and trained model on a given benchmark. Our prompts use CoT and zero-shot prompting. We use a temperature of 0.3 for Llama 3.1 8B Instruct since it was qualitatively good enough to prevent hallucinations and still led to diverse enough samples. MALT requires the presence of an initial training set containing question-answer pairs, which led to the use of MATH, CSQA, and GSM8K.

During inference for the data collection strategy, using an exponential branching factor does not add significant compute overhead because inference calls can be parallelized when sampling from a model with the exact same input. Moreover, during training, we obtain a fixed dataset upon which LoRA fine-tuning can be conducted. LoRA adapters ensure that the model weights themselves aren't duplicated, thus requiring only minimal additional memory for the adapters themselves while the base models remain the same.

A.7. Safety

Our approach can be used not just to enhance the reasoning capabilities of LLM systems, but also address crucial open problems in the safety of multi-agent systems. Importantly, MALT-trained systems of trusted small models could attain better task performance while retaining high degrees of trust, producing more powerful overseers within the AI control setting (Greenblatt et al., 2024). Another prominent application of our approach would be to train verifiers as safety critics within a multi-agent setup. This could scale up the settings such as OpenAI CriticGPT (McAleese et al., 2024) to any number of models, resulting in more powerful safety critics and allowing for the legibility of solutions to be improved.

A.8. Limitations and Ethics Statement

We note that even at low temperatures, model performance on benchmarks often exhibits high variance. To address this within our computational constraints, we conducted evaluations on subsets of test-sets (size 100) across four seeds. While CommonsenseQA is known to contain many biased or incorrectly labelled questions (Geva et al., 2019), we utilized it in a manner consistent with prior work. We aim to make all our training data, inference logs, and code publicly available.