DIFFERENTIABLE EVOLUTIONARY REINFORCEMENT LEARNING

Sitao Cheng[1*], **Tianle Li**[2*], **Xuhan Huang**[3*], **Xunjian Yin**[4], **Difan Zou**[2]

[1]University of Waterloo [2]The University of Hong Kong
[3]The Chinese University of Hong Kong, Shenzhen [4]Duke University

sitao.cheng@uwaterloo.ca tianleli@connect.hku.hk xuhanhuang@link.cuhk.edu.cn

ABSTRACT

The design of reward functions presents an arduous challenge in reinforcement learning (RL). Existing automated reward modeling typically relies on derivative-free evolutionary heuristics that treat the reward function as a black box, failing to capture the causal relationship between structure changes and task performance. To bridge this gap, we propose **Differentiable Evolutionary Reinforcement Learning (DERL)**, a bi-level training framework for autonomous discovery of optimal reward signal. In DERL, a *Meta-Optimizer* evolves a reward function by composing structured atomic primitives, guiding the evolution of inner-loop policy. Crucially, DERL is differentiable in meta-optimization—updating the Meta-Optimizer via policy gradient derived from inner-loop validation performance. This allows the progressively learning of the "meta-gradient" of task success for denser and more actionable feedback. We validate DERL across three distinct domains: robotic agent (ALFWorld), scientific simulation (ScienceWorld), and mathematical reasoning (GSM8k, MATH). Results show that DERL achieves state-of-the-art performance on agent benchmarks, significantly outperforming non-differentiable methods, especially in out-of-distribution scenarios. Analysis of the evolutionary trajectory demonstrates that DERL successfully captures the intrinsic structure of tasks, enabling self-improving agent alignment without human intervention.

1 INTRODUCTION

The efficacy of reinforcement learning (RL) hinges fundamentally on the quality of the reward signal, the lens through which an agent perceives environment and learns toward desirable behaviors (Schulman et al., 2017; DeepSeek-AI et al., 2025). However, crafting optimal rewards remains a persistent bottleneck. In complex reasoning tasks, while outcome signals are often too sparse to drive learning over long horizons, manual reward design is prone to "reward hacking", where agents exploit specification flaws to maximize scores (Amodei et al., 2016; Yan et al., 2025). While Ouyang et al. (2022) adopts a dense reward model to ease the problem, it still heavily relies on human annotation.

As highlighted by "The Bitter Lesson" (Sutton, 2019), strategies relying on human priors are ultimately less scalable than general methods that leverage computation to learn directly from experience. Studies have pivoted towards automatic reward evolution, employing genetic algorithms to the reward (seen as an agent configuration) via stochastic mutations (Such et al., 2017; Jaderberg et al., 2017b) or another prompted agent (Shao et al., 2025; Zhang et al., 2025a; Novikov et al., 2025). In this context "*evolution*" refers to non-differentiable perturbation or prompt modification. A critical limitation is the black-box nature of reward evolution, which fails to model the non-arbitrary structure—*the causal relationship between the reward change and resulting agent performance shift*—blindly traversing the optimization landscape without exploiting the intrinsic structural logic that drives improvement.

When tuning a system, human experts intuitively maintains a *meta-gradient*—the *consciousness* that a specific reward change yields a specific behavior shift (Knox & Stone, 2009). We hypothesize that language models (LLMs) are able to capture the meta-gradient to update their own parameters,

[*]Equal Contribution

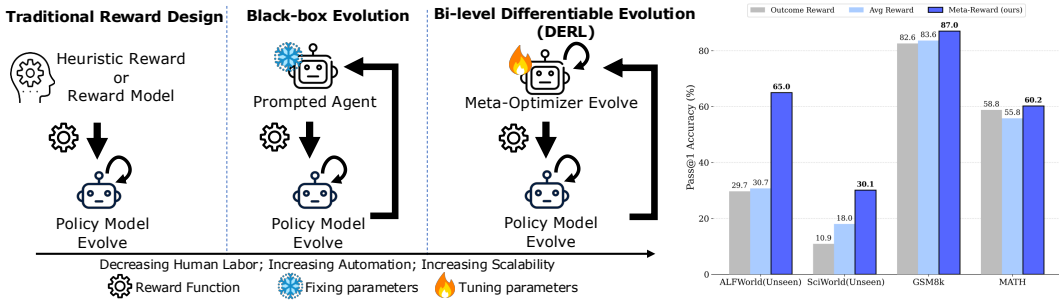

Figure 1: **Comparison:** Traditional reward design relies on manual labor or black-box evolution. In contrast, our DERL uses a Meta-Optimizer to generate a parameterized Meta-Reward. This guides policy evolution via gradients derived from validation performance, establishing a **differentiable, closed-loop** process. **Performance:** Meta-Reward consistently outperforms RL baselines, demonstrating the effectiveness of DERL.

thereby generating progressively better reward functions. To this end, we propose **Differentiable Evolutionary Reinforcement Learning (DERL)** for autonomous discovery of optimal objectives (Figure 1). Unlike manual design or black-box evolution, DERL is fully *differentiable*, updating a *Meta-Optimizer* via policy gradients. DERL features a **bi-level evolutionary training**: an *inner-loop* where the policy evolves based on the generated *Meta-Reward*, and an *outer-loop* where the Meta-Optimizer evolves by learning from the inner policy's validation signal (Figure 2). While DERL is generalizable to any configurations, we focus on **Reward Modeling**, primary driving agent behavior.

The instantiation of DERL addresses two challenges: *tractable action space* and *supervision for Meta-Optimizer*. First, instead of predicting an arbitrary function to introduce a vast search space, our Meta-Optimizer constructs rewards by composing *atomic primitives*—executable blocks like tools (*e.g.,* format checkers, partial goal verifiers) (Qin et al., 2023; Huang et al., 2024). This ensures expressiveness by inclusion of meaningful signals and the capture of structural logic rather than struggling with text parsing. Second, DERL uses *validation performance* of the inner-loop policy as direct feedback signal, alleviating human labor. By observing how different reward structures impact the policy's performance, the Meta-Optimizer approximates the gradient of task success, learning to generate Meta-Rewards with increasingly dense and actionable feedback signals via policy gradient.

We validate DERL on three domains: Robotic Agents (ALFWorld (Shridhar et al., 2020)), Scientific Simulation (ScienceWorld (Wang et al., 2022)), and Mathematical Reasoning (GSM8k (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021)). Results demonstrate that DERL generalizes effectively across diverse tasks, consistently outperforming other RL baselines. Notably, DERL achieves state-of-the-art performance on ALFWorld and ScienceWorld and shows superior robustness in OOD problems. Further analysis of evolutionary process reveals the capture of the meta-gradient: as training progresses, our Meta-Rewards evolve to encode the intrinsic logic of tasks, demonstrating a self-exploratory capability that aligns with the true gradient of optimization. Our contributions:

• We introduce **Differentiable Evolutionary Reinforcement Learning (DERL)**, a bi-level training framework that automates reward discovery. Unlike traditional black-box evolution, DERL enables the *Meta-Optimizer* to capture the gradient between reward structure changes and task performance, allowing it to update its own parameters to generate increasingly effective Meta-Rewards.

• We propose **a novel Meta-Optimizer architecture** that constructs rewards by composing *atomic primitives*—modular, executable blocks—rather than an arbitrary function. By utilizing the validation performance of the inner-loop policy as a supervision signal, we formulate reward generation as an RL problem, eliminating human annotation while ensuring a distinct, logical search space.

• We validate DERL across robotic, scientific, and mathematical domains. DERL achieves **state-of-the-art performance** on ALFWorld and ScienceWorld, demonstrating superior robustness in OOD scenarios. Analysis shows that our Meta-Optimizer successfully evolves to capture the intrinsic structure of tasks, progressively refining the reward signal without human intervention.

2 Differentiable Evolutionary Reinforcement Learning (DERL)

We first introduce the formulation of DERL framework, modeling reward design as a bi-level optimization process. We then detail the reward parameterization and training.

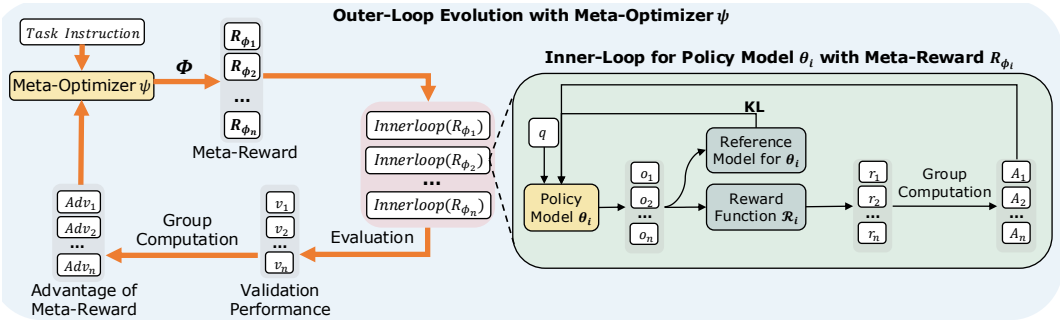

Figure 2: Bi-level evolutionary training framework for DERL. **Blue Block**: Evolution of Meta-Optimizer ψ with n generated Meta-Rewards R (*i.e., rollouts*). Taking a fixed task instruction as input, ψ updates the parameter Φ of R with the signal from validation performance v. **Green Block**: The inner-loop training for policy model θ_i with Meta-Reward R_{ϕ_i} by GRPO. We evaluate the validation performance for each θ_i as the reward of R_{ϕ_i}, making it a differentiable signal for ψ to evolve through reinforcement learning.

2.1 Bi-Level Evolutionary Training

Studies have shifted toward automated reward design to ease human labor. Some treat the reward as a configuration to be optimized via evolutionary search (Romera-Paredes et al., 2024), primarily relying on genetic algorithms by stochastic mutations (Jaderberg et al., 2017a; Chen et al., 2023) or heuristic optimization via prompted agents (Ma et al., 2024; 2025a). Functioning as zero-order optimizers, these methods are blind to the optimization landscape and fail to capture the *meta-gradient*, resulting in sample inefficiency comparable to grid search versus gradient descent.

Mitigating this requires reformulating the discrete search into a continuous, differentiable optimization process, thereby enabling gradient-guided evolution to capture the *meta-gradients* to guide searching. Inspired by Bello et al. (2017), who employs RL to discover optimization algorithms, we propose DERL, a bi-level evolutionary training framework (Figure 2). In DERL, the outer-loop (level) optimizes a Meta-Optimizer ψ via RL to generate a configuration ϕ (which parameterizes the reward function R_ϕ, detailed in Section 2.2), while the inner-loop (level) optimizes a policy model θ under a reward function R_ϕ. Specifically, the bi-level optimization problem is formulated as follows:

Inner-loop (Policy Model Evolution) Given a configuration ϕ from the outer-loop, the inner-loop policy θ is optimized to maximize the expected parameterized reward:

$$\mathcal{J}_\phi^{\text{inner}}(\theta) = \mathbb{E}_{x\in\mathcal{D},\tau\sim\pi_\theta(\cdot|x)}[R_\phi(\tau)], \tag{1}$$

where $\mathcal{D}$ represents the training dataset and τ denotes the trajectories sampled from the policy model.

Outer-loop (Meta-optimizer Evolution) The objective is to train the Meta-Optimizer to generate configurations that yield high-performing inner policies. We postulate that the data-driven policy gradient can automate the discovery of complex reward functions. Formally, the Meta-Optimizer ψ acts as a generator policy $\pi_\psi(\cdot|\texttt{ins})$, taking a fixed task instruction $\texttt{ins}$ as input and the configuration ϕ which instantiates the Meta-Reward R_ϕ as output (equation 1). Once the inner policy converges to an optimal θ^* under R_ϕ, it is evaluated against the performance score $\text{Perf}(\cdot)$ (*e.g.,* accuracy) on a validation set, which is adopted as the **outer feedback signal** (*i.e.,* the reward) for the Meta-Optimizer ψ. Consequently, the Meta-Optimizer aims to maximize the following bi-level objective:

$$\begin{aligned} \mathcal{J}^{\text{outer}}(\psi) &= \mathbb{E}_{\phi\sim\pi_\psi(\cdot|\texttt{ins})}[\text{Perf}(\theta^*)], \\ \text{s.t.} \quad \theta^* &= \arg\max_\theta \mathcal{J}_\phi^{\text{inner}}(\theta). \end{aligned} \tag{2}$$

This bi-level training framework transforms the *discrete evolutionary search* of reward function into a *continuous,* ***differentiable*** *optimization* over the meta-policy parameters ψ. Unlike genetic algorithms that rely on heuristic, stochastic mutations (blindly searching the space), DERL *learns the "search direction",* transforming the zero-order reward search into a first-order optimization of the generator.

2.2 INSTANTIATION OF DERL

Reward Parameterization A key challenge in Meta-Optimizer evolution lies in designing the structure of the generated configuration to parameterize the reward function for inner-loop. Traditional reward design typically adopts heuristic scalar functions, often sparse and necessitating manual effort to analyze validation failures (Shao et al., 2024). Alternatively, others employ trained reward models to address brittleness and scalability, which incurs a high annotation cost (Schulman et al., 2017). To ease this problem, we directly parameterize the reward function structure. Instead of predicting a single arbitrary reward function, which introduces a vast search space, our Meta-Optimizer generates a **configuration** ϕ—a symbolic formulation that specifies the *structure and weights* for **composing** multiple atomic primitives to evaluate the inner-loop agent. This design enables the automatic evolution of structural, non-linear evaluation criteria, thereby reducing human labor.

Specifically, we define the reward function as a symbolic composition of *atomic primitives* $\mathcal{G} = \{g_1, g_2, \ldots, g_k\}$—functions that evaluate specific aspects of a model's output o, to ensure a structured and expressive search space. Typically, a primitive requires the model output o and a context variable $\mathcal{C}$, containing information like the ground-truth a^* and query q. Example primitives include binary outcome correctness (*i.e.,* comparing o to a^*) adherence to formatting rules in $\mathcal{C}$, or process heuristics like step counts. Based on the primitives, the Meta-Optimizer ψ predicts the *structure and weights* (ϕ) that combine these signals through mathematical composition instead of generating a function directly. More details are in Appendix B. The reward function is therefore defined as:

$$R_\phi(o, \mathcal{C}) = \text{Func}(g_1(o, \mathcal{C}), \ldots, g_k(o, \mathcal{C}); \phi) \tag{3}$$

where $\text{Func}(\cdot)$ represents the symbolic execution of the configuration ϕ, which involves applying weights and mathematical operators (*e.g.,* summation, logical conditions) to the primitive outputs.

Inner-loop (Policy Model Evolution) We utilize Group Relative Policy Optimization (GRPO) Shao et al. (2024) for policy model optimization (the green block of Figure 2). The objective is defined as:

$$\mathop{\mathbb{E}}_{\substack{q\sim P(\mathcal{D}) \\ \{o_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(\cdot|q)}} \left[\frac{1}{G}\sum_{i=1}^{G} \min\left(\frac{\pi_\theta(o_i \mid q)}{\pi_{\theta_{\text{old}}}(o_i \mid q)} A_i,\ \text{clip}\left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{\text{old}}}(o_i|q)},\ 1-\epsilon,\ 1+\epsilon \right) A_i \right) - \beta\, D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}) \right], \tag{4}$$

where we sample a group of G outputs $\{o_i\}_{i=1}^{G}$ from the old policy $\pi_{\theta_{\text{old}}}$ for each question q from the training set $\mathcal{D}$. π_θ and $\pi_{\theta_{\text{old}}}$ denotes the current and previous policy, respectively. ϵ and β denotes hyper-parameters for the clipping range and KL-divergence penalty against a reference policy π_{ref}, with D_{KL} detailed in Shao et al. (2024). Crucially, A_i represents the group-wise advantage derived from our **parameterized reward function** within each group. For each output o_i in a group, generated *w.r.t.,* a given context $\mathcal{C}$, reward r_i is computed. The advantage is then defined as:

$$A_i = \frac{r_i - \text{mean}(\{r_j\}_{j=1}^{G})}{\text{std}(\{r_j\}_{j=1}^{G})}, \quad \text{where } r_i = R_\phi(o_i, \mathcal{C}). \tag{5}$$

To investigate the impact of model plasticity and optimization efficiency, we implement two distinct *initialization strategies* for inner-loop: **1) Standard Init** (denoted as **DERL**): In this strategy, for each inner-loop, the policy model θ is initialized from the base model. This ensures that the performance of θ is solely attributable to the efficacy of the current reward configuration ϕ, providing an unbiased evaluation signal to the Meta-Optimizer. **2) Population-based Variant** (denoted as **DERL-Pop.**): In the first inner-loop, we initialize the policy model from the base model. In later inner-loop, we start training the policy model from the model with the best validation performance in the last loop. This is similar to population-based evolutionary method where the policy model is evolved from different training configurations (Shao et al., 2025; Jaderberg et al., 2017b), but our Meta-Optimizer captures the meta-gradient in a differentiable and dynamic way. For fair comparison, we ensure that the total training step for DERL-pop.'s inner-loop policy remains the same as standard DERL, saving computation since the outer-loop can evolve more frequently.

Outer-loop (Meta-optimizer Evolution) We similarly employ GRPO to optimize the outer-loop (blue block in Figure 2). In each iteration, the Meta-Optimizer π_ψ samples a group of n reward configurations $\Phi = \{\phi_1, \phi_2, \ldots, \phi_n\}$, where n denotes the rollout size. Each configuration ϕ_i is used to instantiate a reward function R_{ϕ_i} to train a corresponding inner-loop policy θ_i. Upon completion

of the inner-loop training, we evaluate each policy θ_i on a held-out validation set $\mathcal{V}$ to compute the performance score $v_i = \text{Perf}(\theta_i)$ using the pass@1 accuracy metric:

$$\text{Perf}(\theta) = \frac{1}{|V|} \sum_{(q,a^*)\in\mathcal{V}} \mathbb{I}(f_\theta(q) = a^*), \tag{6}$$

where $f_\theta(q)$ denotes output by the trained policy π_θ with deterministic decoding, a^* denotes ground truth, and $\mathbb{I}(\cdot)$ denotes indicator function. The resulting validation scores $\{v_1, \ldots, v_n\}$ serve as feedback signals (outer rewards) for the corresponding configurations $\{\phi_1, \ldots, \phi_n\}$. We then compute the group-wise advantage (same as the inner-loop) to update the Meta-Optimizer parameters ψ. Crucially, DERL establishes a closed-loop computation that propagates non-differentiable validation signals back to the Meta-Optimizer via estimated meta-gradients (see Appendix F.1). This differentiable nature replaces manual heuristics with a direct optimization path, enabling the autonomous synthesis of dense feedback signals from sparse outcomes. Consequently, DERL effectively resolves scalability challenges in complex domains where manual reward engineering is prohibitively expensive.

3 EXPERIMENTS

We validate DERL across three diverse domains that necessitate complex reasoning, *i.e.,* Robotic Agents, Scientific Simulation and Mathematical Reasoning. We aim to address two research questions: **1)** Can DERL discover reward functions better than heuristics signals? **2)** Does the learned Meta-Reward generalize better to out-of-distribution (OOD) scenarios in complex reasoning tasks?

3.1 EXPERIMENT SETUPS

Robotic Agent We utilize a multi-round robotic agent task, *i.e.,* ALFWorld (Shridhar et al., 2020), requiring to complete embodied household tasks with natural language or visual observations. To rigorously assess the generalization capability, similar to Zhang et al. (2025b), we evaluate on three difficulty levels based on the distribution shift of training and testing data: **L0** (in-distribution, seen): trained on all 6 task types and evaluated on seen variants; **L1** (in-distribution, unseen): trained on all 6 task types but evaluated on unseen variants; **L2** (out-of-distribution), trained only on 4 task types and evaluated on the remaining 2 unseen types.

We compare DERL with standard RL baselines: **1)** GRPO + Out.: GRPO with binary outcome rewards. **2)** GRPO + Avg.: As we introduce the atomic primitives, a common practice is to calculate the weighted sum over all functions. To demonstrate DERL's exploration of an optimal reward structure over the search space, we compare with the average weighted sum. **3)** GiGPO (Feng et al., 2025) with a two-level structure for finer-grained credit assignment. **4)** RLVMR (Zhang et al., 2025b) with structured verifiable process reward, which is the previous state-of-the-art method. We do not compare with LLM-based reward models due to their high resource intensity, particularly since ground-truth outcome signals are readily available for the target domains.

Scientific Simulation We adopt ScienceWorld (Wang et al., 2022), an interactive text environment at the level of a standard elementary school science curriculum, testing the agents' scientific reasoning abilities. To ensure consistent evaluation of generalization, we evaluate on the three difficulty levels (*i.e.,* L0, L1, and L2) and compare against the same set of baselines as in the Robotic Agent tasks.

Mathematical Reasoning We adopt two established benchmarks: GSM8K (Cobbe et al., 2021) for grade-school math and MATH (Hendrycks et al., 2021) for advanced competition-level problems. We vary the training data by using either the MATH training set, which contains more difficult maths problems, or combining the MATH and GSM8k, which contains both easy and hard problems.

We benchmark against a set of baselines varying in reward structure: **1) Outcome**: a standard binary outcome-based reward; **2) Outcome + Format**: the outcome rewards augmented with format reward; and **3) Avg Reward**: the average reward over all individual atomic primitives.

3.2 IMPLEMENTATION DETAILS

Robotic Agent and Scientific Simulation We introduce four atomic primitives to construct the reward space. One is binary outcome reward. Others are captured from three stages of the interaction

Table 1: Performance of DERL on three distinct domains. **Bold** and underline denotes the best and second best performance.

(a) Success rates on ALFWorld and ScienceWorld (Qwen2.5-1.5B-Instruct) across three levels of generalization difficulty (Section 3.1). Out. and Avg. denotes outcome reward and average reward over all atomic primitives. GRPO baselines are run by ourselves. Other results are from Zhang et al. (2025b). Our DERL outperforms all baselines in all difficulty levels, achieving **state-of-the-art** performance.

Method	ALFWorld			ScienceWorld		
	L0	L1	L2	L0	L1	L2
GRPO + Out.	76.6	71.1	29.7	21.1	13.7	10.9
GRPO + Avg.	88.1	85.4	30.5	37.9	31.3	18.0
GiGPO	86.7	83.2	48.0	25.8	15.2	4.7
RLVMR	89.1	87.9	56.3	46.9	34.4	26.5
DERL	91.0	**89.1**	65.0	47.7	43.0	30.1
DERL-pop.	**91.8**	88.3	**76.4**	**98.2**	**95.3**	**31.3**

(b) Accuracy on GSM8k and MATH (`Qwen-2.5-3B`) with different training data and reward functions. Our DERL outperforms all baselines, including the outcome reward, outcome + format reward, and the average reward over atomic primitives. All results are run under the same configuration.

Reward	Train Data	GSM8k	MATH
Outcome	MATH+GSM8k	82.6	58.8
Out.+Format	MATH+GSM8k	86.4	55.9
Avg.	MATH+GSM8k	86.5	55.8
Outcome	MATH	82.9	59.1
Out.+Format	MATH	83.9	56.8
Avg.	MATH	83.6	54.9
DERL	MATH+GSM8k	87.0	60.2
DERL-pop.	MATH+GSM8k	**87.6**	60.2
DERL	MATH	83.2	60.5
DERL-pop.	MATH	84.1	**60.9**

trajectory, inspired by Zhang et al. (2025b). Specifically, we compute the average reward over the first, middle and last third of stages of the interaction trajectory, respectively. For instance, given a six-step interaction with a step-wise reward sequence of $[1, 0, 1, 1, 0, 0]$, the atomic primitives corresponding to the three temporal stages yield values of 0.5, 1, and 0, respectively. This straightforward design incentivizes the model to attend to distinct temporal phases of the task. More in Appendix C.

Mathematical Reasoning We construct the reward space with four straightforward atomic primitives: 1) Binary outcome reward; 2) Format reward which verifies if the answer is enclosed in "$boxed\{\}$"; 3) Step-by-step reward, identifying whether the output contains *CoT tokens e.g., "step 1"*; 4) Soft outcome reward, which credits the presence of ground truth anywhere in the output, helpful when the model indeed knows the answer but generates in a wrong format. Appendix C.

3.3 Results

Robotic Agent and Scientific Simulation Table 1a presents the performance comparison on Robotic Agent and Scientific Simulation benchmarks across three generalization levels. We observe two key findings: **1) State-of-the-Art (*sota*) Performance.** DERL achieves *sota* success rates across all difficulty levels on both benchmarks. This indicates that the Meta-Optimizer effectively explores the function space to discover Meta-Rewards that drive policy improvement beyond standard outcome signals. Notably, our population-based instantiation, DERL-pop., further demonstrates exceptional performance, achieving **91.8%** on ALFWorld (L0) and **98.2%** on Science World (L0). This shows that by initializing the inner-loop policy with the best-performing model from previous generations, the Meta-Optimizer can effectively adapt the reward signal dynamically as the policy model evolves, creating a curriculum-like effect that accelerates convergence and elevates the performance ceiling. It is worth noting that DERL-pop. consumes significantly less computations by optimizing the outer-loop more frequently. Appendix H details the training dynamics of DERL-pop. **2) Robustness in OOD Scenarios.** A critical limitation of heuristic rewards is the brittleness under distribution shifts. As shown in **L2 (OOD)** columns, standard baselines falter significantly. For instance, while *"GRPO + Avg."* improves in-distribution (L0) performance by approximately 10% over *"GRPO + Out."*, it fails to translate this gain to the OOD setting (showing only a 0.8% improvement). This suggests that the straightforward reward summation encourages overfitting rather than genuine reasoning. In contrast, our **DERL** substantially improves the OOD robustness, achieving **65.0%** and **30.1%** success rates on ALFWorld and ScienceWorld, respectively. This more than doubles the performance of the outcome reward baseline.

Take-away 1. *The Meta-Reward captures the intrinsic structure of the task, enabling generalization to unseen scenarios where heuristic combinations fail.*

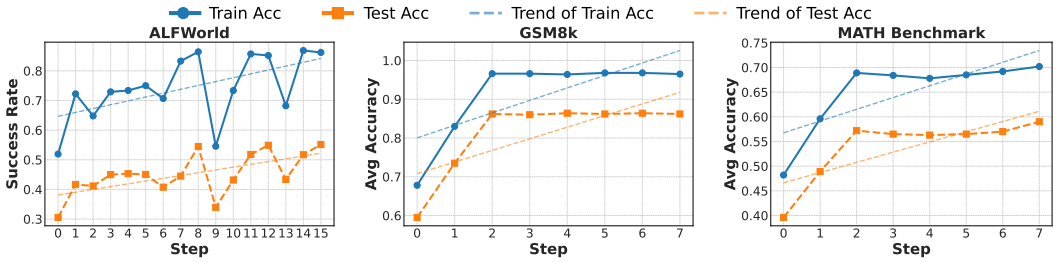

Figure 3: Training dynamics of the Meta-Optimizer on ALFWorld, GSM8k and MATH Benchmarks. The x-axis represents the training steps of outer-loop. The blue and orange line denotes the average validation and testing performance over Meta-Reward (*i.e., "rollouts"*), respectively. The results show that DERL gradually converges as the training progresses, without overfitting.

Mathematical Reasoning presents a unique challenge where the outcome reward is already strong, but sparse for hard question exploration. Table 1b illustrates the performance on GSM8K and MATH. We observe that naively incorporating auxiliary signals (*e.g., GRPO with Outcome + Format* or *Avg Reward*) often degrades performance on the more difficult MATH dataset (dropping from 58.8% to 55.8%), likely due to "reward hacking" or distraction from the core reasoning path (*e.g.,* prioritizing formatting over correct reasoning). However, our DERL successfully navigates this pitfall. By autonomously optimizing the reward structure without any human effort, DERL outperforms all baseline reward functions, including the strong outcome reward (*e.g.,* 60.2% vs. 58.8% on MATH), with the population-based instantiation DERL-pop. further improving the performance. This demonstrates DERL 's ability to navigate the delicate trade-off between signal density and signal fidelity.

Take-away 2. *Even in domains with strong ground-truth signals,* DERL *discovers non-trivial reward compositions that provide denser feedback without introducing the noise associated with manual heuristic design.*

4 Analysis

Having demonstrated the empirical superiority of DERL, we now investigate the internal mechanisms driving these improvements. We focus on two key aspects: the optimization dynamics of the outer-loop and the structural evolution of the generated reward functions.

4.1 Optimization Dynamics

A critical question is **whether the Meta-Optimizer genuinely learns a progressive optimization strategy or merely performs a random search over the function space**. To investigate this, we visualize the training dynamics (*i.e.,* how the Meta-Optimizer evolves over training steps) on ALFWorld, GSM8K and MATH benchmarks in Figure 3. Results on ScienceWorld is not shown because the Meta-Optimizer converges faster. Specifically, we demonstrate the average validation accuracy of inner-loop policies $\theta_1, \theta_2, ..., \theta_n$ trained with n Meta-Reward.

We observe a **consistent, monotonic upward trend** in average performance of both validation and testing as the outer-loop progresses. Specifically, we find that the trend in mathematical reasoning is more stable as the Meta-Optimizer recognizes that the verifiable task is mainly driven by the outcome reward. For agent tasks, outer-loop optimization involves more exploration than exploitation, ultimately showcasing robust evolution. Crucially, the concurrent rise in validation and testing performance provides empirical verification that the Meta-Optimizer is **not overfitting** to the specific instances in outer-loop training. Instead, it successfully approximates the "meta-gradient" of task success. By leveraging the validation performance as a supervisory signal, the Meta-Optimizer gradually refines the reward function, generating increasingly effective signals that guide the inner-loop agent toward higher performance.

Take-away 3. *The Meta-Optimizer drives a* ***gradient-guided evolution*** *rather than a stochastic random search, validating that* DERL *captures the intrinsic meta-gradient of the task, enabling the refinement of generalizable reward structures without overfitting to the outer-loop training instances.*

4.2 Evolution Dynamics of Reward Structures

To elucidate specific characteristics of the learned rewards, we analyze the structural composition of the Meta-Reward generated throughout the evolutionary process, *i.e.,* the evolution dynamics. We categorize the combinations of atomic primitives (denoted as g_1, g_2, g_3, and g_4) into three distinct structure types based on mathematical properties: **1) Stable Structure**, which adopts linear combinations or normalization mechanisms. For example, linear additions (*e.g.,* $0.5 \cdot g_1 + 0.8 \cdot g_2$) or division operations (*e.g.,* $\frac{g_1}{g_2+1}$) that act similarly to sigmoid functions bound the output range. This structure mirrors robust designs in deep learning, preventing numerical explosion while retaining sufficient expressivity to guide the agent. **2) Unstable Structure**, in contrast, predominantly features unbounded products without normalization. A typical example is a chain of sequential multiplications (*e.g.* $g_1 \cdot (g_2 + 0.2) \cdot g_3$). This structure creates a severe "veto" mechanism: if any single atomic signal approaches zero, the entire reward vanishes, leading to high variance and unstable gradient update. **3)** *Invalid Structure* refers to mathematically adversarial forms, such as assigning negative coefficients to positive signals (*e.g.,* $-(g_1 + 0.5 \cdot g_2)$), which penalize desirable behaviors and offer no optimization utility.

Figure 4 tracks the distribution of these structural types over the course of training on ALFWorld. The optimization trajectory reveals a distinct *gradient-guided refinement* process. In the early exploration phase, Unstable Structures appear frequently as the optimizer explores the search space. However, as training progresses, we observe a sharp decline in its prevalence. Simultaneously, the proportion of Stable Structures exhibits a strong upward trend, eventually becoming dominant. This dynamic suggests that the Meta-Optimizer effectively acts as an evolutionary filter for mathematical robustness. Without explicit human programming or constraints, our DERL implicitly discovers that consistent, bounded, and numerically stable rewards are critical prerequisites for effective policy optimization.

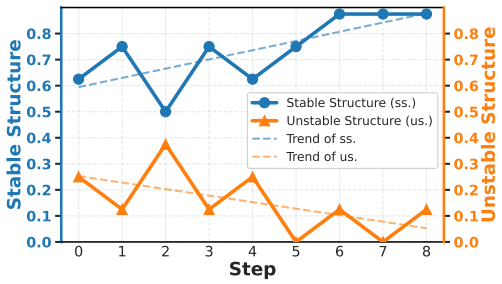

Figure 4: Evolution dynamics of reward structures on ALFWorld. We visualize the proportion of Stable and Unstable Structures over outer-loop steps. The consistent upward trend of stable structures validates the Meta-Optimizer's selection preference for mathematical robustness.

Take-away 4. *The Meta-Optimizer implicitly learns to prioritize numerical stability and boundedness. This demonstrates that* DERL *discovers essential reward design principles solely through gradient feedback, without requiring manual constraints.*

5 Conclusions

We introduced Differentiable Evolutionary Reinforcement Learning (DERL), a bi-level evolutionary training framework that automates the discovery of reward functions. By parameterizing the reward structure as a composition of atomic primitives and taking validation performance of the inner-loop policy as a supervisory signal, DERL bridges the gap between black-box evolutionary heuristics and gradient-based optimization. Our Meta-Optimizer manages to capture the "meta-gradient" of task success, allowing it to progressively learn dense, actionable feedback signals without reliance on expensive human effort.

Empirical results demonstrate that DERL consistently outperforms standard reinforcement learning baselines and human-designed heuristics in three diverse domain. Notably, DERL exhibits superior generalization capabilities in OOD scenarios, achieving state-of-the-art performance on ALFWorld and ScienceWorld benchmarks. Further analysis of the evolution dynamics reveals that the Meta-Optimizer naturally converges toward numerically stable and robust reward structures, effectively filtering out volatile signal combinations. This confirms that DERL is not merely a search algorithm, but a mechanism for discovering the intrinsic structural logic required for effective agent training.

REFERENCES

Rishabh Agarwal, Chen Liang, Dale Schuurmans, and Mohammad Norouzi. Learning to generalize from sparse and underspecified rewards. In *International conference on machine learning*, pp. 130–140. PMLR, 2019.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W. Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent, 2016. URL https://arxiv.org/abs/1606.04474.

Anonymous. Temperature as a meta-policy: Adaptive temperature in LLM reinforcement learning. In *Submitted to The Fourteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=AoTHU2OmS6. under review.

Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V. Le. Neural optimizer search with reinforcement learning. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pp. 459–468. PMLR, 2017. URL http://proceedings.mlr.press/v70/bello17a.html.

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, et al. Symbolic discovery of optimization algorithms. *Advances in neural information processing systems*, 36:49205–49233, 2023.

Xingwu Chen, Tianle Li, and Difan Zou. Reshaping reasoning in llms: A theoretical analysis of rl training dynamics through pattern selection, 2025. URL https://arxiv.org/abs/2506.04695.

Sitao Cheng, Liangming Pan, Xunjian Yin, Xinyi Wang, and William Yang Wang. Understanding the interplay between parametric and contextual knowledge for large language models. *arXiv preprint arXiv:2410.08414*, 2024a.

Sitao Cheng, Ziyuan Zhuang, Yong Xu, Fangkai Yang, Chaoyun Zhang, Xiaoting Qin, Xiang Huang, Ling Chen, Qingwei Lin, Dongmei Zhang, Saravan Rajmohan, and Qi Zhang. Call me when necessary: LLMs can efficiently and faithfully reason over structured environments. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 4275–4295, Bangkok, Thailand, August 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.254. URL https://aclanthology.org/2024.findings-acl.254/.

Sitao Cheng, Xunjian Yin, Ruiwen Zhou, Yuxuan Li, Xinyi Wang, Liangming Pan, William Yang Wang, and Victor Zhong. From atomic to composite: Reinforcement learning enables generalization in complementary reasoning. *arXiv preprint arXiv:2512.01970*, 2025.

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

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang,

Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Jinyuan Fang, Yanwen Peng, Xi Zhang, Yingxu Wang, Xinhao Yi, Guibin Zhang, Yi Xu, Bin Wu, Siwei Liu, Zihao Li, et al. A comprehensive survey of self-evolving ai agents: A new paradigm bridging foundation models and lifelong agentic systems. *arXiv preprint arXiv:2508.07407*, 2025.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. *arXiv preprint arXiv:2505.10978*, 2025.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, et al. A survey of self-evolving agents: On path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*, 2025.

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

Jian Hu, Jason Klein Liu, and Wei Shen. Reinforce++: An efficient rlhf algorithm with robustness to both prompt and reward models. *arXiv preprint arXiv:2501.03262*, 2025.

Xiang Huang, Sitao Cheng, Yuheng Bao, Shanshan Huang, and Yuzhong Qu. Markqa: A large scale kbqa dataset with numerical reasoning. *arXiv preprint arXiv:2310.15517*, 2023.

Xiang Huang, Sitao Cheng, Shanshan Huang, Jiayu Shen, Yong Xu, Chaoyun Zhang, and Yuzhong Qu. QueryAgent: A reliable and efficient reasoning framework with environmental feedback based self-correction. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5014–5035, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.274. URL https://aclanthology.org/2024.acl-long.274/.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population based training of neural networks, 2017a. URL https://arxiv.org/abs/1711.09846.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017b.

W Bradley Knox and Peter Stone. Interactively shaping agents via human reinforcement: The tamer framework. In *Proceedings of the fifth international conference on Knowledge capture*, pp. 9–16, 2009.

Michel Ma, Takuma Seno, Kaushik Subramanian, Peter R. Wurman, Peter Stone, and Craig Sherstan. Automated reward design for gran turismo, 2025a. URL https://arxiv.org/abs/2511.02094.

Xueguang Ma, Qian Liu, Dongfu Jiang, Ge Zhang, Zejun Ma, and Wenhu Chen. General-reasoner: Advancing llm reasoning across all domains. *arXiv preprint arXiv:2505.14652*, 2025b.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=IEduRUO55F.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Junhyuk Oh, Matteo Hessel, Wojciech M Czarnecki, Zhongwen Xu, Hado P van Hasselt, Satinder Singh, and David Silver. Discovering reinforcement learning algorithms. *Advances in Neural Information Processing Systems*, 33:1060–1070, 2020.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nat.*, 625(7995):468–475, 2024. doi: 10.1038/S41586-023-06924-6. URL https://doi.org/10.1038/s41586-023-06924-6.

Bidipta Sarkar, Mattie Fellows, Juan Agustin Duque, Alistair Letcher, Antonio León Villares, Anya Sims, Dylan Cope, Jarek Liesen, Lukas Seier, Theo Wolf, et al. Evolution strategies at the hyperscale. *arXiv preprint arXiv:2511.16652*, 2025.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Rulin Shao, Akari Asai, Shannon Zejiang Shen, Hamish Ivison, Varsha Kishore, Jingming Zhuo, Xinran Zhao, Molly Park, Samuel G Finlayson, David Sontag, et al. Dr tulu: Reinforcement learning with evolving rubrics for deep research. *arXiv preprint arXiv:2511.19399*, 2025.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.

Richard Sutton. The bitter lesson. *Incomplete Ideas (blog)*, 13(1):38, 2019.

Zhengyang Tang, Zihan Ye, Chenyu Huang, Xuhan Huang, Chengpeng Li, Sihang Li, Guanhua Chen, Ming Yan, Zizhuo Wang, Hongyuan Zha, Dayiheng Liu, and Benyou Wang. Calm before the storm: Unlocking native reasoning for optimization modeling, 2025. URL https://arxiv.org/abs/2510.04204.

Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artificial intelligence review*, 18(2):77–95, 2002.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader? *arXiv preprint arXiv:2203.07540*, 2022.

Yiping Wang, Qing Yang, Zhiyuan Zeng, Liliang Ren, Liyuan Liu, Baolin Peng, Hao Cheng, Xuehai He, Kuan Wang, Jianfeng Gao, Weizhu Chen, Shuohang Wang, Simon Shaolei Du, and Yelong Shen. Reinforcement learning for reasoning in large language models with one training example, 2025. URL https://arxiv.org/abs/2504.20571.

Zhepei Wei, Xiao Yang, Kai Sun, Jiaqi Wang, Rulin Shao, Sean Chen, Mohammad Kachuee, Teja Gollapudi, Tony Liao, Nicolas Scheffer, et al. Truthrl: Incentivizing truthful llms via reinforcement learning. *arXiv preprint arXiv:2509.25760*, 2025.

Zhongwen Xu, Hado P van Hasselt, and David Silver. Meta-gradient reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

Zhongwen Xu, Hado P van Hasselt, Matteo Hessel, Junhyuk Oh, Satinder Singh, and David Silver. Meta-gradient reinforcement learning with an objective discovered online. *Advances in Neural Information Processing Systems*, 33:15254–15264, 2020.

Chuanhao Yan, Fengdi Che, Xuhan Huang, Xu Xu, Xin Li, Yizhi Li, Xingwei Qu, Jingzhe Shi, Chenghua Lin, Yaodong Yang, Binhang Yuan, Hang Zhao, Yu Qiao, Bowen Zhou, and Jie Fu. Re:form – reducing human priors in scalable formal software verification with rl in llms: A preliminary study on dafny, 2025. URL https://arxiv.org/abs/2507.16331.

Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. G\" odel agent: A self-referential agent framework for recursive self-improvement. *arXiv preprint arXiv:2410.04444*, 2024.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin godel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*, 2025a.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. *arXiv preprint arXiv:2408.15240*, 2024.

Zijing Zhang, Ziyang Chen, Mingxiao Li, Zhaopeng Tu, and Xiaolong Li. Rlvmr: Reinforcement learning with verifiable meta-reasoning rewards for robust long-horizon agents. *arXiv preprint arXiv:2507.22844*, 2025b.

A Related Work

Agentic Evolution of LLMs The capabilities of LLMs have recently shifted towards agent systems that encompass complex planning, tool usage, and self-correction mechanisms (Qin et al., 2023; Cheng et al., 2024b). However, the deployment of LLMs as agents is critically bottlenecked by their reliance on human-engineered configurations (*e.g., prompts, workflow, codes, reward functions, etc*), which are non-scalable and brittle (Sutton, 2019; Sarkar et al., 2025). While evolutionary algorithms attempt to optimize agent configurations, they operate in a discrete, black-box manner by permutation

(Jaderberg et al., 2017b; Chen et al., 2025; Fang et al., 2025) or a prompted agent (Yin et al., 2024; Novikov et al., 2025; Zhang et al., 2025a; Shao et al., 2025), relying only on sparse final fitness scores and failing to exploit the rich information embedded in the training dynamics (Gao et al., 2025). Our DERL introduces a parameterized Meta-Optimizer that enables a gradient-guided search for optimal configurations. Leveraging the validation performance of the optimizee as a reward signal, DERL validates that the meta-gradient can be captured by the evolutionary process, moving beyond human priors toward a scalable, computation-driven optimization mechanism.

Learning to Learn Meta-learning focuses on developing models or algorithms that can rapidly adapt to new tasks by leveraging experience gained from other related tasks (Vilalta & Drissi, 2002; Xu et al., 2018). It essentially automates the traditional manual processes, *e.g.,* hyper-parameter optimization, the selection of appropriate learning algorithms (Andrychowicz et al., 2016). Recent work has extended this concept to RL, where a meta-learner is designed to optimize the inner-loop learning process of an RL agent (Bello et al., 2017; Agarwal et al., 2019; Xu et al., 2020; Oh et al., 2020; Anonymous, 2025). However, as reward modeling is demanding, whether the optimization of reward signal can be learned by a meta-model is understudied. Our DERL is the first to formalize the automated reward search as a bi-level meta-optimization problem with LLMs, leveraging principles from meta-learning to optimize a meaningful reward function.

Reward Modeling The success of LLM alignment hinges on the reward function, providing feedback for agent evolution through RL (Schulman et al., 2017). This is complicated by the dilemma between sparse, objective outcome reward (Shao et al., 2024; Tang et al., 2025) and dense, but expensive, human-annotated reward (such as those used in RLHF) (Wang et al., 2025; Ouyang et al., 2022). Recent studies seek open-ended reward by training a model on large-scale LLM-annotated web-crawled data (Ma et al., 2025b; Zhang et al., 2024; Ma et al., 2024). Others design heuristic rewards which requires complex manual coordination and may even degrade performance if naively combined (Zhang et al., 2025b; Wei et al., 2025; Yu et al., 2025; Yan et al., 2025). To address these challenges, our DERL employs a Meta-Optimizer to automatically generate reward functions without relying on external human preference data.

B Details on Structured Meta-Reward

Merits of Structured Meta-Reward The merits of our structured Meta-Reward design (*i.e.,* generating a configuration composing atomic primitives instead of an arbitrary function) are three-fold: **1)** Coverage of a informative and continuous search space. Equation equation 3 takes abundant factors into account, ensuring an expressive space covering the standard outcome signal. **2)** Structural reasoning. The Meta-Optimizer can focus on considering different aspects of the problem instead of processing tedious textual output. **3)** Extensibility, decoupling the definition of atomic signals from their utilization. This renders the system agnostic to the specific choice of $\mathcal{G}$, thereby facilitating seamless generalization to new tasks. One can incorporate diverse potential discriminators—ranging from rigorous constraints to task-specific heuristics, or even potentially detrimental signals. The evolutionary process automatically filters and weights these components, circumventing the need for manual validation of their individual utility.

Implementation To ensure the stability of the symbolic generation, we adopt three strategies: **1) Cold Start:** We apply Supervised Fine-Tuning to the Meta-Optimizer on a small set of valid format examples to initialize the policy. **2) Constrained Decoding:** We enforce token-level constraints to ensure the validity of tokens in the generated output ϕ. **3) Validity Penalty:** In rare cases where a generated ϕ_i is mathematically ill-defined (*e.g.,* resulting in execution errors), we assign a penalty reward $v_i = 0$ to suppress such generations.

C Implementation Details

Robotic Agent and Scientific Simulation We implement DERL with GRPO via VeRL (Sheng et al., 2024). For outer-loop, we utilize `Qwen-2.5-0.5B-Instruct` as the Meta-Optimizer and set the number of *rollouts* to 8. Other hyper-parameters remain defaulted. For inner-loop, we employ `Qwen2.5-1.5B-Instruct` as the base policy with a cold start (same as other baselines). We set epoch to 40 for ALFWorld and 80 for ScienceWorld, respectively. After obtaining the optimal

Meta-Reward, we train the policy model (from scratch) using this reward function for 100 steps, the same as RLVMR, whereas other baselines are trained for 150 steps. The Meta-Optimizer achieves convergence in approximately ten and five outer-loop iterations for ALFWorld and ScienceWorld, respectively. For DERL-pop., we train the outer-loop for 10 rounds and set the training epochs for inner-loops to 10 on ALFWorld. On ScienceWorld, we train the outer-loop for 3 rounds and the inner-loop for 33 rounds, to ensure the consistency of the total training epochs for the inner-loop. We report the success rate on the test set, *i.e.,* whether the model can ultimately complete the task.

Mathematical Reasoning For outer-loop, we keep all configurations the same as other tasks. For inner-loop, we adopt `Qwen-2.5-3B` as the base policy model. We train for 10 epochs and enforce a time limit of 3.5 hours for inner-loop training. With these settings, the Meta-Optimizer achieves convergence in approximately 8 outer-loop iterations. We then take the optimal Meta-Reward to train the base policy for 15 epochs for fair comparison with baselines. For DERL-pop., we train each inner-loop for 2 epochs and report the testing result after 7th outer-loop iteration for fair comparison. We evaluate the exact match of the ground truth answer in the test set.

D Limitations and Future Work

While DERL demonstrates significant promise, several limitations remain that outline important directions for future research.

Computational Cost. The primary bottleneck of our framework is the computational expense associated with the bi-level optimization structure. Since every update to the Meta-Optimizer requires the training of multiple inner-loop policies ("rollouts"), the process is resource-intensive compared to standard single-level RL. A detailed cost analysis is conducted in Appendix G. We already provided DERL-pop. with higher efficiency and better performance. Future work could explore the integration of lightweight proxy tasks or more sample-efficient outer-loop algorithms (*e.g.,* REINFORCE++ (Hu et al., 2025)) to approximate the meta-gradient with reduced compute.

Dependency on Atomic Primitives. The expressivity of the discovered reward functions is currently bounded by the set of atomic primitives defined in the search space. While our selection of primitives (*e.g.,* format checks, partial goal verifiers) are proved effective for the studied domains, the Meta-Optimizer cannot invent entirely new functional capabilities outside of this pre-defined grammar. Expanding the search space to include more granular or semantically rich primitives—potentially extracted automatically from task descriptions—remains an open challenge.

Long-Horizon Credit Assignment. Although DERL improves upon sparse outcome rewards, the generated Meta-Rewards are still evaluated based on the final validation performance of the policy. In tasks with extremely long horizons or deceptive intermediate goals, the signal propagation from the final metric back to the specific reward parameters may still suffer from attenuation. Investigating intermediate meta-supervision signals could further enhance the stability and efficiency of the evolutionary process.

E Preliminary Experiments

In this section, we present a preliminary experiment designed to investigate the capacity of our proposed Meta-Reward mechanism to simulate and potentially enhance standard outcome-based rewards. Specifically, we utilize a computational graph parameterized by a small set of weights to derive a reward function, which is then utilized to train the inner-loop model.

E.1 Experimental Setup

Architecture. As illustrated in Figure 5, the Meta-Optimizer is implemented via a graph neural network representing general computational graphs. The Meta-Reward function, parameterized by ϕ_t, is represented by a set of twelve learnable weights (*e.g.,* $w_{add}, w_{sub}, w_{mul}, \dots$) distributed across the computational nodes. The set of atomic primitives used in this graph remains consistent with those described in the main body of this paper (Section 2).

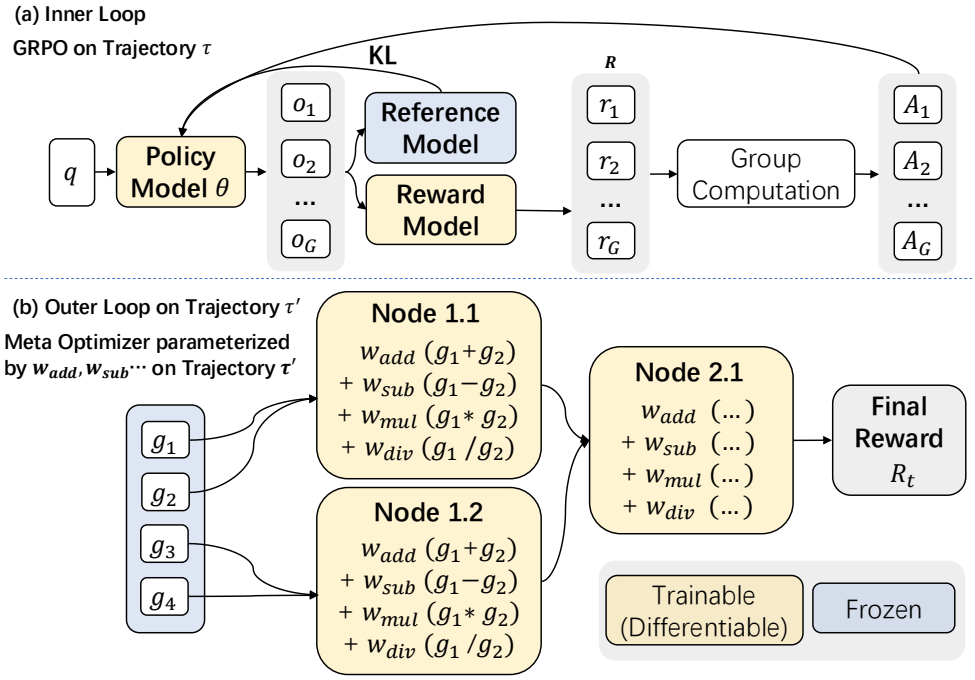

Figure 5: Demonstration of the training loop our differentiable evolutionary reward. We adopt GRPO as an example RL algorithm for the inner-loop. In the outer-loop, we leverage a graph neural network to represent general computational graphs and obtain the final reward function Φ_t parameterized by $\phi_t = \{w_{add}, w_{sub}, ...\}$ through differentiable optimization.

Training Protocols. We explore two distinct strategies to evolve the Meta-Optimizer:

- **Supervised Fine-Tuning (SFT):** In this setting, the Meta-Optimizer is trained to directly regress the ground truth outcome reward. The objective is to minimize the divergence between the generated Meta-Reward and the standard outcome signal (0.0 or 1.0).
- **Reinforcement Learning (RL):** We formulate the optimization of weights w as an RL problem. For each computation step, we sample operations based on the distribution of w within each node. If the resulting Meta-Reward aligns with the ground truth outcome reward (*i.e.,* $\mathbb{I}(\text{Meta-Reward} = \text{Outcome})$), a reward of 1.0 is assigned to the current configuration of w; otherwise, the reward is 0.0.

Model and Data. To generate a diverse set of trajectories for training, we perform inference on the training sets of the GSM8K and MATH benchmarks using `Qwen2.5-3B-Instruct`. These trajectories serve as the basis for optimizing the Meta-Optimizer. Once Meta-Reward is learned, it is frozen and used to train the inner-loop policy model (`Qwen2.5-3B`). The inner-loop training settings are identical to the baseline configuration to ensure a fair comparison.

Table 2: Performance comparison on mathematical reasoning benchmarks. We compare the preliminary results of Meta-Reward with different baselines. The Meta-Optimizer utilize a 12-parameter graph optimizer to shape the reward signal

Reward Function	GSM8K	MATH
Outcome	82.6	58.8
Avg Reward	**86.5**	55.8
Meta-Reward (SFT)	85.5	**62.9**
Meta-Reward (RL)	83.2	59.9

E.2 Results and Analysis

Table 2 presents the performance comparison between the standard outcome reward, an average reward baseline (same as the main experiments in Section 3), and our proposed Meta-Reward (SFT and RL). The results indicate that the Meta-Reward, despite being parameterized by only 12 weights, effectively discovers a reward function that outperforms the sparse outcome reward. Notably, on the challenging MATH dataset, the Meta-Reward (SFT) achieves a significant improvement over the Outcome baseline (62.9% vs. 58.8%).

These findings suggest that the Meta-Reward mechanism avoids overfitting to the rigid binary outcome signal. Analogous to an educational setting, using a binary outcome reward is akin to instructing a student solely to "score 100 points," which provides a sparse and high-variance signal. In contrast, our approach—constrained by the computational graph structure—encourages the model to learn a generalized heuristic. Although this is not an explicit process reward annotated by humans, the optimization process discovers an implicit, dense reward function that guides the model more effectively toward the correct reasoning path than the raw outcome signal alone.

F Gradient Propagation on DERL

In this section, we elaborate on the information flow between the meta-optimizer (parameterized by ψ) and the inner-loop policy model (the optimizee, parameterized by θ). A distinct feature of our DERL framework is the preservation and utilization of meta-gradient information, which allows the optimizer to explicitly learn from the validation performance of the optimizee.

F.1 Gradient Propagation Flow

The interaction between the optimizer and the optimizee unfolds as a bi-level optimization process, as illustrated in Figure 6. Let v_t denote the validation performance (or evaluation metric) of the policy model θ_t at step t. The optimization process consists of two coupled loops:

- **Inner Loop (Optimizee):** The policy model updates its parameters $\theta_{t-1} \to \theta_t$ based on the guidance of the Meta-Reward $\mathcal{R}_{\phi_t}$ provided by the optimizer. The optimizer then generates the update instructions (parameterized by ϕ_t) conditioned on its current state ψ_t.
- **Outer Loop (Optimizer):** The meta-optimizer evolves $\psi_{t-1} \to \psi_t$ by maximizing the expected future validation performance of the optimizee.

Crucially, the update of the optimizer ψ is driven by the gradient of the validation performance, denoted as $\nabla \mathcal{J}^{\text{outer}}(\psi_t)$. This term represents the *meta-gradient*: it quantifies the sensitivity of the optimizee's performance with respect to the optimizer's parameters. By backpropagating the signal from the validation performance v through the update step to ψ, our DERL establishes a direct feedback loop.

F.2 Meta-Gradient Propagation Compared with Previous Evolution

The core innovation of our framework lies in the end-to-end differentiability of the optimization trajectory, or how the feedback signal v_t is utilized. In traditional reinforcement learning or prompt-based optimization methods, the optimizer ψ is often treated as a static entity (*e.g.,* a fixed prompted agent or a random perturbation generator). In such cases, the dependency chain is broken, and the "meta-gradient"—the gradient of the validation performance with respect to the optimizer's parameters—is lost.

In contrast, our approach treats ψ as a learnable entity. We explicitly compute the gradient flow from the evaluation metric back to the optimizer parameters. As depicted in the bottom flow of Figure 6, the optimizer updates its own parameters ψ to maximize the expected future validation performance of the optimizee:

$$\psi_t = \psi_{t-1} + \eta \cdot \nabla \mathcal{J}^{\text{outer}}(\psi_{t-1}) \tag{7}$$

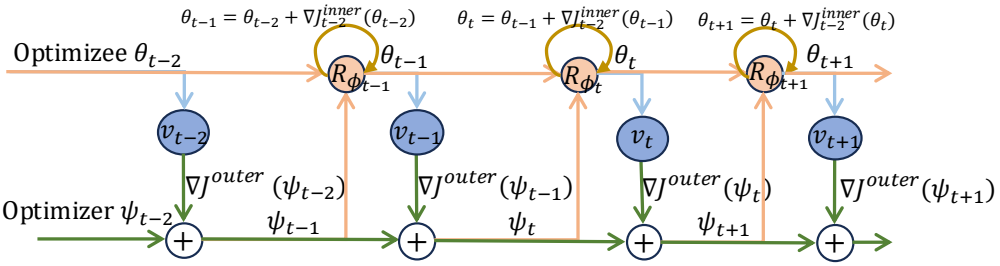

Figure 6: Illustration of Gradient Propagation in bi-level evolutionary training loop. The top row (Orange) represents the trajectory of the optimizee θ, updated via instructions ϕ derived from the optimizer. The bottom row (Green) represents the evolution of the Meta-Optimizer ψ. The blue nodes v_t denote the validation performance evaluation. Unlike static methods, our framework computes the meta-gradient $\nabla \mathcal{J}^{\text{outer}}(\psi_t)$ (vertical arrows), allowing the optimizer to update its own parameters ψ to explicitly maximize the optimizee's performance.

This derivation highlights that updating ψ is fundamentally learning the *meta-gradient*. Unlike prior works where the optimizer is fixed (resulting in $\frac{\partial \phi}{\partial \psi} = 0$ or undefined), our DERL framework maintains a differentiable (or estimable) path. This enables the Meta-Optimizer to iteratively improve the reward structure ϕ driven by direct performance feedback. In doing so, DERL serves as a foundational proof-of-concept for completely autonomous, self-improving frameworks.

G Computational Cost Analysis

In this section, we provide a detailed breakdown of the computational costs associated with the bi-level evolutionary training framework (DERL) and discuss potential strategies for efficiency improvements.

G.1 Computational Breakdown

The training process consists of an outer-loop (Meta-Optimizer evolution) and an inner-loop (Policy Model evolution). We incorporate parallelism in most parts of DERL to improve GPU untilization. The computational cost is as summarized as follows:

The training process consists of an outer-loop (Meta-Optimizer evolution) and an inner-loop (Policy Model evolution). We incorporate parallelism in most parts of DERL to improve hardware utilization. The computational cost is summarized as follows:

Inner-Loop Latency (The Bottleneck). The primary computational bottleneck lies in the inner-loop, where the policy model θ_i evolves (*e.g.,* interacts with the environment) using the Meta-Reward $\mathcal{R}_i$. Since our outer-loop utilizes the GRPO algorithm (Shao et al., 2024), the Meta-Optimizer generates n distinct Meta-Rewards (as *rollouts*) per step. To mitigate latency, we implement a fully parallelized architecture similar to standard GRPO:

- **Parallel Execution:** All n rollouts (where $n = 8$ in our experiments) are evaluated simultaneously, with each inner-loop allocated a dedicated set of compute resources. Each inner-loop for each task finally consumes similar computation resources.
- **Malformed Rewards:** Meta-Rewards that fail to compile or produce valid computation graphs are immediately terminated, assigned a reward of 0.0, and incur zero training cost (though the system waits for concurrent inner-loops to complete before updating the outer-loop).
- **Time Budgeting:** A significant challenge in Meta-Reward discovery is that certain reward functions may incentivize excessively long reasoning chains, increasing training costs unpredictably. To address this, we impose a strict computational budget. For mathematical reasoning tasks, we set a maximum floating-point operation cap approximately $1.3\times$ the cost of standard binary-outcome training. If training exceeds this threshold, the process is

halted, and the latest checkpoint is saved for evaluation. For other tasks, we rely on a fixed number of inner epochs, as the action space is more controllable.

Evaluation Cost. Following the inner-loop, we evaluate the validation performance to calculate the advantage for the Meta-Optimizer. We utilize `vLLM` for high-throughput inference. By parallelizing the evaluation across all n rollouts, the validation phase incurs negligible latency compared to training.

Outer-Loop Update. The Meta-Optimizer utilizes a lightweight 0.5B parameter model. Updating this model using the collected rollout data ($n = 8$) is computationally negligible, taking only minutes to complete.

Based on the parallelization strategy described above, the wall-clock time for one complete outer-loop iteration is determined by the slowest successful inner-loop trial plus evaluation and update overhead.

$$T_{total} \approx \max(T_{inner}) + T_{eval} + T_{update}$$

G.2 Resource Estimation

To contextualize the resource requirements of DERL, we compare its cost against the baseline of training a single inner-loop policy model. Let C_{inner} denote the computational cost required to train one standard inner-loop.

The total computational cost for standard DERL, which runs for E_{outer} outer-loop epochs with n parallel rollouts per step, can be estimated as:

$$C_{\text{DERL}} \approx n \times E_{\text{outer}} \times C_{\text{inner}}$$

This cost scales linearly with the number of outer-loop iterations required for the Meta-Optimizer to converge. In contrast, for DERL-pop, we simplify the process selecting the best reward function from a single population generation. In this setting, the total cost is significantly reduced to:

$$C_{\text{DERL-pop}} \approx n \times C_{\text{inner}}$$

Consequently, DERL-pop offers a more efficient alternative, consuming way less wall-clock time while still benefiting from population-based exploration. We utilized high-memory data center accelerators to accommodate the memory requirements of the parallel inner-loop training.

G.3 Efficiency Improvements and Future Work

In our current implementation, we utilized a relatively large number of rollouts ($n = 8$) and a full inner-loop training protocol to empirically verify that LLMs can effectively learn meta-gradients through Reinforcement Learning. However, our preliminary experiments (demonstrated in Section E) suggest that simple parameterizations (*e.g.,* 12 parameters) can also yield competitive results.

This observation points toward a promising direction for future work: reducing the heavy computational burden of the inner-loop by adopting lightweight RL algorithms, such as *REINFORCE++*. By simplifying the inner-loop requirements or using proxy tasks, the reliance on massive parallel resources can be significantly reduced, making the evolution of Meta-Rewards accessible to a broader range of computational budgets.

Evaluation Cost. Following the inner-loop, we evaluate the validation performance to calculate the advantage for the Meta-Optimizer. We utilize `vLLM` for high-throughput inference. By parallelizing the evaluation across all n rollouts, the validation phase typically requires only a few minutes.

Outer-Loop Update. The Meta-Optimizer itself utilizes a lightweight 0.5B parameter model. Updating this model using the collected rollout data ($n = 8$) is computationally negligible, taking only minutes to complete.

H Training Dynamics of DERL-pop.

As shown in the main body of our paper, DERL-pop. achieves significantly better performance in all tested domains, demonstrating that the Meta-Optimizer is able to dynamically explore a better reward function. Specifically, the reinforcement learning with the explored Meta-Reward generalizes to

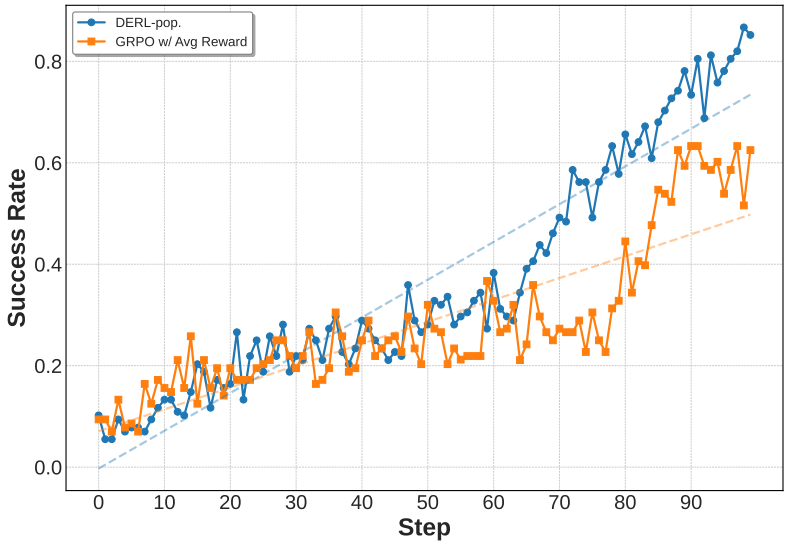

Figure 7: Training dynamics of DERL-population. We present the training dynamics of DERL-pop. and GRPO w/ Avg Reward to demonstrate the superiority of the population method.

out-of-distribution environments beyond the parametric skills learning through training data, (Huang et al., 2023; Cheng et al., 2024a; 2025). We show in this section when the generalization emerges through the training dynamic of DERL-pop.

Figure 7 illustrates the training dynamics of DERL-pop. and the comparison with the GRPO w/ Avg Reward baseline. The experiments are based on the L0 difficulty of the ScienceWorld task. For GRPO w/ Avg Reward, we train for a full 100 steps. For DERL-pop., we train the inner layers for only 33 steps each time, and then the next round of training is based on the best-performing model from the previous round, rather than starting from scratch. It demonstrate that in the first 33 steps, the two models perform on par with each other. However, starting from the second outer-loop of DERL-pop., it starts to surpass the baseline. When exceeding 66 steps, DERL-pop further shows significantly better results. This showcases how the dynamic nature of DERL-pop.'s reward function surpasses standard fixed reward function.

I Examples of Outer-loop Evolution

We demonstrate the detailed training dynamic of DERL by showcasing each Meta-Reward explored in the outer-loop on ALFWorld (L2) in Table 3. We show four outer-loop iterations. We observe that there may be a lot of low-quality meta-rewards in the early stages, but DERL can quickly learn high-quality rewards.

Table 3: Evolution of Meta-Reward structures and their corresponding reward across outer-loop training steps on ALFWorld.

Step	Meta Reward	Reward
0	$g1 * (g2 - 1)/2 + (g3 + 1) * (g4 - 1) * 2/3$	0
	$g1 + 0.5 * (g2 + 0.5 * (g3 + 0.5 * (g4)))) - 0.5 * (g2 + 0.5 * (g3 + \cdots$	0
	$g1 + 0.01 * (g2 - 0.001) + 0.0001 * (g3 - 0.0001) + 0.000005 * (g4 - \cdots$	0.8496
	$g1 * 0.5 + 0.2 * (g2 + 0.1) - 0.3/2 + 0.4 * (g3 * 0.1) + 0.1 * (g4 * \cdots$	0.8789
	$g1 * (g2 + (g3 - 1.0) * (g4 - 0.0))$	0.0234
	$g1 + 0.5 * (g2 + 0.5 * (g3 + 0.5 * (g4 + 0.5)))$	0.8848
	$g1 + 0.5 * (g2 + 0.2 * (g3 + 0.1 * (g4 + 0.05))))$	0
	$g1 * (g2 + 2 * (g3/3)) + (g4 - 4 * (g2 - 1)) - 2.0$	0.8945
1	$(0.5 * (g1 - 0.1)) + (0.5 * (g2 - 0.1)) + (0.5 * (g3 - 0.1)) + (0.5 * \cdots$	0.7793
	$g1 * (g2 + (g3 * (g4 - (g2 + (g3 * (g4 - (g3 * (g4 - (g3 * (g4 - (g3 * \cdots$	0
	$-(g1 + 0.5 * (g2 + 0.3 * (g3 - 0.2 * (g4 + 0.1 * 1))) + 0.1 * 1)/1.2$	0.0020
	$g1 * (g2 - 1) * (1 - 0.5) + 0.5 * *2 * (g3 - 1) * (1 - 0.25) + 0.25 * * \cdots$	0
	$g1 + (g2 * (g3/2)) - (g4/3)$	0.8594
	$g1 * (g2 + 0.5) + 0.2 * (g3 * 0.3 + 0.1) - 0.1 * (g4 * 0.2 + 0.5)$	0.8477
	$g1 + 2 * (g2 + 3 * (g3 - 2)) * 0.1 + 4 * 0.3 - 5 * (g4 + 1)$	0.8984
	$g1 * (g2 - 1)/2 + (g3 - 1)/3 + (g4 - 1)/4 + 3.0$	0.0098
2	$g1 + 0.5 * (g2 + 0.5 * (g3 + 0.5 * (g4))))$	0
	$g1 + 0.5 * (g2 - 0.5) + 0.3 * (g3 - 0.05) + 0.2 * (g4 + 0.05) - 0.2 * \cdots$	0
	$g1 + (g2 * (g3/2)) - (g4/4)$	0.8438
	$g1 + 0.5 * (g2 + 0.5 * (g3 + 0.5 * (g4 - 1))))$	0
	$g1 + 0.01 + 0.0001 * (g2) + 0.000001 * (g3) + 0.00000001 * (g4)$	0.8652
	$g1 * (g2 + (g3 * (g4/2))) + (g1 - (g2 + (g3 * (g4/2)))) * 0.5$	0.8867
	$g1 * 0.99 + 0.01 * (g2 + 0.99) + 0.005 * (g3 + 0.99) + 0.0005 * (g4 + \cdots$	0.8477
	$g1 + 0.5 * (g2 - 0.5) + 0.2 * (g3 - 0.5) + 0.1 * (g4 - 0.5)$	0.8906
3	$g1 + (g2/2.0) - (g3 * 0.1) + (g4 * 0.05)$	0.8438
	$g1 + 0.5 * (g2 + 0.5 * (g3 + 0.5 * (g4 + 0.5)))$	0.875
	$g1 + 0.05 * (g2 + 0.05 * (g3 + 0.05 * (g4 + 0.05)))$	0.8242
	$g1 + 0.5 * (g2/2.0) + 0.1 * (g3 * 2.0) + 0.25 * (g4/4.0)$	0.8632
	$g1 + 0.5 * (g2 + 0.4 * (g3 + 0.2 * (g4 + 0.1)))$	0.8496
	$g1 + 0.5 * (g2 + 0.5 * (g3 + 0.5 * (g4 + 0.5)))$	0.8926
	$g1 + 0.5 * (g2 + 0.5 * (g3 + 0.5 * (g4 + 0.5)))$	0.8789
	$g1 + (g2 * (g3/2)) - (g4/3)$	0.8984

J THE USE OF LARGE LANGUAGE MODELS

Multiple LLM products, including GPT-5 and Gemini-2.5-pro, are deployed to polish the writing. However, none of the paragraphs is written by LLMs directly, and all research ideas are independently proposed by authors without any AI assistance. Gemini-2.5-pro is included to assist coding, but all generated code is then carefully inspected by authors.