SEGO: Sequential Subgoal Optimization for Mathematical Problem-Solving

Xueliang Zhao[♠*] **Xinting Huang**[★] **Wei Bi**[★] **Lingpeng Kong**[♠]
[♠]The University of Hong Kong
[★]Tencent AI Lab
xlzhao22@connect.hku.hk

Abstract

Large Language Models (LLMs) have driven substantial progress in artificial intelligence in recent years, exhibiting impressive capabilities across a wide range of tasks, including mathematical problem-solving. Inspired by the success of subgoal-based methods, we propose a novel framework called **SE**quential sub**G**oal **O**ptimization (SEGO) to enhance LLMs' ability to solve mathematical problems. By establishing a connection between the subgoal breakdown process and the probability of solving problems, SEGO aims to identify better subgoals with theoretical guarantees. Addressing the challenge of identifying suitable subgoals in a large solution space, our framework generates problem-specific subgoals and adjusts them according to carefully designed criteria. Incorporating these optimized subgoals into the policy model training leads to significant improvements in problem-solving performance. We validate SEGO's efficacy through experiments on two benchmarks, GSM8K and MATH, where our approach outperforms existing methods, highlighting the potential of SEGO in AI-driven mathematical problem-solving.

1 Introduction

In recent years, the emergence of Large Language Models (LLMs) has marked a significant milestone in the field of artificial intelligence. Models such as ChatGPT and LLaMA have demonstrated remarkable capabilities across diverse tasks. Within this context, addressing mathematical problems has attracted considerable interest from researchers, as it serves as a prominent showcase of the reasoning capabilities inherent in LLMs. Reasoning involves a multitude of aspects, among which the ability to decompose the overall problem into smaller, more manageable subproblems (i.e., subgoals) is particularly essential for effective problem-solving.

In this paper, we draw inspiration from the successful application of subgoal-based methods in both RL and LLMs (Zhang et al., 2020; Zhao et al., 2023) and introduce a novel framework called SEGO (SEquential subGoal Optimization). Intuitively, a good subgoal should serve as a bridge to solving a bigger problem, such that breaking down the problem into these subgoals makes the subproblems easier to solve, thereby increasing the likelihood of solving the entire problem. SEGO quantifies this intuition by establishing a theoretical connection between the subgoal breakdown process and the probability of solving the problem (Eq. 6). Concretely, we construct a lower bound on the probability of solving the complete problem using a proposal distribution considering a specific subgoal. We then employ a method inspired by annealed importance sampling (Neal, 2001) to efficiently navigate through vast search spaces, seeking the subgoal corresponding to the theoretically optimal proposal distribution, while ensuring the process doesn't get trapped in suboptimal subgoals (§3.2). By incorporating these sequentially optimized subgoals into the training of the policy model, we achieve significant improvements in solving mathematical problems.

To empirically validate the efficacy of SEGO, we conducted experiments on two primary benchmarks: GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021). Our approach demonstrated marked superiority against existing methods with comparable model sizes, highlighting the potential of SEGO in advancing the field of AI-driven mathematical problem-solving. We hope that our findings can open up new avenues for future research on the applicability of LLMs to complex tasks in diverse domains (Yao et al., 2022; Liu et al., 2023).

[*] This work was done during an internship at Tencent AI Lab.

Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 7544–7565

2 Preliminaries

2.1 Problem Formulation

This study focuses on a goal-conditioned reinforcement learning (RL) framework, consisting of goal space ($\mathcal{G}$), state space ($\mathcal{S}$), action space ($\mathcal{A}$), transition probability ($\mathcal{P}$), and reward function ($\mathcal{R}$). The transition probability $\mathcal{P}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$ indicates the probability of transitioning from a current state $\boldsymbol{s}$ to a new state $\boldsymbol{s}'$ after an action $\boldsymbol{a}$. The reward function $\mathcal{R}(\boldsymbol{s}, \boldsymbol{g})$ gives a reward of 1 if the goal $\boldsymbol{g}$ is reached at state $\boldsymbol{s}$, and 0 otherwise. The policy $\pi(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g})$ maps state-goal pairs to actions in $\mathcal{A}$.

Building on the goal-conditioned RL framework, in mathematical problem-solving, an *action* denotes a step in the solution process, while the *state* comprises cumulative actions. The *(sub-)goal* is the specific problem targeted for resolution. The transition probability, $\mathcal{P}(s'|s, a)$, uniquely assigns a probability of 1 to the state $s' = [s; a]$ where $[;]$ represents sequence concatenation, and 0 to all others. The reward function $\mathcal{R}(s, g)$ evaluates whether state s correctly solves the goal g. In this work, we employ a program-aided approach (Gao et al., 2023; Chen et al., 2022; Drori et al., 2022) to form the solution. For illustration, consider the goal $\boldsymbol{g}$ as "Calculate $\sin(30°)$". Here, a state $\boldsymbol{s}$ could be "import math; def solve(): angle = math.radians(30);", and an action $\boldsymbol{a}$, "return math.sin(angle)". This work aims to create a policy network that predicts trajectories for new goals. It uses a demonstration dataset $\mathcal{D} = \{\tau : (\boldsymbol{g}; \boldsymbol{s}_0, \boldsymbol{a}_0, \ldots, \boldsymbol{s}_\ell, \boldsymbol{a}_\ell)\}$, where τ represents a trajectory of length ℓ with states $\boldsymbol{s}_t$ and actions $\boldsymbol{a}_t$ at every timestep. The special state, $\hat{\boldsymbol{s}}$, consists solely of essential imports and function definitions. The task is to predict a trajectory, starting from $\hat{\boldsymbol{s}}$ and aligned with a given goal $\boldsymbol{g}$. This is formulated as:

$$p(\tau \mid \hat{\boldsymbol{s}}, \boldsymbol{g}) = \prod_{t=0}^{\ell} \pi(\boldsymbol{a}_t \mid \boldsymbol{s}_t, \boldsymbol{g}) \cdot \mathcal{P}(\boldsymbol{s}_{t+1} \mid \boldsymbol{a}_t, \boldsymbol{s}_t), \quad \text{with } \boldsymbol{s}_0 = \hat{\boldsymbol{s}}. \tag{1}$$

2.2 Subgoal-based Reinforcement Learning

The main idea behind subgoal-based RL involves decomposing a challenging task into two more manageable sub-tasks, each of which can be addressed by the existing policy (Li et al., 2022). In subgoal-based RL, a typical approach consists of three phases: subgoal collection, trajectory sampling, and training, which together form a cyclical process.

The subgoal collection phase is central to this framework and follows a "generate-select" pipeline. Specifically, for a challenging goal $\boldsymbol{g}$, the process *generates* a variety of potential subgoals, each paired with its respective state. A suitable subgoal, $\boldsymbol{g}_w$, and its state, $\boldsymbol{s}_w$[1], are then *selected* based on criteria that vary among different algorithms (Li et al., 2022; Zhang et al., 2021; Chane-Sane et al., 2021). These criteria typically ensure that the chosen subgoal is attainable from the initial state and facilitates achieving the final goal. For example, the subgoal might be "Calculate the radian value of 30°" with the state "import math; def solve(): angle = math.radians(30);". In the trajectory sampling phase, trajectories τ_1 and τ_2 are drawn from the distributions $p(\tau|\hat{\boldsymbol{s}}, \boldsymbol{g}_w)$ and $p(\tau|\boldsymbol{s}_w, \boldsymbol{g})$ respectively. The final training phase utilizes these trajectories to optimize the policy network, thereby enhancing the ability to achieve both subgoals and the ultimate goal.

3 Method

This work addresses challenges in subgoal-based RL, focusing on the suboptimality of generated subgoals and their selection process's lack of theoretical guarantees. We introduce the SEGO framework, which innovates beyond the traditional "generate-select" pipeline. SEGO employs a "generate-(sequentially) optimize-select" approach (Figure 1), encompassing a policy network, subgoal generator, subgoal optimizer, reward network, and value network. Notably, only the policy network is used in the testing phase.

The "generate-(sequentially) optimize-select" pipeline starts with initial subgoal generation, followed by sequential optimizations. In each iteration, a new subgoal is proposed and evaluated for its increased likelihood of achieving the goal. Improved subgoals are retained for further refinement. This results in a collection of refined subgoals, from which the most suitable are selected based on specific criteria.

SEGO presents substantial advantages: (1) Its sequential optimization aligns generated subgoals more closely with an optimal subgoal distribution. (2) It accurately calculates subgoal weights based on an unbiased estimate of the probability of reaching a goal from a given state.

[1] In this work, the subscript "w" denotes "waypoint", which is used interchangeably with "subgoal".

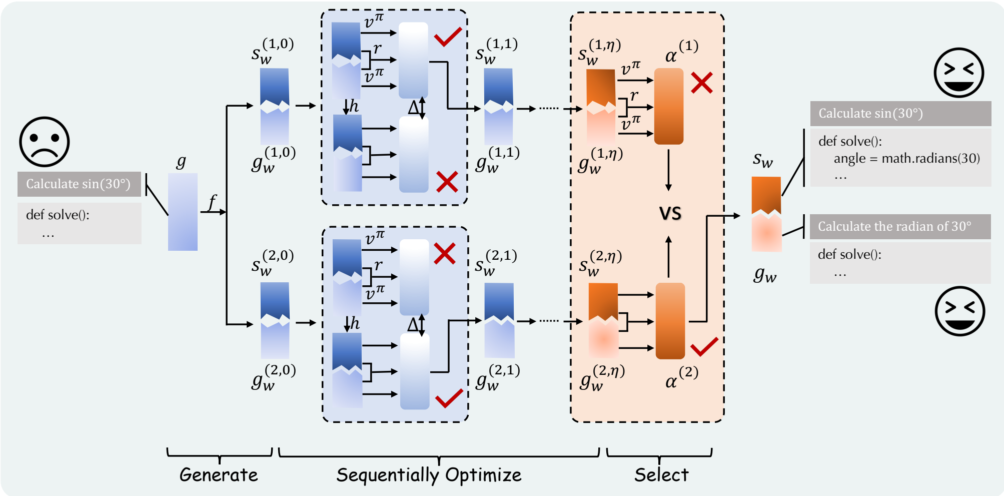

Figure 1: An overview of the "generate-(sequentially) optimize-select" pipeline. Within this pipeline, the symbols f, h, r, and v^π correspond to the subgoal generator, subgoal optimizer, reward network, and value network, respectively. The terms g, s_w, and g_w denote the intended goal, the subgoal state, and the subgoal. The pipeline initiates by generating a diverse set of subgoals. Each subgoal is then optimized in sequence. The process ends with the selection of the most appropriate subgoal.

Road Map. We start by discussing the initialization fine-tuning in §3.1, which includes setting up key components and preparing initial training data. Next, we detail subgoal-based fine-tuning in §3.2, the core of our framework. This section explains the "generate-(sequentially) optimize-select" process and how the resultant data updates various component parameters. The overall algorithm is outlined in §3.3.

3.1 Initialization Fine-tuning

The SEGO framework employs the following key components: policy network, subgoal generator, subgoal optimizer, reward network, and value network, all of which are implemented using large language models (LLMs) (Touvron et al., 2023a,b; Rozière et al., 2023). We defer the training details of these components after an overview of their implementation. More details about these components are provided in Appendix B.

Policy Network. The policy network $\pi(\boldsymbol{a} \mid \boldsymbol{s}, \boldsymbol{g})$ processes the current state and goal, represented as token sequences, to predict actions. This network employs standard decoding methods like greedy search or top-k sampling (Holtzman et al., 2019).

Subgoal Generator and Subgoal Optimizer. The subgoal generator f, represented as $\boldsymbol{s}_w, \boldsymbol{g}_w = f(\boldsymbol{s}, \boldsymbol{g})$, breaks complex tasks into simpler subtasks, transforming the current state and goal into a subgoal and its associated state. This method ensures manageable progression towards the ultimate goal. The subgoal optimizer h, denoted as $\boldsymbol{s}'_w, \boldsymbol{g}'_w = h(\boldsymbol{s}_w, \boldsymbol{g}_w, \boldsymbol{s}, \boldsymbol{g})$, refines these subgoals and states to improve goal decomposition efficiency.

Reward Network and Value Network. The reward network $r(\boldsymbol{s}, \boldsymbol{g})$ evaluates if the current state achieves the goal, acting as a surrogate for the actual reward function $\mathcal{R}$. The value network $v^\pi(\boldsymbol{s}, \boldsymbol{g})$, a regression model, assesses the success probability from a given state under policy π.

Training of the SEGO components begins with a goal collection, $\mathcal{D}_g = \{\boldsymbol{g}\}$, and a trajectory dataset, $\mathcal{D} = \{\tau : (\boldsymbol{g}; \boldsymbol{s}_0, \boldsymbol{a}_0, \boldsymbol{s}_1, \boldsymbol{a}_1, \ldots)\}$, created using GPT-3.5-turbo.[2] This dataset comprises various mathematical problem-solving trajectories. The policy network is trained with triplets $(\boldsymbol{g}, \boldsymbol{s}_i, \boldsymbol{a}_i)$ from $\mathcal{D}$ to predict action $\boldsymbol{a}_i$ for a given state $\boldsymbol{s}_i$ and goal $\boldsymbol{g}$. For each trajectory, a state $\boldsymbol{s}_t$ is randomly chosen, and GPT-3.5-turbo predicts the intermediate subgoal $\boldsymbol{g}_w$ and its state $\boldsymbol{s}_w$. This step trains the subgoal generator to predict subgoals and their states from $(\boldsymbol{s}_t, \boldsymbol{g})$. GPT-3.5-turbo also introduces slight modifications to $\boldsymbol{g}_w$ and $\boldsymbol{s}_w$, producing $\tilde{\boldsymbol{g}}_w$ and $\tilde{\boldsymbol{s}}_w$. The subgoal optimizer is trained to restore $(\boldsymbol{g}_w, \boldsymbol{s}_w)$ from these corrupted versions, considering the current state $\boldsymbol{s}_t$ and goal $\boldsymbol{g}$.

After initializing the policy network, it gener-

[2]Further details regarding the trajectory dataset are elaborated in Appendix C.

Algorithm 1 SEGO: Sequential Subgoal Optimization

Requires: π, f, h, r, v^{π}: policy network, subgoal generator, subgoal optimizer, reward network, value network, respectively.
K_{max}: maximum iterations for subgoal-based fine-tuning.
$\mathcal{D}_g$: a collection of goals (or mathematical problems).

Construct the trajectory dataset $\mathcal{D}$ using GPT-3.5-turbo and $\mathcal{D}_g$.
Initialize and fine-tune π, f, h, r, and v^{π} with $\mathcal{D}_g$ and $\mathcal{D}$.
$k \leftarrow 0$.
while $k < K_{\text{max}}$ **do**
  $\mathcal{D}_p, \mathcal{D}_v \leftarrow \emptyset$. ▷ Prepare datasets $\mathcal{D}_p$ for policy and $\mathcal{D}_v$ for value network training.
  for $\tau \in \mathcal{D}$ **do** ▷ Each τ is a trajectory in the form $(\boldsymbol{g}; \boldsymbol{s}_0, \boldsymbol{a}_0, \ldots, \boldsymbol{s}_t, \boldsymbol{a}_t, \ldots)$.
    Generate a diverse set of subgoals via Eq.2.
    Optimize each subgoal using Eq.3 and Eq.4.
    Select a subgoal via Eq.5.
    Sample new trajectories τ_1 and τ_2 utilizing the selected subgoal.
    Calculate $\bar{\alpha}$ and form a triplet $(\boldsymbol{g}, \boldsymbol{s}_t, \bar{\alpha})$. ▷ $\bar{\alpha}$ estimates the probability of achieving $\boldsymbol{g}$ from $\boldsymbol{s}_t$ under π.
    Update $\mathcal{D}_p \leftarrow \mathcal{D}_p \cup \{\tau_1, \tau_2\}$, $\mathcal{D}_v \leftarrow \mathcal{D}_v \cup \{(\boldsymbol{g}, \boldsymbol{s}_t, \bar{\alpha})\}$.
  Train policy network π with $\mathcal{D}_p$ and value network v^{π} with $\mathcal{D}_v$.
  $k \leftarrow k + 1$.

ates trajectories for each goal in $\mathcal{D}_g$, with goals linked to human-annotated answers. Trajectories leading to correct answers are positive examples, $\{\tau : (\boldsymbol{g} : \boldsymbol{s}_0, \boldsymbol{a}_0, \ldots, \boldsymbol{s}_\ell, \boldsymbol{a}_\ell)\}$, and those missing the correct answers are negative examples, $\{\tau : (\boldsymbol{g} : \tilde{\boldsymbol{s}}_0, \tilde{\boldsymbol{a}}_0, \ldots, \tilde{\boldsymbol{s}}_\ell, \tilde{\boldsymbol{a}}_\ell)\}$. The reward network is trained to classify the final state-goal pair $(\boldsymbol{s}_\ell, \boldsymbol{g})$ as positive and $(\tilde{\boldsymbol{s}}_\ell, \boldsymbol{g})$ as negative. Simultaneously, the value network trains to approximate to 1 for $(\boldsymbol{s}_t, \boldsymbol{g})$ and 0 for $(\tilde{\boldsymbol{s}}_t, \boldsymbol{g})$, where $\boldsymbol{s}_t$ and $\tilde{\boldsymbol{s}}_t$ are randomly selected from their respective sets.

3.2 Subgoal-based Fine-tuning

The policy network, when only fine-tuned at the initialization phase, struggles with complex problems (Luo et al., 2023). Inspired by recent advancements in subgoal-based RL (Li et al., 2022; Zhang et al., 2021; Chane-Sane et al., 2021) and annealed importance sampling (Neal, 2001), we introduce a fine-tuning stage that emphasizes decomposing tasks into subgoals. Additionally, it evolves from the traditional “generate-select” pipeline to a more advanced “generate-(sequentially)optimize-select” approach.

Subgoal Collection. For each trajectory $\tau : (\boldsymbol{g}; \boldsymbol{s}_0, \boldsymbol{a}_0, \boldsymbol{s}_1, \boldsymbol{a}_1, \ldots) \in \mathcal{D}$, this phase aims to generate a subgoal pair $(\boldsymbol{s}_w, \boldsymbol{g}_w)$ that decompose $\boldsymbol{g}$ into more manageable subtasks. The procedure starts by generating N independent pairs of initial subgoals, denoted as $\{(\boldsymbol{s}_w^{(i,1)}, \boldsymbol{g}_w^{(i,1)})\}_{i=1}^{N}$. Subsequently, each pair $(\boldsymbol{s}_w^{(i,1)}, \boldsymbol{g}_w^{(i,1)})$ proceeds through a sequential optimization process, which yields a sequence of subgoal pairs: $\{(\boldsymbol{s}_w^{(i,1)}, \boldsymbol{g}_w^{(i,1)}), \ldots, (\boldsymbol{s}_w^{(i,\eta)}, \boldsymbol{g}_w^{(i,\eta)})\}$, where η represents the maximum number of iterations within the sequential optimization.

Within each trajectory τ, a state $\boldsymbol{s}_t$ is randomly selected from the set $\{\boldsymbol{s}_0, \boldsymbol{s}_1, \ldots\}$. Subsequently, the subgoal generator is tasked with producing a series of subgoals, defined as follows:

$$\boldsymbol{s}_w^{(i,1)}, \boldsymbol{g}_w^{(i,1)} = f(\boldsymbol{s}_t, \boldsymbol{g}), \quad \text{for } i = 1, \ldots, N \tag{2}$$

To ensure the generation of diverse subgoals, a top-k sampling strategy (Holtzman et al., 2019) is implemented.

The pipeline then progresses to a sequential optimization process. At the j-th iteration, the subgoal optimizer proposes a potentially improved subgoal pair, which is defined as:

$$\begin{aligned} \boldsymbol{s}_w^{(i,j)}, \boldsymbol{g}_w^{(i,j)} &= h(\boldsymbol{s}_w^{(i,j-1)}, \boldsymbol{g}_w^{(i,j-1)}, \boldsymbol{s}_t, \boldsymbol{g}), \\ &\text{for } i = 1, \ldots, N \end{aligned} \tag{3}$$

To ensure the improvement of the new subgoal pair $(\boldsymbol{s}_w^{(i,j)}, \boldsymbol{g}_w^{(i,j)})$ over its predecessor $(\boldsymbol{s}_w^{(i,j-1)}, \boldsymbol{g}_w^{(i,j-1)})$, it is necessary to establish a rigorous criteria for evaluation. To do that, a criteria is derived from a theoretical perspective to guarantee an unbiased estimation, as detailed in Proposition 4.3. Formally, the criteria is defined as:

$$\begin{aligned} \Delta =& \beta_{j-1} \log \frac{p(\boldsymbol{s}_w^{(i,j)}, \boldsymbol{g}_w^{(i,j)} \mid \boldsymbol{s}_t, \boldsymbol{g}; f)}{p(\boldsymbol{s}_w^{(i,j-1)}, \boldsymbol{g}_w^{(i,j-1)} \mid \boldsymbol{s}_t, \boldsymbol{g}; f)} \\ &+ (1 - \beta_{j-1}) \log \Bigg(\frac{v^{\pi}(\boldsymbol{s}_w^{(i,j)}, \boldsymbol{g})}{v^{\pi}(\boldsymbol{s}_w^{(i,j-1)}, \boldsymbol{g})} \times \frac{v^{\pi}(\boldsymbol{s}_t, \boldsymbol{g}_w^{(i,j)})}{v^{\pi}(\boldsymbol{s}_t, \boldsymbol{g}_w^{(i,j-1)})} \\ &\times \frac{\exp\left(r(\boldsymbol{s}_w^{(i,j)}, \boldsymbol{g}_w^{(i,j)})\right)}{\exp\left(r(\boldsymbol{s}_w^{(i,j-1)}, \boldsymbol{g}_w^{(i,j-1)})\right)} \Bigg) \\ &+ \log \frac{p(\boldsymbol{s}_w^{(i,j-1)}, \boldsymbol{g}_w^{(i,j-1)} \mid \boldsymbol{s}_w^{(i,j)}, \boldsymbol{g}_w^{(i,j)}, \boldsymbol{s}_t, \boldsymbol{g}; h)}{p(\boldsymbol{s}_w^{(i,j)}, \boldsymbol{g}_w^{(i,j)} \mid \boldsymbol{s}_w^{(i,j-1)}, \boldsymbol{g}_w^{(i,j-1)}, \boldsymbol{s}_t, \boldsymbol{g}; h)} \end{aligned} \tag{4}$$

where the sequence of weights β_j satisfies $1 = \beta_0 > \beta_1 > \ldots > \beta_\eta = 0$. If $\Delta \leq 0$, the subgoal pair at the j-th step is redefined as $(\boldsymbol{s}_w^{(i,j-1)}, \boldsymbol{g}_w^{(i,j-1)})$; otherwise, $(\boldsymbol{s}_w^{(i,j)}, \boldsymbol{g}_w^{(i,j)})$ is maintained. Intuitively, as the coefficient β_j approaches 0, the criteria increasingly emphasizes the comparison between the values of two subgoals within the optimal distribution (Proposition 4.2), represented in logarithmic form. Specifically, $v^\pi(\boldsymbol{s}_w, \boldsymbol{g})$ and $v^\pi(\boldsymbol{s}, \boldsymbol{g}_w)$ serve as proxies for $p^\pi(\boldsymbol{g} \mid \boldsymbol{s}_w)$ and $p^\pi(\boldsymbol{g}_w \mid \boldsymbol{s})$, respectively. This comparison favors the subgoal that better aligns with the optimal distribution, thus incrementally steering the subgoal optimization towards more theoretically effective choices. The final term in Δ acts as a regularization factor.

The weight $\alpha^{(i)}$ associated with each subgoal pair $(\boldsymbol{s}_w^{(i,\eta)}, \boldsymbol{g}_w^{(i,\eta)})$ is defined as follows:

$$\log \alpha^{(i)} = \sum_{j=1}^{\eta} \Bigg[(\beta_j - \beta_{j-1}) \log p(\boldsymbol{s}_w^{(i,j)}, \boldsymbol{g}_w^{(i,j)} \mid \boldsymbol{s}_t, \boldsymbol{g}; f) + (\beta_{j-1} - \beta_j) \Big(\log v^\pi(\boldsymbol{s}_w^{(i,j)}, \boldsymbol{g}) + \log v^\pi(\boldsymbol{s}_t, \boldsymbol{g}_w^{(i,j)}) + r(\boldsymbol{s}_w^{(i,j)}, \boldsymbol{g}_w^{(i,j)}) \Big) \Bigg] \tag{5}$$

Subsequently, the subgoal pair $(\boldsymbol{s}_w, \boldsymbol{g}_w)$ is selected based on a softmax distribution over these weights, i.e., $(\boldsymbol{s}_w, \boldsymbol{g}_w) \sim \mathrm{Softmax}(\log \alpha^{(i)})$.

Trajectory Sampling and Component Training. Upon obtaining a trajectory τ and the predicted subgoal pair $(\boldsymbol{s}_w, \boldsymbol{g}_w)$, the subsequent procedure involves generating two new trajectories through the policy network. These trajectories, denoted as τ_1 and τ_2, are sampled from $p(\tau \mid \boldsymbol{s}_t, \boldsymbol{g}_w)$ and $p(\tau \mid \boldsymbol{s}_w, \boldsymbol{g})$ (defined in Eq.1), respectively. Within these trajectories, for each triplet $(\boldsymbol{g}, \boldsymbol{s}_i, \boldsymbol{a}_i)$, the policy network is trained to predict the action $\boldsymbol{a}_i$ given the state $\boldsymbol{s}_i$ and the goal $\boldsymbol{g}$.

As a byproduct of this sequential optimization process, the average coefficient $\bar{\alpha} = \frac{1}{N}\sum_{i=1}^{N} \alpha^{(i)}$ acts as an unbiased estimator that correlates with the probability of successfully achieving the goal $\boldsymbol{g}$ from the state $\boldsymbol{s}_t$ when guided by the policy network π (see Proposition 4.3). Leveraging this byproduct, the value network is further trained to regress towards $\bar{\alpha}$, using the state $\boldsymbol{s}_t$ and the goal $\boldsymbol{g}$ as inputs.

3.3 SEGO: Sequential Subgoal Optimization

After completing the initialization phase, our approach involves repeated cycles of subgoal collection, trajectory sampling and component training. This procedure leads to the development of our final framework, SEGO, detailed in Algorithm 1.

Remarks. In this work, we concentrate on mathematical problem-solving, yet our proposed methodology serves as a universal framework for tackling a wide range of complex tasks that can be modeled as goal-conditioned reinforcement learning problems (see §2.1), including code generation (Chen et al., 2021) and commonsense reasoning (Clark et al., 2018). To do that, one only needs to customize the goal, action, and state space definitions to suit the task specifics and adjust prompts for trajectory generation and subgoal prediction using GPT-3.5-turbo, aligning them with the specific requirements of the task.

4 Theoretical Analysis

We begin by constructing a lower bound on the probability of successfully solving the complete problem. This is done through the consideration of a proposal distribution focused on a specific subgoal. Letting $p^{\pi(\cdot|\cdot,\boldsymbol{g})}(\boldsymbol{g} \mid \boldsymbol{s})$ represent the probability of achieving a goal $\boldsymbol{g}$ from a state $\boldsymbol{s}$ under policy $\pi(\cdot \mid \cdot, \boldsymbol{g})$, we have the following proposition:

Proposition 4.1. *The objective defined below constitutes a lower bound on the probability of reaching the goal g from state s:*

$$\log p^{\pi(\cdot|\cdot,\boldsymbol{g})}(\boldsymbol{g} \mid \boldsymbol{s}) \geq \mathbb{E}_{q(\boldsymbol{g},\boldsymbol{s}|\boldsymbol{g},\boldsymbol{s})} \Big[\log p^{\pi(\cdot|\cdot,\boldsymbol{g})}(\boldsymbol{g} \mid \boldsymbol{s}_w) + \log p^{\pi(\cdot|\cdot,\boldsymbol{g})}(\boldsymbol{g}_w \mid \boldsymbol{s}) + r(\boldsymbol{s}_w, \boldsymbol{g}_w) - \log q(\boldsymbol{s}_w, \boldsymbol{g}_w \mid \boldsymbol{s}, \boldsymbol{g}) \Big]. \tag{6}$$

We provide the proof in Appendix A.1. Next, we derive the analytical solution for the optimal subgoal distribution and obtain the following proposition.

Proposition 4.2. *The optimal subgoal distribution satisfies the following condition:*

$$q^\star(\boldsymbol{s}_w, \boldsymbol{g}_w \mid \boldsymbol{s}, \boldsymbol{g}) = \frac{p^{\pi(\cdot|\cdot,\boldsymbol{g})}(\boldsymbol{g} \mid \boldsymbol{s}_w) p^{\pi(\cdot|\cdot,\boldsymbol{g})}(\boldsymbol{g}_w \mid \boldsymbol{s}) \exp(r(\boldsymbol{s}_w, \boldsymbol{g}_w))}{Z},$$
$$\textit{where } Z = \iint p^{\pi(\cdot|\cdot,\boldsymbol{g})}(\boldsymbol{g} \mid \boldsymbol{s}'_w) p^{\pi(\cdot|\cdot,\boldsymbol{g})}(\boldsymbol{g}'_w \mid s) \times \exp(r(\boldsymbol{s}'_w, \boldsymbol{g}'_w)) \mathrm{d}\boldsymbol{s}'_w \mathrm{d}\boldsymbol{g}'_w. \tag{7}$$

We provide the proof in Appendix A.2. Proposition 4.2 reveals that the optimal subgoal should not only be reachable from the starting point but also aid in ultimately reaching the final goal. We further investigate the ability of SEGO to provide an unbiased estimate of the Z. Inspired by annealed

Model	Base	Prompt	Params	GSM8K	MATH
GPT-4 (OpenAI, 2023)	-	CoT	-	92.0	42.5
PaLM-2 (Anil et al., 2023)	PaLM	CoT	540B	80.7	34.3
Minerva (Lewkowycz et al., 2022)	PaLM	CoT	540B	58.8	33.6
LLaMA2 (Touvron et al., 2023b)	LLaMA2	CoT	7B	14.6	2.5
			13B	28.7	3.9
WizardMATH (Luo et al., 2023)	LLaMA2	CoT	7B	54.9	10.7
			13B	63.9	14.0
MetaMath (Yu et al., 2023)	LLaMA2	CoT	7B	66.5	19.8
			13B	72.3	22.4
CodeLLaMA (Rozière et al., 2023)	CodeLLaMA	PoT	7B	25.2	14.2
			13B	36.1	18.1
MAmmoTH-Coder (Yue et al., 2023)	CodeLLaMA	PoT	7B	59.4	33.4
			13B	64.7	36.3
SEGO (ours)	**CodeLLaMA**	**PoT**	**7B**	**68.7**	**36.8**
			13B	**72.5**	**40.0**

Table 1: Evaluation results on GSM8K and MATH. "CoT" and "PoT" represent chain-of-thoughts (Wei et al., 2023) program-of-thoughts (Chen et al., 2022) respectively.

importance sampling (Neal, 2001), we arrive at the following proposition:

Proposition 4.3. *Let $\bar{\alpha}$ be defined as $\bar{\alpha} = \frac{1}{N}\sum_{i=1}^{N}\alpha^{(i)}$, wherein each $\alpha^{(i)}$ adheres to the definition in Eq.5. It follows that $\bar{\alpha}$ constitutes an unbiased estimator of Z.*

We provide the full proof of the unbiasedness in Appendix A.3. Proposition 4.3 reveals that the training objective for the value network can be approximated as a proportional estimate of the probability of attaining the goal $\boldsymbol{g}$ from state $\boldsymbol{s}$ following the current policy π.

5 Experiments

5.1 Dataset and Evaluation

Evaluation and Training Data. Our model is evaluated using two datasets: GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021). GSM8K contains $8,792$ math word problems for elementary students, with $1,319$ reserved for testing. MATH, with $12,500$ problems (including $5,000$ for testing), focuses on advanced mathematics, featuring questions from competitions like the AMC and AIME. Data preprocessing follows the methodologies in the original papers to ensure consistent evaluation. We provide the details of the training data in Appendix D.

Evaluation Metric. We evaluate by comparing the results of the solution generated by the policy network in SEGO to the provided correct answers within the datasets. For evaluation, we report the accuracy, specifically focusing on the rate at which the policy network correctly solves the problems on the first attempt.

5.2 Baselines

Due to space constraints, details on the baselines are available in Appendix D.

5.3 Main Results

As indicated in Table 1, our key findings include: (1) SEGO's performance on the GSM8K and MATH datasets is notable. SEGO (7B) achieves 68.7% accuracy on GSM8K and 36.8% on MATH, while SEGO (13B) reaches 72.5% and 40.0%, respectively. These results surpass those of comparable models, underscoring SEGO's effectiveness in mathematical problem-solving; and (2) The integration of finetuning and the Program of Thought (PoT) approach substantially enhances model performance, particularly in complex tasks. This is evident in SEGO and MetaMath, where finetuning aligns models with task specifics, and in comparisons involving CodeLLaMA and LLaMA2 on the MATH dataset, showcasing PoT's efficiency. Additionally, incorporating Sequential Subgoal Optimization into SEGO underlines the significance of strategic planning in complex mathematical problem-solving, resulting in notably improved accuracy.

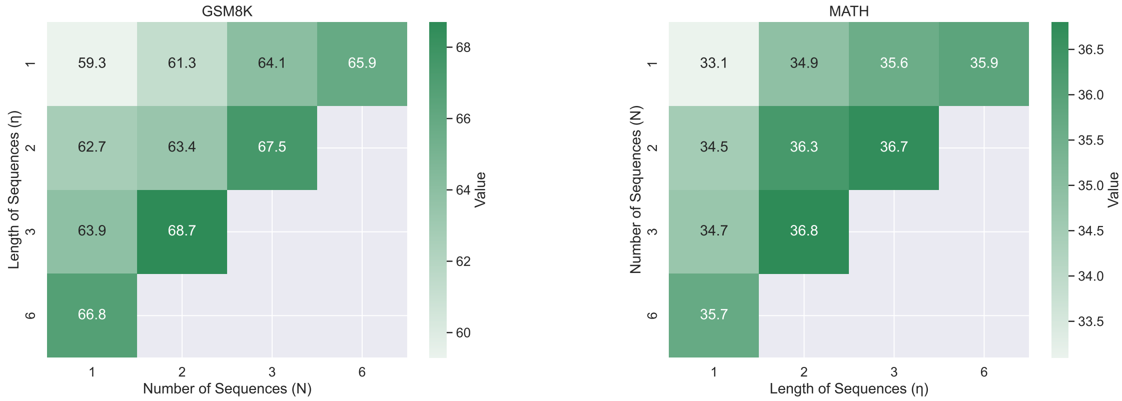

Figure 2: The balance between the number of sequences (N) and the length of sequences (η) on the test sets of GSM8K and MATH.

Models	GSM8K	MATH
Ours	68.7	36.8
-Sequential	61.3	34.9
-Sequential & Subgoal	57.1	32.6
-Sequential & Subgoal & FT	25.2	14.2

Table 2: Ablation study results on GSM8K and MATH datasets.

6 Analysis

6.1 Ablation Study

In our study, we conducted ablation experiments on 7B CodeLLaMA using SEGO and three variants to assess each component's impact: (1) **-Sequential**: the sequential subgoal optimization is omitted. (2) **-Sequential & Subgoal**: the subgoal-based finetuning is omitted. (3) **-Sequential & Subgoal & FT**: both subgoal-based finetuning and initialization fine-tuning are omitted. Results in Table 2 show the crucial role of sequential subgoal optimization in SEGO, with its absence in the **-Sequential** variant leading to reduced accuracy. The significant performance drop in the **-Sequential & Subgoal & FT** variant, comparable to the base 7B CodeLLaMA, highlights the collective value of all components in enhancing SEGO's mathematical problem-solving capabilities.

6.2 Analysis of Hyperparameters

In this section, we conduct a detailed examination of the hyperparameters N and η, where N represents the number of sequences and η denotes the length of each sequence, as defined in Proposition 4.3. All the experiments in this section are anchored on the 7B CodeLLaMA to ensure consistency in the results.

The balance between N and η. We begin by exploring various combinations of N and η, illustrated in Figure 2, to comprehend the synergistic effects of these parameters on the model's performance. The results on GSM8K and MATH reveal that incrementing both N and η typically enhances the model's accuracy, achieving 68.7% on GSM8K and 36.8% on MATH at $N = 2$ and $\eta = 3$. However, the enhancements appear to stabilize beyond certain thresholds, indicating optimal points for these parameters.

In-depth analysis of Hyperparameters N and η. We further conduct an in-depth analysis of the hyperparameters N and η, examining each one's individual impact by holding one constant and varying the other. The results are illustrated in Figure 3. From the results, it is clear that when $N = 2$, the model achieves peak accuracy at $\eta = 3$ for both GSM8K and MATH, with no significant gains beyond this point. Similarly, with $\eta = 3$, optimal accuracy is reached at $N = 2$, remaining stable thereafter.

6.3 Analysis of Subgoal Evolution

Validity and Progression of Subgoals. To deepen our understanding of subgoals during the Reinforcement Learning phase, we analyze the evolution of subgoal validity and its correlation with the performance on the test set. A subgoal (i.e., $\boldsymbol{g}_w$ and $\boldsymbol{s}_w$) is deemed valid if both τ_1 and τ_2, sampled with policies $\pi(\cdot|\boldsymbol{s}_w, \boldsymbol{g})$ and $\pi(\cdot|\boldsymbol{s}, \boldsymbol{g}_w)$, yield correct solutions for goals $\boldsymbol{g}$ and $\boldsymbol{g}_w$ respectively. Our findings, illustrated in Figure 4 (Left), reveal a positive correlation between the progression of training steps and the percentage of valid subgoals. This increase in valid subgoals is paralleled by improvements in accuracy on both GSM8K and MATH

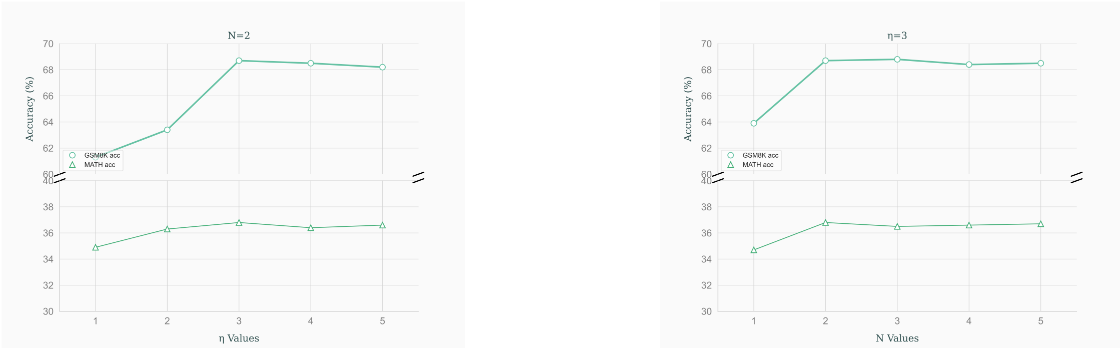

Figure 3: Analysis of model accuracy for variations N and η. Left: Fixed $N = 2$ and various η; Right: Fixed $\eta = 3$ and various N.

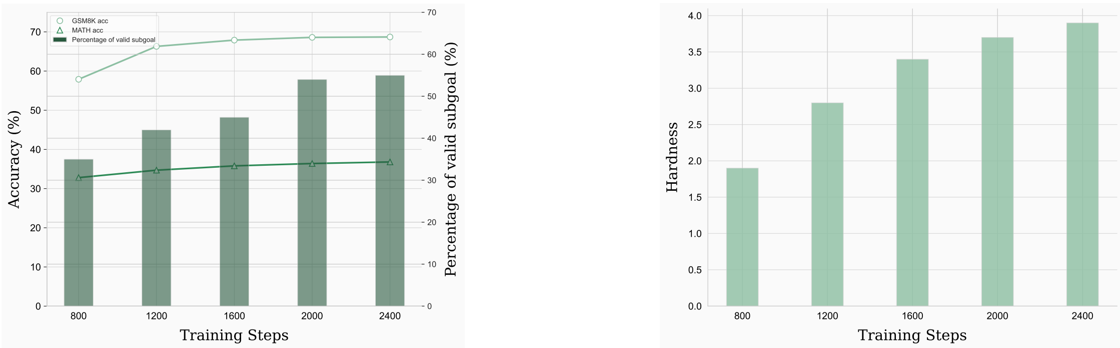

Figure 4: Left: Changes in the percentage of valid subgoals during the RL training. Right: Changes in hardness of problems yielding valid subgoals.

datasets, suggesting that the validity of subgoals is a crucial factor in enhancing the model's problem-solving capabilities.

Hardness of Problems Yielding Valid Subgoals. To further our understanding of subgoals, we delve into the relationship between the hardness of problems and the emergence of valid subgoals. This analysis aims to reveal any trends in the difficulty of problems that tend to yield valid subgoals, providing insights into the learning progression. The hardness of each problem is labeled by ChatGPT, with more details available in Appendix E. The results, shown in Figure 4 (Right), reveal a correlation between training progression and the model's ability to formulate valid subgoals for increasingly intricate problems, underscoring its evolving sophistication and adaptability in problem-solving.

7 Related Works

Mathematical Reasoning with LLMs. Large Language Models' (LLMs) advancement in mathematical reasoning is largely driven by datasets like GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021), with additional resources like MAWPS (Koncel-Kedziorski et al., 2016) and MWPToolkit (Lan et al., 2022) enhancing the field. Research focuses on two main areas: prompting strategies, involving techniques like Chain-of-Thought (Wei et al., 2023), Progressive-Hint Prompting (Zheng et al., 2023), bi-modal behavioral alignment (Zhao et al., 2024) and learning with verifications, using methods like outcome-based verifiers (Cobbe et al., 2021). Our approach, orthogonal to these methods, emphasizes adaptive curricula with subgoals to improve LLMs' mathematical reasoning. Concurrently, MAmmoTH (Yue et al., 2023) explores instruction finetuning in LLMs for math problem-solving, a concept related to our strategy. This can be considered as an implementation of the instruction finetuning stage within our framework.

Subgoal-based RL. In reinforcement learning, Subgoal Search is crucial for navigating complex tasks, offering insights into subgoal benefits (Zhai et al., 2022), hierarchical structures (Wen et al., 2020), option selection (Jinnai et al., 2019a), and temporal abstraction (Fruit et al., 2017). Research focuses on exploring efficient strategies (Jinnai et al., 2019b; Hartikainen et al., 2019; Pitis et al., 2020; OpenAI et al., 2021) and enhancing planning through various algorithms (Eysenbach et al., 2019; Parascandolo et al., 2020; Li et al., 2022; Moro et al., 2022; Chane-Sane et al., 2021). It also develops curricula for complex subgoals (Zhang et al., 2020, 2021). Our work addresses subgoal learning in mathematical problem-solving, exploring optimal subgoal identification within expansive state spaces. Owing to space constraints, a detailed discussion of related works is provided in Appendix F.

8 Conclusion

In conclusion, this work presents SEGO, an innovative framework aimed at improving LLMs' mathematical problem-solving abilities. Drawing inspiration from subgoal-based RL, SEGO establishes a theoretical link between subgoal decomposition and the probability of solving problems. It enhances LLMs' performance by generating and refining problem-specific subgoals using theoretically defined criteria. Empirical evaluations on benchmark datasets GSM8K and MATH demonstrate SEGO's ability to outperform existing approaches of comparable model sizes.

Ethical Considerations

In accordance with the established Code of Ethics, this research exclusively utilizes data and information that is publicly accessible, thereby ensuring that no private or confidential resources are engaged.

Limitations

While SEGO represents a significant advancement in the realm of mathematical problem-solving, several limitations need further investigation to fully harness its potential. These limitations include aspects such as the efficiency of SEGO, the scope of problem difficulty it addresses, and potential framework extensions:

(1) While SEGO demonstrates enhanced efficacy in identifying subgoals compared to non-sequential methods, there is room for improvement in efficiency. This can be addressed through dynamic resource allocation, such as adjusting the annealing schedule in response to performance metrics or the complexity of the problem at hand, alongside the deployment of more sophisticated proposal distribution mechanisms that more accurately mirror the target distribution.

(2) Our evaluation benchmarks predominantly include elementary to middle school-level problems. Exploring more complex problems, such as those at the undergraduate level, is a promising future direction.

(3) In the current SEGO framework, only the policy network is retained during inference. An intriguing future direction involves integrating the subgoal generator/optimizer and the value network to recursively decompose complex problems into simpler subgoals.

Acknowledgements

We would like to thank the HKU NLP group and the anonymous reviewers for their helpful suggestions, which greatly improved this work. We especially appreciated the valuable discussions with Shansan Gong. This work is partially supported by the joint research scheme of the National Natural Science Foundation of China (NSFC) and the Research Grants Council (RGC) under grant number N_HKU714/21.

References

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*.

Elliot Chane-Sane, Cordelia Schmid, and Ivan Laptev. 2021. Goal-conditioned reinforcement learning with imagined subgoals. In *International Conference on Machine Learning*, pages 1430–1440. PMLR.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2022. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Iddo Drori, Sarah Zhang, Reece Shuttleworth, Leonard Tang, Albert Lu, Elizabeth Ke, Kevin Liu, Linda Chen, Sunny Tran, Newman Cheng, et al. 2022. A neural network solves, explains, and generates university math problems by program synthesis and few-shot learning at human level. *Proceedings of the National Academy of Sciences*, 119(32):e2123433119.

Ben Eysenbach, Russ R Salakhutdinov, and Sergey Levine. 2019. Search on the replay buffer: Bridging planning and reinforcement learning. *Advances in Neural Information Processing Systems*, 32.

Ronan Fruit, Matteo Pirotta, Alessandro Lazaric, and Emma Brunskill. 2017. Regret minimization in mdps with options without prior knowledge. *Advances in Neural Information Processing Systems*, 30.

Yao Fu, Hao Peng, Ashish Sabharwal, Peter Clark, and Tushar Khot. 2023. Complexity-based prompting for multi-step reasoning.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361.

Kristian Hartikainen, Xinyang Geng, Tuomas Haarnoja, and Sergey Levine. 2019. Dynamical distance learning for semi-supervised and unsupervised skill discovery. *arXiv preprint arXiv:1907.08225*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2019. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Yuu Jinnai, David Abel, David Hershkowitz, Michael Littman, and George Konidaris. 2019a. Finding options that minimize planning time. In *International Conference on Machine Learning*, pages 3120–3129. PMLR.

Yuu Jinnai, Jee Won Park, David Abel, and George Konidaris. 2019b. Discovering options for exploration by minimizing cover time. In *International Conference on Machine Learning*, pages 3130–3139. PMLR.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. MAWPS: A math word problem repository. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1152–1157, San Diego, California. Association for Computational Linguistics.

Yihuai Lan, Lei Wang, Qiyuan Zhang, Yunshi Lan, Bing Tian Dai, Yan Wang, Dongxiang Zhang, and Ee-Peng Lim. 2022. Mwptoolkit: an open-source framework for deep learning-based math word problem solvers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 13188–13190.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. 2022. Solving quantitative reasoning problems with language models. *arXiv preprint arXiv:2206.14858*.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. 2023. Making large language models better reasoners with step-aware verifier.

Yunfei Li, Tian Gao, Jiaqi Yang, Huazhe Xu, and Yi Wu. 2022. Phasic self-imitative reduction for sparse-reward goal-conditioned reinforcement learning. In *International Conference on Machine Learning*, pages 12765–12781. PMLR.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *arXiv preprint arXiv:2305.20050*.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *ACL*.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. 2023. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. 2023. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *arXiv preprint arXiv:2308.09583*.

Lorenzo Moro, Amarildo Likmeta, Enrico Prati, Marcello Restelli, et al. 2022. Goal-directed planning via hindsight experience replay. In *International Conference on Learning Representations*, pages 1–16.

Radford M Neal. 2001. Annealed importance sampling. *Statistics and computing*, 11:125–139.

Ansong Ni, Jeevana Priya Inala, Chenglong Wang, Alex Polozov, Christopher Meek, Dragomir Radev, and Jianfeng Gao. 2023. Learning math reasoning from self-sampled correct and partially-correct solutions. In *The Eleventh International Conference on Learning Representations*.

OpenAI. 2023. Gpt-4 technical report.

OpenAI. 2023. GPT-4 Technical Report. *arXiv e-prints*, page arXiv:2303.08774.

OpenAI OpenAI, Matthias Plappert, Raul Sampedro, Tao Xu, Ilge Akkaya, Vineet Kosaraju, Peter Welinder, Ruben D'Sa, Arthur Petron, Henrique P d O Pinto, et al. 2021. Asymmetric self-play for automatic goal discovery in robotic manipulation. *arXiv preprint arXiv:2101.04882*.

Giambattista Parascandolo, Lars Buesing, Josh Merel, Leonard Hasenclever, John Aslanides, Jessica B Hamrick, Nicolas Heess, Alexander Neitz, and Theophane Weber. 2020. Divide-and-conquer monte carlo tree search for goal-directed planning. *arXiv preprint arXiv:2004.11410*.

Silviu Pitis, Harris Chan, Stephen Zhao, Bradly Stadie, and Jimmy Ba. 2020. Maximum entropy gain exploration for long horizon multi-goal reinforcement learning. In *International Conference on Machine Learning*, pages 7750–7761. PMLR.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2018. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *arXiv preprint arXiv:1811.00937*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models.

Zheng Wen, Doina Precup, Morteza Ibrahimi, Andre Barreto, Benjamin Van Roy, and Satinder Singh. 2020. On efficiency in hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 33:6708–6718.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2023. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. 2023. Mammoth: Building math generalist models through hybrid instruction tuning. *arXiv preprint arXiv:2309.05653*.

Yuexiang Zhai, Christina Baek, Zhengyuan Zhou, Jiantao Jiao, and Yi Ma. 2022. Computational benefits of intermediate rewards for goal-reaching policy learning. *Journal of Artificial Intelligence Research*, 73:847–896.

Tianjun Zhang, Benjamin Eysenbach, Ruslan Salakhutdinov, Sergey Levine, and Joseph E Gonzalez. 2021. C-planning: An automatic curriculum for learning goal-reaching tasks. *arXiv preprint arXiv:2110.12080*.

Yunzhi Zhang, Pieter Abbeel, and Lerrel Pinto. 2020. Automatic curriculum learning through value disagreement. *Advances in Neural Information Processing Systems*, 33:7648–7659.

Xueliang Zhao, Xinting Huang, Tingchen Fu, Qintong Li, Shansan Gong, Lemao Liu, Wei Bi, and Lingpeng Kong. 2024. Bba: Bi-modal behavioral alignment for reasoning with large vision-language models. *arXiv preprint arXiv:2402.13577*.

Xueliang Zhao, Wenda Li, and Lingpeng Kong. 2023. Decomposing the enigma: Subgoal-based demonstration learning for formal theorem proving. *arXiv preprint arXiv:2305.16366*.

Chuanyang Zheng, Zhengying Liu, Enze Xie, Zhenguo Li, and Yu Li. 2023. Progressive-hint prompting improves reasoning in large language models. *arXiv preprint arXiv:2304.09797*.

Denny Zhou, Nathanael Scharli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Olivier Bousquet, Quoc Le, and Ed Huai hsin Chi. 2022. Least-to-most prompting enables complex reasoning in large language models. *ArXiv*, abs/2205.10625.

A Proofs

A.1 Proof of proposition 4.1

In this subsection, we establish the proof of Proposition 4.1

Proof. We start by considering the joint distribution $p(\boldsymbol{g}, \boldsymbol{s}_w, \boldsymbol{g}_w \mid \boldsymbol{s})$, which can be factorized as $p^{\pi(\cdot|\cdot,\boldsymbol{g})}(\boldsymbol{g} \mid \boldsymbol{s}_w) p^{\pi(\cdot|\cdot,\boldsymbol{g}_w)}(\boldsymbol{g}_w \mid \boldsymbol{s}) p(\boldsymbol{s}_w \mid \boldsymbol{g}_w)$.

The log-likelihood of reaching the goal $\boldsymbol{g}$ from $\boldsymbol{s}$ can be expressed as:

$$\log p^{\pi(\cdot|\cdot,\boldsymbol{g})}(\boldsymbol{g} \mid \boldsymbol{s}) = \log \mathbb{E}_{q(\boldsymbol{g}_w, \boldsymbol{s}_w | \boldsymbol{g}, \boldsymbol{s})} \left[\frac{p(\boldsymbol{g}, \boldsymbol{s}_w, \boldsymbol{g}_w \mid \boldsymbol{s})}{q(\boldsymbol{g}_w, \boldsymbol{s}_w \mid \boldsymbol{g}, \boldsymbol{s})} \right]$$

Expanding the expectation, we get:

$$\log p^{\pi(\cdot|\cdot,\boldsymbol{g})}(\boldsymbol{g} \mid \boldsymbol{s}) = \log \iint q(\boldsymbol{g}_w, \boldsymbol{s}_w \mid \boldsymbol{g}, \boldsymbol{s}) \frac{p(\boldsymbol{g}, \boldsymbol{s}_w, \boldsymbol{g}_w \mid \boldsymbol{s})}{q(\boldsymbol{g}_w, \boldsymbol{s}_w \mid \boldsymbol{g}, \boldsymbol{s})} \mathrm{d}\boldsymbol{g}_w \mathrm{d}\boldsymbol{s}_w$$

Utilizing Jensen's inequality, we establish a lower bound for the log-likelihood as follows:

$$\begin{aligned}\log p^{\pi(\cdot|\cdot,\boldsymbol{g})}(\boldsymbol{g} \mid \boldsymbol{s}) \geq \mathbb{E}_{q(\boldsymbol{g}_w, \boldsymbol{s}_w | \boldsymbol{g}, \boldsymbol{s})} \Big[& \log p^{\pi(\cdot|\cdot,\boldsymbol{g})}(\boldsymbol{g} \mid \boldsymbol{s}_w) + \log p^{\pi(\cdot|\cdot,\boldsymbol{g}_w)}(\boldsymbol{g}_w \mid \boldsymbol{s}) \\ & + \log p(\boldsymbol{s}_w \mid \boldsymbol{g}_w) - \log q(\boldsymbol{g}_w, \boldsymbol{s}_w \mid \boldsymbol{g}, \boldsymbol{s}) \Big]\end{aligned}$$

Given that $\log p(\boldsymbol{s}_w \mid \boldsymbol{g}_w) = r(\boldsymbol{s}_w, \boldsymbol{g}_w) - \log \left(\sum_{s'_w} \exp(r(\boldsymbol{s}'_w, \boldsymbol{g}_w)) \right)$ and that $\log \left(\sum_{s'_w} \exp(r(\boldsymbol{s}'_w, \boldsymbol{g}_w)) \right)$ can be absorbed into the lower bound as a constant term, which does not affect the optimization process, the lower bound $\mathcal{L}$ can be written as:

$$\mathcal{L} = \mathbb{E}_{q(\boldsymbol{g}_w, \boldsymbol{s}_w | \boldsymbol{g}, \boldsymbol{s})} \left[\log p^{\pi(\cdot|\cdot,\boldsymbol{g})}(\boldsymbol{g} \mid \boldsymbol{s}_w) + \log p^{\pi(\cdot|\cdot,\boldsymbol{g}_w)}(\boldsymbol{g}_w \mid \boldsymbol{s}) + r(\boldsymbol{g}_w, \boldsymbol{s}_w) - \log q(\boldsymbol{g}_w, \boldsymbol{s}_w \mid \boldsymbol{g}, \boldsymbol{s}) \right] \tag{8}$$

This completes the proof of proposition 4.1. The underlying premise of this approach is predicated on the assumption that the ratio of exponentiated rewards, $\frac{\exp(r(\boldsymbol{s},\boldsymbol{g}))}{\exp(r(\boldsymbol{s}',\boldsymbol{g}'))}$, is equivalent to the ratio of the probabilities $\frac{p(\boldsymbol{s}|\boldsymbol{g})}{p(\boldsymbol{s}'|\boldsymbol{g}')}$. In essence, this implies that the reward function $r(\boldsymbol{s}, \boldsymbol{g})$ is directly proportional to the conditional probability $p(\boldsymbol{s} \mid \boldsymbol{g})$. □

A.2 Proof of proposition 4.2

In this subsection, we establish the proof of Proposition 4.2

Proof. The optimization objective for finding $q(\boldsymbol{g}_w, \boldsymbol{s}_w \mid \boldsymbol{g}, \boldsymbol{s})$ is:

$$\mathbb{E}_{q(\boldsymbol{g}_w, \boldsymbol{s}_w | \boldsymbol{g}, \boldsymbol{s})} \Big[\log p^{\pi(\cdot|\cdot,\boldsymbol{g})}(\boldsymbol{g} \mid \boldsymbol{s}_w) + \log p^{\pi(\cdot|\cdot,\boldsymbol{g}_w)}(\boldsymbol{g}_w \mid \boldsymbol{s}) + r(\boldsymbol{g}_w, \boldsymbol{s}_w) - \log q(\boldsymbol{g}_w, \boldsymbol{s}_w \mid \boldsymbol{g}, \boldsymbol{s}) \Big]$$

Introducing a Lagrange multiplier λ, the Lagrangian $\mathcal{J}$ is constructed as:

$$\begin{aligned}\mathcal{J} = \mathbb{E}_{q(\boldsymbol{g}_w, \boldsymbol{s}_w | \boldsymbol{g}, \boldsymbol{s})} \Big[& \log p^{\pi(\cdot|\cdot,\boldsymbol{g})}(\boldsymbol{g} \mid \boldsymbol{s}_w) + \log p^{\pi(\cdot|\cdot,\boldsymbol{g}_w)}(\boldsymbol{g}_w \mid \boldsymbol{s}) + r(\boldsymbol{g}_w, \boldsymbol{s}_w) \\ & - \log q(\boldsymbol{g}_w, \boldsymbol{s}_w \mid \boldsymbol{g}, \boldsymbol{s}) \Big] + \lambda \left(\int q(\boldsymbol{g}_w, \boldsymbol{s}_w \mid \boldsymbol{g}, \boldsymbol{s}) \mathrm{d}\boldsymbol{g}_w \mathrm{d}\boldsymbol{s}_w - 1 \right)\end{aligned}$$

Differentiating $\mathcal{J}$ with respect to $q(\boldsymbol{g}_w, \boldsymbol{s}_w \mid \boldsymbol{g}, \boldsymbol{s})$ and setting it to zero yields:

$$\log p^{\pi(\cdot|\cdot,\boldsymbol{g})}(\boldsymbol{g} \mid \boldsymbol{s}_w) + \log p^{\pi(\cdot|\cdot,\boldsymbol{g}_w)}(\boldsymbol{g}_w \mid \boldsymbol{s}) + r(\boldsymbol{g}_w, \boldsymbol{s}_w) - \log q(\boldsymbol{g}_w, \boldsymbol{s}_w \mid \boldsymbol{g}, \boldsymbol{s}) - 1 + \lambda = 0$$

Simplifying, we get:

$$q(\boldsymbol{g}_w, \boldsymbol{s}_w \mid \boldsymbol{g}, \boldsymbol{s}) = \exp(\lambda - 1) p^{\pi(\cdot|\cdot,\boldsymbol{g})}(\boldsymbol{g} \mid \boldsymbol{s}_w) p^{\pi(\cdot|\cdot,\boldsymbol{g}_w)}(\boldsymbol{g}_w \mid \boldsymbol{s}) \exp(r(\boldsymbol{g}_w, \boldsymbol{s}_w))$$

To ensure $q(\boldsymbol{g}_w, \boldsymbol{s}_w \mid \boldsymbol{g}, \boldsymbol{s})$ is a valid probability distribution, it is normalized as:

$$q^\star(\boldsymbol{g}_w, \boldsymbol{s}_w \mid \boldsymbol{g}, \boldsymbol{s}) = \frac{p^{\pi(\cdot|\cdot,\boldsymbol{g})}(\boldsymbol{g} \mid \boldsymbol{s}_w) p^{\pi(\cdot|\cdot,\boldsymbol{g}_w)}(\boldsymbol{g}_w \mid \boldsymbol{s}) \exp(r(\boldsymbol{g}_w, \boldsymbol{s}_w))}{\iint p^{\pi(\cdot|\cdot,\boldsymbol{g})}(\boldsymbol{g} \mid \boldsymbol{s}'_w) p^{\pi(\cdot|\cdot,\boldsymbol{g}'_w)}(\boldsymbol{g}'_w \mid \boldsymbol{s}) \exp(r(\boldsymbol{g}'_w, \boldsymbol{s}'_w)) \mathrm{d}\boldsymbol{g}'_w \mathrm{d}\boldsymbol{s}'_w}$$

The denominator serves as the normalizing constant, ensuring that $q^\star(\boldsymbol{g}_w, \boldsymbol{s}_w \mid \boldsymbol{g}, \boldsymbol{s})$ sums to one over its domain, thereby satisfying the properties of a probability distribution.

This concludes the proof. $\square$

A.3 Proof of proposition 4.3

For the sake of clarity, we define ω as the tuple $(\boldsymbol{g}_w, \boldsymbol{s}_w)$ and use $q^\star(\omega)$ as shorthand for $q^\star(\boldsymbol{g}_w, \boldsymbol{s}_w \mid \boldsymbol{g}, \boldsymbol{s}_0)$. To rigorously proof this proposition, we define a series of functions and transition operators:

Definition 1. *We introduce $f_j(\cdot)$ for $j \in \{0, \ldots, \eta\}$ as a weighted blend of $f_\eta(\cdot)$ and $p(\cdot \mid \boldsymbol{s}, \boldsymbol{g}; f)$, given by $f_j(\omega) = f_\eta(\omega)^{1-\beta_j} p(\omega \mid \boldsymbol{s}, \boldsymbol{g}; f)^{\beta_j}$. The sequence of weights β_j satisfies $1 = \beta_0 > \beta_1 > \ldots > \beta_\eta = 0$. Specifically, $f_\eta(\omega)$ satisfies $\frac{f_\eta(\omega)}{Z_f} = q^\star(\omega)$ where Z_f is the normalizing constant.*

Definition 2. *Let $T_j(\omega, \omega')$ for $j \in \{1, \ldots, \eta - 1\}$ denote a transition operator, formulated as*

$$T_j(\omega, \omega') = p(\omega' \mid \omega, \boldsymbol{s}, \boldsymbol{g}; h) \min\left(1, \frac{f_j(\omega') p(\omega \mid \omega', \boldsymbol{s}, \boldsymbol{g}; h)}{f_j(\omega) p(\omega' \mid \omega, \boldsymbol{s}, \boldsymbol{g}; h)}\right).$$

Then the process of sequentially sampling subgoals is defined as follows:

Definition 3. *Let the process start with the sampling of ω_1 from $f_0(\cdot)$. Sequentially, ω_2 is derived from ω_1 via the transition operator T_1, perpetuating this mechanism until ω_η is obtained from $\omega_{\eta-1}$ through $T_{\eta-1}$. The joint distribution probability is articulated as $\frac{g(\omega_1, \ldots, \omega_\eta)}{Z_g}$, wherein $g(\omega_1, \ldots, \omega_\eta) = f_0(\omega_1) T_1(\omega_1, \omega_2) \ldots T_{\eta-1}(\omega_{\eta-1}, \omega_\eta)$ and Z_g is the normalization constant.*

Finally, the weight α for each sequence is given by $\alpha = \prod_{j=1}^{\eta} \frac{f_j(w_j)}{f_{j-1}(\omega_j)}$.

To establish the validity of the proposition, we begin by proving the essential lemmas:

Lemma 1. *Let $f_j(\omega)$ and $T_j(\omega, \omega')$ be as specified in Definition 2. Define $p_j(\omega)$ as*

$$p_j(\omega) = \frac{f_j(\omega)}{\int f_j(\omega') \, \mathrm{d}\omega'}.$$

Then, the following detailed balance condition holds:

$$p_j(\omega) T_j(\omega, \omega') = p_j(\omega') T_j(\omega', \omega).$$

Proof. The proof can be divided into two cases:

Case 1: $p_j(\omega') p(\omega \mid \omega', \boldsymbol{s}, \boldsymbol{g}; h) > p_j(\omega) p(\omega' \mid \omega, \boldsymbol{s}, \boldsymbol{g}; h)$

Starting with $p_j(\omega') T_j(\omega', \omega)$, we have:

$$\begin{aligned} p_j(\omega') T_j(\omega', \omega) &= \cancel{p_j(\omega')} \cancel{p(\omega \mid \omega', \boldsymbol{s}, \boldsymbol{g}; h)} \frac{p_j(\omega) p(\omega' \mid \omega, \boldsymbol{s}, \boldsymbol{g}; h)}{\cancel{p_j(\omega')} \cancel{p(\omega \mid \omega', \boldsymbol{s}, \boldsymbol{g}; h)}} \\ &= p_j(\omega) p(\omega' \mid \omega, \boldsymbol{s}, \boldsymbol{g}; h) \\ &= p_j(\omega) T_j(\omega, \omega'). \end{aligned}$$

Case 2: $p_j(\omega')p(\omega \mid \omega', \boldsymbol{s}, \boldsymbol{g}; h) \leq p_j(\omega)p(\omega' \mid \omega, \boldsymbol{s}, \boldsymbol{g}; h)$
Starting with $p_j(\omega)T_j(\omega, \omega')$, we have:

$$\begin{aligned}
p_j(\omega)T_j(\omega, \omega') &= \cancel{p_j(\omega)p(\omega' \mid \omega, \boldsymbol{s}, \boldsymbol{g}; h)} \frac{p_j(\omega')p(\omega \mid \omega', \boldsymbol{s}, \boldsymbol{g}; h)}{\cancel{p_j(\omega)p(\omega' \mid \omega, \boldsymbol{s}, \boldsymbol{g}; h)}} \\
&= p_j(\omega')p(\omega \mid \omega', \boldsymbol{s}, \boldsymbol{g}; h) \\
&= p_j(\omega')T_j(\omega', \omega).
\end{aligned}$$

In both cases, we find that $p_j(\omega)T_j(\omega, \omega') = p_j(\omega')T_j(\omega', \omega)$, thereby proving the lemma. $\square$

Lemma 2. *Let $f_j(\omega)$ and $T_j(\omega, \omega')$ be as defined in Definition 2. Define the normalized distribution $p_j(\omega)$ as*

$$p_j(\omega) = \frac{f_j(\omega)}{\int f_j(\omega')\, \mathrm{d}\omega'}.$$

Then, $T_j(\omega, \omega')$ preserves the invariance of $p_j(\omega)$, formally defined as

$$\int T_j(\omega', \omega)p_j(\omega')\, \mathrm{d}\omega' = p_j(\omega).$$

Proof. We proceed by leveraging the results from Lemma 1. Specifically, we have:

$$\begin{aligned}
\int T_j(\omega', \omega)p_j(\omega')\, \mathrm{d}\omega' &= \int T_j(\omega, \omega')p_j(\omega)\, \mathrm{d}\omega' \\
&= p_j(\omega) \int T_j(\omega, \omega')\, \mathrm{d}\omega'
\end{aligned}$$

Given that $\int T_j(\omega, \omega')\, \mathrm{d}\omega' = 1$, we have $\int T_j(\omega', \omega)p_j(\omega')\, \mathrm{d}\omega' = p_j(\omega)$. This confirms that $T_j(\omega, \omega')$ preserves the invariance of $p_j(\omega)$, thereby proving Lemma 2. $\square$

Now we give the proof of Proposition 4.3.

Proof. We first define the function f as follows:

$$f(\omega_1, \ldots, \omega_\eta) = \frac{f_\eta(\omega_\eta)}{f_{\eta-1}(\omega_\eta)} T_{\eta-1}(\omega_{\eta-1}, \omega_\eta) \ldots \frac{f_2(\omega_2)}{f_1(\omega_2)} T_1(\omega_1, \omega_2) f_1(\omega_1)$$

Given the definition of Z_f, we have

$$Z_f = \int f_\eta(\omega)\, \mathrm{d}\omega$$

By Lemma 2, we have:

$$\int T_j(\omega_j, \omega_{j+1}) f_j(\omega_j)\, \mathrm{d}\omega_j = f_j(\omega_{j+1})$$

Thus, we can write:

$$\begin{aligned}
&\int \frac{f(\omega_1, \cdots, \omega_\eta)}{Z_f} \mathrm{d}\omega_1 \cdots \mathrm{d}\omega_\eta \\
=& \int \frac{f_\eta(\omega_\eta)}{Z_f} \mathrm{d}\omega_\eta \int \frac{T_{\eta-1}(\omega_{\eta-1}, \omega_\eta) f_{\eta-1}(\omega_{\eta-1})}{f_{\eta-1}(\omega_\eta)} \mathrm{d}\omega_{\eta-1} \cdots \int \frac{T_1(\omega_1, \omega_2) f_1(\omega_1)}{f_1(\omega_2)} \mathrm{d}\omega_1 \\
=& \int \frac{f_\eta(\omega_\eta)}{Z_f} \mathrm{d}\omega_\eta \\
=& 1
\end{aligned}$$

This implies that Z_f is also the normalizing constant of $f(\omega_1, \ldots, \omega_\eta)$.

Since $f_0(\cdot)$ is a distribution, it is evident that $Z_g = 1$.

We have:

$$\begin{aligned}
\mathbb{E}_{g(\omega_1,\ldots,\omega_\eta)}\left[\frac{1}{N}\sum\alpha\right] &= \mathbb{E}_{g(\omega_1,\ldots,\omega_\eta)}\left[\frac{f(\omega_1,\ldots,\omega_\eta)}{g(\omega_1,\ldots,\omega_\eta)}\right] \\
&= Z_f\left[\int\frac{f(\omega_1,\ldots,\omega_\eta)}{Z_f}\,\mathrm{d}\omega_1\cdots\mathrm{d}\omega_\eta\right] \\
&= Z_f
\end{aligned}$$

This concludes the proof of Proposition 4.3. □

B More Implementation Details for Each Module

The framework of SEGO is composed of five components, each serving a distinct purpose to enhance the system's overall efficacy.

B.1 Policy Network

The policy network $\pi(\boldsymbol{a} \mid \boldsymbol{s}, \boldsymbol{g})$ takes as input the current state and intended goal and returns an action. Since the goal and the state can both be expressed as token sequences, we first concatenate these sequences before feeding them into the policy network. This network is tasked with predicting the subsequent action, also framed as a token sequence, utilizing standard decoding techniques like greedy search or top-k sampling (Holtzman et al., 2019).

The training of the policy network is conducted through instruction finetuning, utilizing the following instruction template:

```
Construct a Python script to address the given problem:
{problem}

### Response:
{solution}
```

In this template, `problem` and `solution` represent the goal g and the trajectory respectively. The base model for this process is CodeLLaMA, and it undergoes full parameter finetuning to optimize its performance. As the sequential subgoal optimization process progresses, the model is further trained by utilizing self-generated successful trajectories. This prompt template is also employed to generate the trajectory dataset using `gpt-3.5-turbo-0613`.

B.2 Subgoal Generator

The subgoal generator, represented as f, aims to decompose a complex task into two more manageable sub-tasks. It works by taking the current state $\boldsymbol{s}$ and goal $\boldsymbol{g}$, and outputting a pair consisting of a subgoal and its corresponding state: $\boldsymbol{s}_w, \boldsymbol{g}_w = f(\boldsymbol{s}, \boldsymbol{g})$. This approach ensures that both the journey from the current state to the subgoal and from the subgoal state to the intended goal become more tractable sub-tasks. Crucially, the subgoal state $\boldsymbol{s}_w$ is a valid solution of the subgoal $\boldsymbol{g}_w$, adhering to the premise that a state is an aggregation of actions, each representing a step in the solution process.

The subgoal generator is trained through instruction finetuning, utilizing data collected from `gpt-3.5-turbo-0613`. The instruction template is defined as:

```
Break down the given problem into a smaller task (a subproblem)
and devise a method to solve it, considering a provided partial
solution to the original problem as a starting point.

### Input:
{problem}
```

```
{partial solution}

### Output:
{subproblem}{solution}[EOS]
```

This module, fundamentally built on the architecture of CodeLLaMA (Rozière et al., 2023), leverages the capabilities of LoRA (Hu et al., 2021) for efficient finetuning. The primary objective is to accurately predict `{subproblem}{solution}[EOS]` from its preceding context, realized through a causal language modeling. This prompt template is also utilized to predict both the subgoal and the corresponding state using `gpt-3.5-turbo-0613`.

B.3 Subgoal Optimizer

The subgoal optimizer, denoted as h, is designed to refine subgoal $\boldsymbol{g}_w$ and its corresponding state $\boldsymbol{s}_w$. Its objective is to yield improved subgoal $\boldsymbol{g}'_s$ and state $\boldsymbol{s}'_w$ that more effectively contribute to decomposing the overall intended goal: $\boldsymbol{s}'_w, \boldsymbol{g}'_w = h(\boldsymbol{s}_w, \boldsymbol{g}_w, \boldsymbol{s}, \boldsymbol{g})$. This component incorporates both the current state $\boldsymbol{s}$ and the intended goal $\boldsymbol{g}$ as inputs, providing insights into the complexity of the intended goal and the current status in the problem-solving process.

The subgoal optimizer is also trained through instruction finetuning, drawing upon data from `gpt-3.5-turbo-0613`. The instruction template for this module is as follows:

```
Optimize the given subproblem to make it more manageable. Then,
develop a method to solve it, considering a provided partial solution
to the original problem as a starting point.

### Input:
{problem}

{partial solution}

{subproblem}{solution}

### Output:
{optimized subproblem}{optimized solution}[EOS]
```

This module, also built on CodeLLaMA, utilizes LoRA for efficient parameter finetuning. The aim here is to accurately predict `{optimized subproblem}{optimized solution}[EOS]` from the provided context, ensuring the outputs are coherent and contextually aligned.

B.4 Reward Network

The reward network, formulated as $r(\boldsymbol{s}, \boldsymbol{g})$, accepts a state $\boldsymbol{s}$ and a goal $\boldsymbol{g}$ as inputs and produces a score to evaluate whether the goal has been achieved in the current state. In mathematical problem-solving, this essentially translates to determining if the state $\boldsymbol{s}$—which is an aggregate of executed actions, with each action representing a step towards the solution—is a valid solution for the problem posed by $\boldsymbol{g}$. Given that the actual reward function, $\mathcal{R}$ (described in §2.1), is applicable only to problems where a ground-truth answer is available, the reward network serves as a surrogate that is crucial for evaluating sub-problems encountered during the algorithm's execution.

This model is built on the architecture of CodeLLaMA and employs LoRA to achieve efficient finetuning. The reward model is trained through instruction finetuning, utilizing the following instruction template:

```
Does the provided solution accurately address the given problem?
{problem} {solution} {Y/N}.
```

B.5 Value Network

The value network, represented as $v^{\pi}(\boldsymbol{s}, \boldsymbol{g})$, is a regression model that takes a state $\boldsymbol{s}$ and an intended goal $\boldsymbol{g}$ as inputs, and outputs a score representing the likelihood of successfully achieving the goal from the state under the policy network π.

This model is trained to approximate the estimated $\hat{\alpha}$, utilizing instruction finetuning. The instruction template is defined as:

```
Determine the probability of resolving the problem, starting from
the partial solution: {problem} {partial solution}.
```

This model, built on the CodeLLaMA architecture, is finetuned using LoRA. It is noted that, during each iteration of the sequential subgoal optimization process, a unique set of LoRA parameters is used to avoid any potential discrepancies between iterations. This approach ensures that the value network accurately reflects the real-time capabilities of the policy network.

C Details about Trajectory Dataset Creation

To construct the goal collection $\mathcal{D}_g$ in Alg. 1, we incorporate mathematical problems sourced from the training subsets of three distinct datasets: GSM8k, MATH, and AQuA. For the generation of solutions corresponding to each problem, we apply a prompt as follows:

```
### Instruction
Construct a Python script to address the given problem:
{problem}

### Response:
{solution}
```

In this format, "solution" is completed by GPT-3.5-turbo. The solution is subsequently broken down into steps, with the i-th state in a trajectory comprising the first i steps, and the i-th action defined as the $(i+1)$-th step.

In a trajectory, all states except the s_0 which includes essential imports and function definitions (e.g., "import math; def solve():") are considered intermediate states.

D Details about Experimental Setup

D.1 Training Data

For training SEGO, we use GSM8K, MATH, and AQuA (Ling et al., 2017) datasets. After filtering for correct answers, the resulting training set includes $10,374$ samples from GSM8K, $10,981$ from MATH, and $35,355$ from AQuA. These problems form the goal collection $\mathcal{D}_g$ in Alg. 1.

D.2 Baselines

Closed-Source Models. (1) **GPT-4**: A model that sets a standard in various academic domains, including those that require intricate mathematical reasoning (OpenAI, 2023). (2) **PaLM-2**: A model that excels at logical reasoning and multilingual tasks, demonstrating advanced capabilities in reasoning and solving complex mathematical problems in multiple languages (Anil et al., 2023). (3) **Minerva**: A model that specializes in quantitative reasoning, providing precise and comprehensive solutions to advanced mathematical, scientific, and engineering problems (Lewkowycz et al., 2022).

Open-Source Models. (1) **LLaMA2**: A model that is trained on 2 trillion tokens of publicly accessible data, exhibits outstanding capabilities in mathematical reasoning (Touvron et al., 2023a). (2) **Wizard-MATH**: A model that enhances the mathematical reasoning capabilities of LLaMA2 by curating more complex and diverse supervised finetuning data (Luo et al., 2023). (3) **MetaMath**: This model employs a question bootstrapping technique, facilitating the generation of questions through both forward and backward reasoning paths. It further enhances its capabilities by incorporating Large Language Models (LLMs) to refine the phrasing of the question text (Yu et al., 2023). (4) **CodeLLaMA**: A model that excels in code-related tasks with implications in mathematical programming and algorithm synthesis, demonstrating superior infilling capabilities and support for extensive input contexts in programming tasks (Rozière et al., 2023).[3] (5) **MAmmoTH-Coder**: This model leverages a training dataset that incorporates both chain-of-thought (CoT) and program-of-thought (PoT) rationales, thereby not only facilitating the utilization of various tools but also accommodating diverse thought processes for solving distinct mathematical problems (Yue et al., 2023).

D.3 Implementation Details

We maintain model consistency by employing CodeLLaMA as the base model for both the policy network and auxiliary modules, including the subgoal generator, subgoal Optimizer, reward network, and value network. Efficient finetuning of the auxiliary modules is achieved through the utilization of LoRA (Hu et al., 2021), configured with parameters $r = 16$, lora_alpha $= 32$, and lora_dropout $= 0.05$, targeting the "q_proj" and "k_proj" modules. The learning rates are set at $1e-5$ and $1e-4$ for the policy and auxiliary modules, respectively, with a uniform batch size of 32. When collecting data from `gpt-3.5-turbo-0613`, we set temperature and top_p as 0.8 and 1.0 respectively. All models go through an initial training phase of $4,800$ steps. Subsequently, a sequential optimization process is conducted, with the number (N) and length (η) of sequences set as 2 and 3 respectively, and the temperature and top_p for the Subgoal GeneratorOptimizer and the policy network configured at 0.2 and 0.95 respectively. This optimization is performed three times, each lasting $1,200$ steps, and when $\eta = 3$, the parameters β_1 and β_2 are precisely set at 0.33 and 0.66 respectively. Rigorous contamination checking, as delineated by OpenAI (2023), is executed to verify the purity of our test sets for GSM8K and MATH. During the test phase, a greedy search strategy is employed.

E The Annotation of Problem Hardness

We employ the following prompt to automatically annotate the difficulty with `gpt-3.5-turbo-0613`:

```
Please assign a score between 1 and 5 to the following question,
indicating its level of difficulty and complexity. A higher score
should be given to denote greater difficulty and complexity.

Please provide only the score, without any additional explanations
or reasons.

### Input:
{question}

### Output:
```

F More Discussions about Related Works

Mathematical Reasoning with LLMs. The exploration of mathematical reasoning in Large Language Models (LLMs) has been significantly influenced by the development of datasets such as GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021), serving as crucial benchmarks for assessing machine learning models in mathematical domains. GSM8K encompasses a variety of grade school math problems, while MATH compiles challenging competition mathematics problems. The introduction of extensive datasets (Koncel-Kedziorski et al., 2016; Ling et al., 2017; Talmor et al., 2018; Geva et al., 2021) and platforms like MWPToolkit (Lan et al., 2022) has enriched the field. This exploration is systematically categorized into two main domains: prompting strategies and learning with verifications. In the realm of prompting strategies, a variety of methods have been conceptualized to enhance the reasoning capabilities of LLMs. Techniques such as Chain-of-Thought Prompting (Wei et al., 2023; Wang et al., 2022), Progressive-Hint Prompting (Zheng et al., 2023), Least-to-Most Prompting (Zhou et al., 2022), and

[3]For CodeLLaMA, we ensure consistency with our models by employing identical decoding methods and prompts during implementation, while for the other models, we refer to the results reported in their respective papers.

bi-modal behavioral alignment (Zhao et al., 2024) have been instrumental in progressively guiding LLMs to accurate conclusions and facilitating the generation of intermediate reasoning steps. Moreover, methodologies like Complexity-Based Prompting (Fu et al., 2023) and Self-Consistency(Wang et al., 2022) exploit higher reasoning complexity and diverse reasoning paths, respectively, to realize significant advancements in multi-step reasoning tasks. Within learning with verifications, the emphasis is on optimizing the mathematical proficiencies of LLMs through the integration of verifiers. Strategies like outcome-based verifiers (Cobbe et al., 2021), step-aware verifiers (Li et al., 2023; Lightman et al., 2023), and learning from partially-correct solutions (Ni et al., 2023) have been deployed to bolster reliability and precision in mathematical reasoning. While the aforementioned domains have significantly advanced mathematical reasoning within LLMs, our approach is orthogonal to these categories. We concentrate on the formulation of adaptive curricula, emphasizing the incorporation of subgoals, to facilitate nuanced learning pathways and enhance the model's mathematical reasoning capabilities. A parallel and notably concurrent work, MAmmoTH (Yue et al., 2023), investigates the impact of instruction finetuning to empower large language models with mathematical problem-solving capabilities. This can be considered as an implementation of the instruction finetuning stage within our framework.

Subgoal-based RL. Subgoal Search is a central component in reinforcement learning, essential for empowering AI systems to navigate through complex, extensive tasks effectively. This concept has played a vital role in uncovering important aspects such as the benefits of recognizing and rewarding subgoals (Zhai et al., 2022), the proper structuring of Markov decision processes for hierarchical reinforcement learning (Wen et al., 2020), the difficulties in selecting the most suitable options for planning (Jinnai et al., 2019a), and the incorporation of temporal abstraction in RL (Fruit et al., 2017). The practical research in this field mainly focuses on exploring and creating subgoals for planning and developing learning curricula for subgoals. Exploration is aimed at finding the best or most efficient strategies, using diverse approaches like reducing cover time (Jinnai et al., 2019b), understanding dynamical distances (Hartikainen et al., 2019), increasing entropy (Pitis et al., 2020), and applying asymmetric self-play (OpenAI et al., 2021). In the area of subgoal planning, a variety of algorithms have been developed to refine decision-making processes. For example, SoRB (Eysenbach et al., 2019) utilizes RL to develop a graph for subgoal sequences, DC-MCTS (Parascandolo et al., 2020) employs learned subgoal proposals to divide tasks, PAIR (Li et al., 2022) combines online RL with offline supervised learning, and (Moro et al., 2022) improve MCTS with Hindsight Experience Replay for goal-oriented planning. Moreover, the work by (Chane-Sane et al., 2021) provides concise insights into improving goal-conditioned reinforcement learning by conceptualizing imagined subgoals, adding a fresh viewpoint to the field. Research in curriculum learning has developed innovative methods to construct curricula that systematically escalate the complexity of subgoals, thereby improving the speed and quality of learning (Zhang et al., 2020, 2021). The exploration of subgoal learning in the realm of complex mathematical problem-solving represents a largely unexplored field. Our work delves into the inherent challenges of applying subgoal learning in mathematical contexts, specifically, the difficulty in identifying the optimal subgoal within expansive state spaces, and introduces a theoretical framework to navigate these challenges.

G Details about Time-complexity

This section presents the analysis of time-complexity for the sequential subgoal optimization process (see §3.2). For each example, the frequency of module invocation is shown in Table 3.

We acknowledge that the primary computational cost in our method stems from the decoding process conducted by the subgoal generator or optimizer. This process indeed requires significantly more time compared to the computation involved in calculating scores. Notably, as shown in §6.2, our method outperforms other approaches that produce more subgoals without sequential optimization, while maintaining a comparable computational budget. This result indicates the effectiveness of SEGO in identifying vital subgoals.

Modules	Times
Value Network (score calculation)	$2 \times N \times \eta$
Reward Network (score calculation)	$N \times \eta$
Subgoal Generator (score calculation)	$N \times \eta$
Subgoal Optimizer (score calculation)	$2 \times N \times \eta$
Subgoal Generator (decoding)	N
Subgoal Optimizer (decoding)	$N \times (\eta - 1)$

Table 3: The frequency of module invocation in the sequential subgoal optimization process.

H Analysis on Whether GPT-3.5-turbo Serves as an Upper Bound

This study seeks to explore the hypothesis that GPT-3.5-turbo represents a performance ceiling for SEGO. Moreover, it examines the applicability of SEGO when paired with more advanced foundational models, specifically employing Mistral, a language model with 7 billion parameters noted for its exceptional performance and efficiency (Jiang et al., 2023). In our experimental setup, both SEGO and GPT-3.5-turbo leverage a program-of-thought (Chen et al., 2022) rationale to ensure a fair comparison. The results are presented in Table 4.

Models	GSM8K	MATH
GPT-3.5-turbo	77.2	37.5
SEGO (with CodeLLaMA-13b)	72.5	40.0
SEGO (with Mistral-7b)	77.9	40.3

Table 4: Comparison of model performance across GSM8K and MATH benchmarks.

The results suggest that SEGO's performance potential is not limited by the upper limits of GPT-3.5-turbo. This point is particularly supported by the results in the MATH benchmark, where SEGO configurations utilizing Mistral-7b and CodeLLaMA-13b models significantly surpass the performance of GPT-3.5-turbo. This performance differential is predominantly attributed to the subgoal-based fine-tuning phase within SEGO, which enables the policy network to generate novel solutions that exceed the upper limits of GPT-3.5-turbo.

I Performance of Various Components

This section delves into the performance evaluation of key components within the SEGO framework.

Reward Network. The efficacy of the reward network was gauged through its performance on a binary classification task, aimed at determining the feasibility of achieving a goal state from a given state, as inferred from the reward scores. The classification accuracy achieved by the reward network is 62.8%.

Value Network. The performance of the value network was evaluated based on the metric $\text{recall}_1@10$, which reflects the network's ability to accurately identify viable subgoals from a set of ten candidates. The criteria for subgoal validity are detailed in §6.3. The results of this evaluation are presented in Table 5, illustrating the value network's performance improvements post after subgoal-based fine-tuning.

Models	$\text{recall}_1@10$
Value network (after initial fine-tuning)	36.1%
Value network (after subgoal-based fine-tuning)	52.7%

Table 5: Performance of the value network

Subgoal Generator and Optimizer. The assessment extended to the subgoal generator and the integrated approach combining subgoal generation with optimization. This combined method, denoted as "subgoal generator + optimizer", involves initially generating a subgoal followed by its refinement via the subgoal optimizer. The efficacy of these approaches, particularly in generating valid subgoals, is summarized in Table 6.

Models	**Percentage of valid subgoals**
subgoal generator	27.4%
subgoal generator + optimizer	29.5%

Table 6: Performance of the subgoal generator and optimizer.

J Case Study

Problem: A four-digit code can consist of the digits 0 ~ 9 except that it does not contain the digits 1, 2, and 4 at all. If repeated digits are allowed, what is the probability that it has at least one even digit?

Sub-problem: A four-digit code can consist of the digits 0 ~ 9 except that it does not contain the digits 1, 2, and 4 at all. If repeated digits are allowed, what is the probability that it has at least one even digit?

Solution:

```
def solve() -> float:
    total_codes = 0  # Total number of possible codes
    codes_with_even_digit = 0  # Number of codes with at least one even digit

    # Iterate through all four-digit codes
    for i in range(10000):
        code = str(i).zfill(4)  # Convert the code to a four-digit string

        # Check if the code contains only digits 0, 3, 5, 6, 7, 8, and 9
        if all(digit in ['0', '3', '5', '6', '7', '8', '9'] for digit in code):
            total_codes += 1

    return total_codes
```

Full solution:

```
def solve() -> float:
    total_codes = 0  # Total number of possible codes
    codes_with_even_digit = 0  # Number of codes with at least one even digit

    # Iterate through all four-digit codes
    for i in range(10000):
        code = str(i).zfill(4)  # Convert the code to a four-digit string

        # Check if the code contains only digits 0, 3, 5, 6, 7, 8, and 9
        if all(digit in ['0', '3', '5', '6', '7', '8', '9'] for digit in code):
            total_codes += 1

            # Check if the code has at least one even digit
            if any(digit in ['0', '6', '8'] for digit in code):
                codes_with_even_digit += 1

    # Calculate the probability as the ratio of codes with even digit to total codes
    probability = codes_with_even_digit / total_codes
    return probability
```

Figure 5: A case from the training data.

In this section, we delve into a specific example to illustrate the efficacy of our model, depicted in Figure 5. In this figure, the elements labeled as the problem, sub-problem, and solution (of the sub-problem) correspond to the final goal, intermediate goal, and intermediate state, respectively. The sub-problem showcased is derived through the sequential subgoal optimization process. Additionally, we provide the full solution, which is derived from the solution of the sub-problem. This case study indicates the model's capability to search for a suitable sub-problem that ultimately facilitates the derivation of the accurate solution to the final goal.