Learning to Discover Abstractions for LLM Reasoning

Yuxiao Qu[2,♦], **Anikait Singh**[1,♦], **Yoonho Lee**[1,♦], **Amrith Setlur**[2], **Ruslan Salakhutdinov**[1], **Chelsea Finn**[1] **and Aviral Kumar**[1]
[1]Stanford University, [2]Carnegie Mellon University, [♦]Equal Contribution

Abstract: Effective reasoning often requires going beyond pattern matching or memorization of solutions to identify and implement "algorithmic procedures" that can be used to deduce answers to hard problems. These algorithmic procedures consist of reusable primitives, intermediate results, or procedures that themselves can be applied across many problems. While current methods of RL post-training on long chains of thought ultimately desire to uncover this kind of algorithmic behavior, their sensitivity to benchmarks and the brittle and locally optimal nature of strategies learned by these systems suggest that this is far from a fulfilled promise. To instantiate this, we introduce *abstractions*: concise natural language descriptions of procedural and factual knowledge that guide the model toward successful reasoning strategies. We train models to be capable of proposing several useful abstractions given a problem, followed by RL training that incentivizes building a solution while using the information provided by these abstractions. This results in a two-agent cooperative RL training paradigm, ***RL For Abstraction Discovery*** (***RLAD***), that jointly trains an abstraction generator and an abstraction-conditioned solution generator. This bi-level setup effectively enables structured exploration, decouples learning signals pertaining to abstraction proposal and solution generation, and improves generalization to harder problems, analogous to what we would expect from hierarchical RL. Empirically, ***RLAD*** improves performance on challenging math benchmarks.

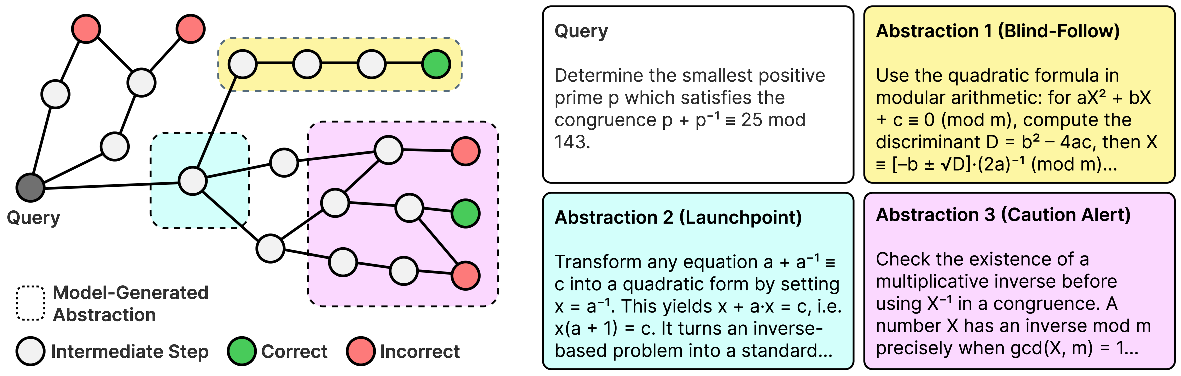
The purpose of abstraction is not to be vague,
but to create a new semantic level in which one can be absolutely precise.
— Edsger W. Dijkstra

1. Introduction

Modern machinery for solving reasoning tasks with large language models (LLMs) relies on incentivizing the use of longer chains of thought via reinforcement learning (RL). This training approach largely incentivizes "depth": subsequent training iterations increase response length by incorporating new operations that usually verify or build upon the existing line of reasoning [37]. In many hard problems, it is instead more desirable to optimize for "breadth": explore a diverse array of solution strategies, rather than committing to a seemingly optimal set of reasoning strategies right away [51, 53]. Optimizing for breadth is important: even when models optimized for depth succeed on some problems, they fail on structurally similar ones that require slightly different strategies, revealing brittle reasoning and poor generalization [39, 20, 25, 16, 29].

How can we help models discover a breadth of reasoning strategies for a given problem? Abstractly, the most natural approach is to train models to hypothesize new solutions to difficult problems and then attempt to utilize these strategies in the solution. We can do this by making models capable of discovering ***reasoning abstractions***: compressed representations of shared procedures that underlie multiple candidate solutions. For example, in math reasoning problems, such abstractions might correspond to useful intermediate lemmas or even some intermediate steps that do not succeed but illustrate what not to do. When presented in context, these abstractions function like hints on an exam, enabling LLMs to solve harder problems by building on the insights appearing in the abstraction, rather than from scratch. That

Corresponding author(s): anikait@stanford.edu, yuxiaoq@andrew.cmu.edu, yoonho@cs.stanford.edu
Project Website: https://rl-abstraction-discovery.github.io/


Query
Model-Generated Abstraction
Intermediate Step
Correct
Incorrect
Query
Abstraction 1 (Blind-Follow)
Abstraction 2 (Launchpoint)
Abstraction 3 (Caution Alert)

Query

Determine the smallest positive prime p which satisfies the congruence $p + p^{-1} \equiv 25 \bmod 143$.

Abstraction 1 (Blind-Follow)

Use the quadratic formula in modular arithmetic: for $aX^2 + bX + c \equiv 0 \pmod m$, compute the discriminant $D = b^2 - 4ac$, then $X \equiv [-b \pm \sqrt{D}]\cdot(2a)^{-1} \pmod m$...

Abstraction 2 (Launchpoint)

Transform any equation $a + a^{-1} \equiv c$ into a quadratic form by setting $x = a^{-1}$. This yields $x + a\cdot x = c$, i.e. $x(a + 1) = c$. It turns an inverse-based problem into a standard...

Abstraction 3 (Caution Alert)

Check the existence of a multiplicative inverse before using X^{-1} in a congruence. A number X has an inverse mod m precisely when $\gcd(X, m) = 1$...

Figure 1: ***Reasoning abstractions illustrated in the solution-space graph for a problem.*** We represent the problem as a node (labeled "query") and various traces (both correct and incorrect) attempting to solve the problem as a graph. In this illustration, *reasoning abstractions* describe useful high-level structure in this space of all traces, such as **(1)** an abstract idea that can induce a predictable sequence of successful states (blind follow), **(2)** an initial step that informs the approach to take (launchpoint), or (3) a common critical error to avoid (caution alert). Note that reasoning abstractions encode a mix of procedural and factual knowledge that may be helpful.

is, when conditioned on abstractions, an LLM should learn to implement useful algorithmic procedures via RL that can utilize and compose the procedural information in the context as best as possible and apply it to the problem at hand. This naturally enhances the diversity of solution strategies and behaviors that a model learns to utilize when encountering an unseen problem, in contrast to committing to a narrow set of approaches, as existing models do. In RL terminology, abstractions serve as high-level subgoals, skills, or priors—which depend on context—guiding the low-level solution-generating policy.

In this work, we imbue LLMs with the capability of proposing and utilizing reasoning abstractions for reasoning problems. Concretely, we build reasoning models that, first, given an input problem, propose one or more reasoning abstractions, expressed in natural language. Subsequently, they generate a solution that utilizes the information and principles prescribed by these abstractions. To achieve this, we jointly train two LLMs via RL: **(1)** an abstraction generator, and **(2)** an abstraction-conditioned solution generator. The abstraction generator is rewarded for the improvement in the accuracy of the solution generator, stemming from the abstractions it provides. The solution generator is rewarded to maximize accuracy in solving a problem while utilizing the abstraction. To obtain a good initialization for the abstraction generator, we run supervised fine-tuning (SFT) on abstractions gathered by prompting an offline teacher model (o4-mini). This teacher plays the role of a human data collector and is *not* accessed after SFT. For the solution generator, we generate solutions from an LLM, conditioning on the abstraction. We call this approach ***RL For Abstraction Discovery*** (***RLAD***).

The main contribution of this paper is the notion of *reasoning abstractions*, how they can be obtained and amplified via RL training, and an illustration of how they can be used to improve reasoning performance. Concretely, we propose an approach to imbue LLMs with the capability of proposing abstractions and evaluate the model on several math reasoning benchmarks: AIME 2025 [23], DeepScaleR Hard [37], and AMC 2023. We find an average 44% improvement over state-of-the-art long chain-of-thought RL approaches (i.e., DAPO [51]) on AIME 2025, and show an effective benefit from generating diverse abstractions over brute-force solution sampling.

	Breast Cancer Detection	Tweet Hate Speech Detection	Corporate Lobbying Relevance	Bank Note Authentication
Abstraction	Classify breast masses as malignant or benign using BI-RADS, shape, and margin criteria. ...If BI-RADS ≥ 5, then malignant. ...if BI-RADS = 4 AND shape = irregular AND margin = ill-defined, then malignant...	...If the text contains explicit derogatory slurs (e.g., b****, c***, s****, h**, r********), classify as hate speech. ...if the text degrades or dehumanizes a protected group (nationality, race, religion...	...If the bill addresses regulation, labeling, pricing, reimbursement, R&D funding, or licensing of the company's core products or services, label "Yes." ...Else if the bill alters taxes, credits, bonds, infrastructure...	1. If variance > 4, predict Fake. 2. Else if variance < −3, predict Original. 3. Else if skewness > 5, predict Fake. 4. Else if entropy > 0 and skewness > 1, predict Fake.
GPT-4o-mini	45%	60%	88%	45%
GPT-4o-mini + Abs	**90%**	**90%**	**94%**	**100%**

Figure 2: ***Examples of good reasoning abstractions in non-math domains.*** Adding the abstraction to the prompt of GPT-4o-mini consistently improves performance on unseen instances.

2. Related Work

Scaling test-time compute and exploration. Recent work highlights the promise of scaling test-time compute in different ways. One approach involves parallel sampling: sampling multiple reasoning rollouts and then selecting a winner via a scoring rule [45, 47, 4, 7, 41, 49, 11, 42]. A complementary line of work iteratively edits a single trace, attempting to implement a sequential search within a single solution trace [21, 31, 32, 13]. As such, the sequential approach performs a bit worse on harder problems [42, 33], where it often gets trapped in strategies that seem optimal but aren't actually [27]. Yet it still performs better than parallel search on easier and medium difficulty problems [42]. Our approach of proposing and leveraging abstractions enables a kind of hybrid between sequential sampling and parallel sampling, guided by the proposed abstractions. This should address the failure modes of current methods. Prior work has also utilized hand-designed scaffolds to integrate multi-step evaluations of intermediate hypotheses into reasoning [50, 12, 11, 15]. In contrast, we do not rely on pre-defined interfaces but learn to *automatically* propose useful abstractions.

Using prior knowledge for LLM reasoning. Several threads of work converge on the idea that *textual artifacts*, such as examples, plans, or prompts, can serve as reusable knowledge that guides LLM behavior. Existing retrieval-augmented generation (RAG) pipelines assume a static corpus, typically of human-written text, and focus on improving retrieval heuristics [14, 3, 44, 46, 2, 17]. Many works use LLMs to learn or refine prompts, either in an input-agnostic fashion [58, 48, 30, 8] or through input-specific edits based on feedback [40, 22, 9, 54, 18]. Other related work explores the use of synthetic demonstrations [56], scratchpads [26], and memory-augmented agents [36] to encode prior problem-solving knowledge. Two recent works demonstrate that LLMs can accumulate and reuse their own experience across tasks [57, 43]. While one can view our reasoning abstractions as a form of prior procedural and factual knowledge produced before the model's solution attempt, this knowledge is **(a)** input-dependent and **(c)** is not acquired from an external source at deployment, but rather is "proposed" by the model itself. Imbuing models with this capability requires a two-player cooperative RL training procedure that we develop. To our knowledge, such procedures have not been used for generating textual artifacts of any type, let alone the abstractions we consider.

3. Preliminaries and Notation

We study reasoning with LLMs, where the LLM is given a problem $\mathbf{x}$ and generates a stream of tokens $\widetilde{\mathbf{y}}$ that ends in an estimate of the answer. We assume access to a rule-based ground-truth 0/1 reward $\mathrm{Acc}_{\mathbf{x}}(\widetilde{\mathbf{y}}, \mathbf{y}^\star) \in \{0, 1\}$ that measures correctness of the produced answer $\widetilde{\mathbf{y}}$ against the ground-truth solution $\mathbf{y}^\star$ for the question $\mathbf{x}$. For training, we are given a dataset $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_i, \mathbf{y}_i^\star)\}_{i=1}^N$ of problems $\mathbf{x}_i$ and solutions $\mathbf{y}_i^\star$ that end with the correct answer. Our goal is to train the LLM $\pi(\cdot|\mathbf{x})$ such that it achieves

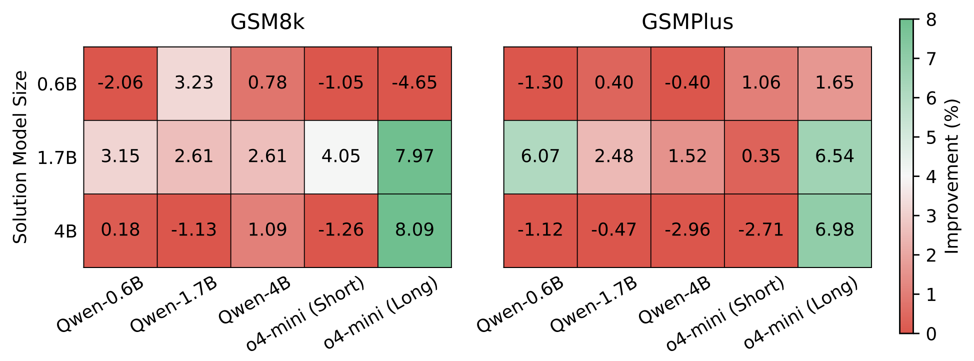


Figure 3: ***Many factors govern gains from abstractions***—abstraction generator, abstraction length, and solver capacity. Notably, without further training, only the strongest generator yields consistent accuracy boosts. The heatmap shows relative accuracy change (%) on GSM8k and GSMPlus. Rows denote solver size, columns denote abstraction source. In Section 5, we will describe our method that can train smaller models to generate high-quality abstractions and use them.

high rewards on a test distribution of problems $\mathcal{P}_{\text{test}}$.

We primarily evaluate models in terms of their average accuracy under $\mathcal{P}_{\text{test}}$. We also measure the pass@k metric, where for problem $\mathbf{x}$, we sample k solutions $\widetilde{\mathbf{y}}_1, \ldots, \widetilde{\mathbf{y}}_k \sim \pi(\cdot|\mathbf{x})$, and consider the problem to be solved if any of these k traces is correct. This metric couples accuracy with diversity, i.e., it attains the largest value when the model effectively finds diverse, good responses. To reduce variance in estimating pass@k, we sample $n \geq k$ samples per problem and use the unbiased estimator introduced in OpenAI Codex [5]: $1 - \binom{n-c}{k} / \binom{n}{k}$, where $c \leq n$ is the number of correct samples.

4. Reasoning Abstractions and Why They Are Useful

Solving reasoning problems often requires composing both *procedural* knowledge (e.g., how to apply a root-finding algorithm) and *factual* knowledge (e.g., relevant lemmas or intermediate results). Current approaches typically train reasoning models to elicit such knowledge entirely through reinforcement learning (RL) with long chains of thought. However, this is often ineffective as RL often tends to optimize for "depth": producing longer traces where each subsequent segment extends the last (e.g., verifying prior calculations), rather than "breadth", which involves exploring diverse solution strategies. In this section, we introduce *abstractions*, that provide a mechanism for explicitly encoding a range of procedural and factual concepts useful in solving a problem.

Intuition. We instantiate reasoning abstractions as concise textual descriptions of core insights that are useful for solving a problem. We show some examples of abstractions in Figure 1, in the domain of math reasoning. Here, these abstractions can correspond to useful techniques (e.g., "launchpoint" in Figure 1), a useful lemma or heuristic principle (e.g., "blind-follow" in Figure 1), and cautionary examples that demonstrate common pitfalls encountered when solving a problem (e.g., "caution alert" in Figure 1). These abstractions distill complex reasoning patterns and potential approaches into useful nuggets, allowing models to generalize across structurally similar problems.

Conceptual understanding. With this intuitive notion in place, we now consider a more conceptual definition. We can view abstractions as a compressed representation of the reasoning procedures embedded within longer chains of thought. Consider the space of possible reasoning traces for a given problem, which can be visualized as a graph structure where nodes represent intermediate states encountered when solving a question (see Figure 1). Good abstractions identify useful substructures within this larger reasoning graph. For example, an abstraction can capture if a set of strategies lead to a similar outcome or another set of tactics leads to an error being consistently made.

Concretely, let us denote the LLM policy that produces a solution conditioned on the problem $\mathbf{x}$ as $\pi_\theta^{\text{sol}}(\cdot|\mathbf{x})$. A good abstraction $\mathbf{z}$ is a sequence of tokens that provides some useful procedural and factual information to improve model performance:

$$\mathbb{E}_{\widetilde{\mathbf{y}}\sim\pi_\theta^{\text{sol}}(\cdot|\mathbf{x},\mathbf{z})}\left[\text{Acc}(\widetilde{\mathbf{y}},\mathbf{y}^*)\right] > \mathbb{E}_{\widetilde{\mathbf{y}}\sim\pi_\theta^{\text{sol}}(\cdot|\mathbf{x})}\left[\text{Acc}(\widetilde{\mathbf{y}},\mathbf{y}^*)\right]. \tag{1}$$

How can we generate good reasoning abstractions? Do good reasoning abstractions exist? We now attempt to understand whether good reasoning abstractions exist and how one might discover them. Perhaps the most natural way to obtain an initial set of reasoning abstractions is to collect a diverse set of traces that attempt to solve a problem and then summarize the useful concepts appearing in these traces, as illustrated in Figure 1. To evaluate the existence and utility of reasoning abstractions (before developing our method to train LLMs to discover useful reasoning abstractions), we instantiate this idea by prompting a model to generate solutions for a given problem and prompting a stronger model to deduce patterns from the responses of the first model. Concretely, we utilize the Qwen3 series of models to produce solutions and a stronger reasoning model, `o4-mini` [1], to generate abstractions. While this approach is not perfect and is not meant to be our final approach, it still enables us to validate the feasibility of reasoning with abstractions. To ensure that the abstractions do not “leak” content of the solution, we verify that post-hoc prompting the solver with only the abstraction and no question yields zero accuracy.

Results and observations. After generating abstractions as above, we measure their quality by evaluating Equation 1, i.e., by checking if conditioning the problem solver on a set of abstractions improves its accuracy. Results in Figure 3 show that conditioning a problem solver on generated abstractions improves accuracy when three conditions hold simultaneously: (i) the abstraction is not too short (e.g., not just a few words that are not informative) and generated by a strong generator (`o4-mini(Long)`) and (ii) the solution generator has sufficient capability (`Qwen3-1.7B` or `Qwen3-4B`) of interpreting and utilizing the generated abstraction. These results confirm that good abstractions (satisfying Eq. 1) exist for math problems, but neither the ability to generate them nor the ability to leverage them in solutions arises naturally. In Section 5, we will describe our method for explicitly training models to propose and use such abstractions effectively.

Good abstractions exist in many domains. We also find that this procedure can be used to identify an initial set of useful reasoning abstractions on many problem domains, including healthcare, human behavior, legal reasoning, and web security. Of course, the proportion of abstraction devoted to procedural knowledge and factual knowledge is different in these domains compared to math reasoning. Nonetheless, we find that using reasoning abstractions improves performance by 30% on average over 37 tasks from RAFT [1], CLUES [24], and LegalBench [10]. We show four representative abstractions in Figure 2 and full results in Table 3 in the appendix.

Takeaways: Reasoning abstractions improve performance

Reasoning abstractions summarize procedural and factual knowledge that is useful for learning to solve problems via diverse strategies. Prompting abstractions generated by merely prompting models already improves performance by 30% on average for reasoning.

[1]`o4-mini` serves solely as a tool for analysis and offline data collection; no calls are made to it during training or evaluation.

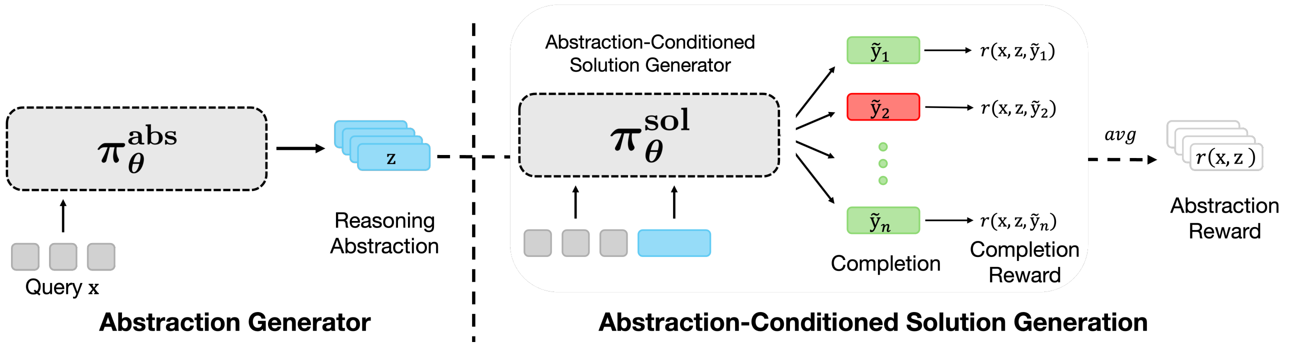


Figure 4: **RLAD *training paradigm.*** We train an abstraction generator, π_θ^{abs}, that proposes some reasoning abstractions conditioned on the question $\mathbf{x}$, denoted as $\mathbf{z}$. Then, the solution generator, π_θ^{sol}, is trained to produce a response, $\tilde{\mathbf{y}}$ conditioned on the generated abstraction $\mathbf{z}$. The reward used for training π_θ^{abs} corresponds to the average success rate of the solution generator conditioned on the proposed abstraction.

5. Learning to Discover Reasoning Abstractions

Having defined the notion of reasoning abstractions and demonstrated that they can improve performance when applied to tackling reasoning problems, we now aim to develop an approach that enables us to improve an LLM's ability to propose and utilize abstractions. Doing so requires training an abstraction generator: an LLM, $\mathbf{z} \sim \pi_\theta^{\text{abs}}(\cdot|\mathbf{x})$ that proposes candidate abstractions $\mathbf{z}$ given problem $\mathbf{x}$, and an abstraction-conditioned solution generator, $\mathbf{y} \sim \pi_\theta^{\text{sol}}(\cdot|\mathbf{x}, \mathbf{z})$, that produces a solution $\mathbf{y}$ given $\mathbf{x}$ and abstraction $\mathbf{z}$. Note that $\mathbf{z}$ is parameterized as a variable-length string of tokens and might consist of one or more pieces of information or procedures. While our approach applies to the case where π_θ^{abs} produces more than one abstraction, we abuse notation and subsume multiple abstractions into a single one to avoid notational clutter. In this section, we describe ***RL For Abstraction Discovery*** (***RLAD***), our method for training these models via RL.

5.1. Training π_θ^{abs} and π_θ^{sol} via RL

The core principle behind our approach is that an abstraction $\mathbf{z}$ is successful at a given problem $\mathbf{x}$ if it can maximally help $\pi_\theta^{\text{sol}}(\cdot|\mathbf{x}, \mathbf{z})$ find correct responses to question $\mathbf{x}$, without actually leaking the answer itself. To convert this into an RL objective, we design a reward function that rewards an abstraction $\mathbf{z}$ with the expected success of solutions generated by π_θ^{sol} conditioned on $\mathbf{z}$:

$$r_{\pi_\theta^{\text{sol}}}(\mathbf{x}, \mathbf{z}) := \mathbb{E}_{\tilde{\mathbf{y}} \sim \pi_\theta^{\text{sol}}(\cdot|\mathbf{x},\mathbf{z})} \left[\text{Acc}_{\mathbf{x}}(\tilde{\mathbf{y}}, \mathbf{y}^*) \right], \tag{2}$$

where $\mathbf{y}^*$ is the ground-truth answer and $\text{Acc}_{\mathbf{x}}(\cdot, \cdot)$ denotes the 0/1 accuracy on problem $\mathbf{x}$. To train π_θ^{sol}, one can then adopt the fairly straightforward approach of maximizing 0/1 binary outcome reward, now conditioned on a given abstraction $\mathbf{z}$ sampled previously from π_θ^{abs}, akin to recent results RL [6]. Formally, we set the reward for a solution as: $r(\mathbf{x}, \mathbf{z}, \tilde{\mathbf{y}}) := \text{Acc}_{\mathbf{x}}(\tilde{\mathbf{y}}, \mathbf{y}^*)$. With these reward functions in place, perhaps the most natural approach then would be to train π_θ^{abs} to maximize $r_{\pi_\theta^{\text{sol}}}$ for a fixed π_θ^{sol} on a dataset of prompts $\mathcal{D}_{\pi_\theta^{\text{abs}}}$, while also iteratively training π_θ^{sol} to maximize the reward function r on modified prompts generated by concatenating a set of sampled abstraction $\mathbf{z}$ on a dataset of problems, $\mathcal{D}_{\pi_\theta^{\text{sol}}}$. This maximization could be done via on-policy RL methods like GRPO [38] or (batched) offline RL methods like DPO [35] and STaR [55].

Challenges with naïve reward design. While the approach so far is extremely simple, it presents some challenges. In particular, the reward functions defined above can result in spurious, undesirable solutions in a rather nuanced manner: **(1)** if π_θ^{abs} learns to solve problem $\mathbf{x}$ in its entirety, it will still be rewarded highly by $r_{\pi_\theta^{\mathrm{sol}}}$ but is not a desirable abstraction; **(2)** if $\pi_\theta^{\mathrm{sol}}$ is too weak or too strong, such that it is either always able to solve the problem $\mathbf{x}$ or never solves it, then $r_{\pi_\theta^{\mathrm{sol}}}$ will not provide a meaningful signal to update $\pi_\theta^{\mathrm{abs}}$; and **(3)** similar to the above failure modes, training $\pi_\theta^{\mathrm{sol}}$ via on-policy RL may result in it ignoring the abstraction $\mathbf{z}$ altogether no matter how useful it is. Abstractly, all of these challenges correspond to a "signal obfuscation" problem, where an imbalance in the strength of $\pi_\theta^{\mathrm{abs}}$ and $\pi_\theta^{\mathrm{sol}}$ may drown out the learning signal for the other.

Modifying reward design. To address these signal obfuscation challenges, we make a slight but consequential change to the training process. In particular, we train $\pi_\theta^{\mathrm{sol}}$ on a mixture of prompts $\mathbf{x}$ augmented by abstractions $\mathbf{z}$ and prompts $\mathbf{x}$ without any abstractions at all. In this process, while we utilize $\mathrm{Acc}_{\mathbf{x}}$ as discussed above on a given response, we simply zero out rewards for any trace generated on $\mathbf{x}$ without abstractions. When utilizing KL-constrained RL, e.g., GRPO [38], $\pi_\theta^{\mathrm{sol}}$ is now trained to closely mimic the distribution of responses as the reference LLM on questions $\mathbf{x}$ but must attempt to find ways to optimize reward on the same question $\mathbf{x}$ when augmented with an abstraction. This can be accomplished only when $\pi_\theta^{\mathrm{sol}}$ learns to utilize the provided abstraction carefully, hence addressing one of the challenges above. Second, we ensure that $\mathbf{z} \sim \pi_\theta^{\mathrm{abs}}(\cdot|\mathbf{x})$ itself does not contain the answer to the question $\mathbf{x}$, which means that $\mathrm{Acc}(\mathbf{z}, \mathbf{y}^*)$ is penalized to be small. Finally, we utilize separate partitions of the training dataset to train $\pi_\theta^{\mathrm{abs}}$ and $\pi_\theta^{\mathrm{sol}}$, thereby avoiding overfitting on subsets of the data. We present detailed ablations of these design choices in Appendix B.3. Formally, the updated versions of these reward functions are shown as:

$$r(\mathbf{x}, \mathbf{z}, \widetilde{\mathbf{y}}) := \begin{cases} 0, & \text{if } \mathbf{z} = \emptyset \\ \mathrm{Acc}_{\mathbf{x}}(\widetilde{\mathbf{y}}, \mathbf{y}^*), & \text{otherwise} \end{cases} \tag{3}$$

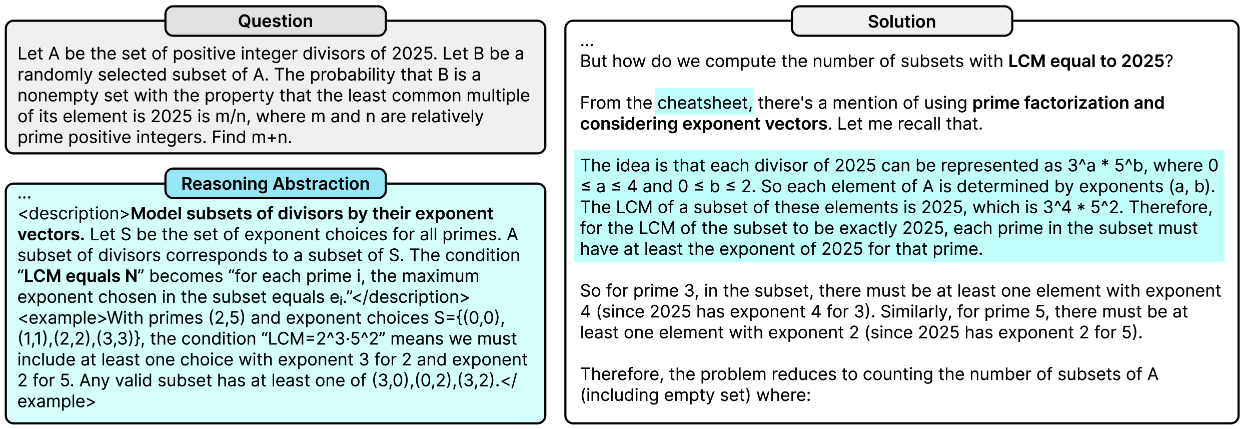
$$r_{\pi_\theta^{\mathrm{sol}}}(\mathbf{x}, \mathbf{z}) := \mathbb{E}_{\widetilde{\mathbf{y}} \sim \pi_\theta^{\mathrm{sol}}(\cdot|\mathbf{x}, \mathbf{z})}[\mathrm{Acc}_{\mathbf{x}}(\widetilde{\mathbf{y}}, \mathbf{y}^*)]. \tag{4}$$

5.2. Warmstarting $\pi_\theta^{\mathrm{sol}}$ and $\pi_\theta^{\mathrm{abs}}$ from Good Initializations

While the above approach prescribes a recipe for RL training of $\pi_\theta^{\mathrm{abs}}$ and $\pi_\theta^{\mathrm{sol}}$, any such recipe critically relies on the ability of the initialization to generate somewhat meaningful abstractions and solutions conditioned on the abstraction input, respectively, from the beginning of RL training. How can we ensure that our model initializations have this capability? Inspired from the approach of running an initial phase of SFT to imbue into the model the basic structure of a long chain-of-thought before running RL [6, 33], we run an initial phase of SFT to imbue into $\pi_\theta^{\mathrm{abs}}$ and $\pi_\theta^{\mathrm{sol}}$ the basic capabilities of producing abstractions and attempting to follow abstractions respectively, even if the resulting models are not very good. For this initial warm-start phase, we follow the protocol outlined in Section 4 and construct a corpus $\{(\mathbf{x}_i, \mathbf{z}_i, \mathbf{y}_i)\}_{i=1}^{M}$ by prompting a teacher model once, exactly as one would collect human-written hints. For each training problem-solution pair $(\mathbf{x}, \mathbf{y}^*)$, in our training set, we first generate an abstraction $\mathbf{z}$ using an instruction-tuned model, discarding any that leak $\mathbf{y}^*$. We then sample a solution trace $\mathbf{y}$ conditioned on $(\mathbf{x}, \mathbf{z})$. As mentioned in Section 5.1, we partition this corpus into non-overlapping splits for $\pi_\theta^{\mathrm{sol}}$ and $\pi_\theta^{\mathrm{abs}}$ to avoid overfitting.

5.3. Practical Approach and Algorithm Details

For warmstarting the abstraction generator, we utilize abstractions collected once from an offline teacher (`o4-mini`). We then use a weaker solution generator (`GPT-4.1-mini`) to evaluate the efficacy of each abstraction by comparing the success rate of the solution generator with and without the abstraction. We filter abstractions that don't increase solution generation performance to form our seed set of abstractions.



Question

Let A be the set of positive integer divisors of 2025. Let B be a randomly selected subset of A. The probability that B is a nonempty set with the property that the least common multiple of its element is 2025 is m/n, where m and n are relatively prime positive integers. Find m+n.

Reasoning Abstraction

...
<description>**Model subsets of divisors by their exponent vectors.** Let S be the set of exponent choices for all primes. A subset of divisors corresponds to a subset of S. The condition "**LCM equals N**" becomes "for each prime i, the maximum exponent chosen in the subset equals e_i."</description>
<example>With primes (2,5) and exponent choices S={(0,0),(1,1),(2,2),(3,3)}, the condition "LCM=2^3·5^2" means we must include at least one choice with exponent 3 for 2 and exponent 2 for 5. Any valid subset has at least one of (3,0),(0,2),(3,2).</example>

Solution

...
But how do we compute the number of subsets with **LCM equal to 2025**?

From the cheatsheet, there's a mention of using **prime factorization and considering exponent vectors**. Let me recall that.

The idea is that each divisor of 2025 can be represented as 3^a * 5^b, where 0 ≤ a ≤ 4 and 0 ≤ b ≤ 2. So each element of A is determined by exponents (a, b). The LCM of a subset of these elements is 2025, which is 3^4 * 5^2. Therefore, for the LCM of the subset to be exactly 2025, each prime in the subset must have at least the exponent of 2025 for that prime.

So for prime 3, in the subset, there must be at least one element with exponent 4 (since 2025 has exponent 4 for 3). Similarly, for prime 5, there must be at least one element with exponent 2 (since 2025 has exponent 2 for 5).

Therefore, the problem reduces to counting the number of subsets of A (including empty set) where:

Figure 5: ***Example of a reasoning abstraction.*** Here, we provide an example of a reasoning abstraction for a given problem. In the solution, we see (**in blue**) references to the abstraction and keywords from the abstraction being utilized in the thinking trace of the reasoning model.

Then, we run SFT for 5 epochs on the seed dataset to obtain an initial abstraction generator. For solution generation, we utilize Qwen-3-1.7B [34], a 1.7B reasoning model distilled from Qwen-3-32B.

After SFT, we employ ***RLAD*** to further fine-tune the abstraction generator and abstraction-conditioned solution generator via RL. For the abstraction generator, we opt to use "batched" offline RL instantiation of our approach via RFT [52] and RPO [28], since reward computation by rolling out the solution generator on the fly was infeasible in our RL infrastructure and compute. To train the solution generator, we utilize the DAPO approach [51] and include token-level policy loss normalization, asymmetric clipping, and a curriculum for prompt difficulty and length. Building upon implementation of concurrent work [37], we employ a two stage curriculum where we partition the DeepScaleR [19] mixture by success rate of the base model into three sets: (1) easy, (2) medium, and (3) hard, where we fine-tune first on easy problems with an 8K token budget and then on medium problems. We utilize the hard split as a held-out evaluation subset, which we denote as `DeepScaleR [Hard]`. We outline hyperparameters and details in Appendix A.1 and provide a pseudocode in Algorithm 1.

Takeaways: *RLAD* method design

RLAD jointly optimizes an abstraction generator π_θ^{abs} and solution generator π_θ^{sol} with RL. We initialize π_θ^{abs} by distilling traces from an offline teacher model (`o4-mini`). We optimize the two models in an alternating manner, using reward functions that incentivize π_θ^{sol} to utilize abstractions and discourage π_θ^{abs} from "leaking" the response to the input problem (Eq. 3)

6. Experimental Evaluation

The goal of our experiments is to evaluate the efficacy of ***RLAD*** in improving the reasoning capabilities of LLMs through abstraction-guided solution generation. Specifically, we aim to answer the following research questions: **(1)** Does ***RLAD*** improve pass@1 accuracy across several mathematical reasoning benchmarks compared to direct solution generation? **(2)** How does ***RLAD*** scale as more abstractions and solutions are generated? **(3)** What makes the generated abstractions useful, how faithfully are they followed, and how do they guide and improve solution generation? To this end, we compare ***RLAD*** with strong base models on three representative mathematical reasoning datasets: AMC 2023, AIME 2025, and DeepScaleR Hard [19], which itself is a subset of hard problems from the OmniMATH mixture on which DeepSeek-R1 distilled Qwen-32B model attains an accuracy of $\leq 10\%$. We also conduct several ablations to better understand the abstractions produced by ***RLAD***.

6.1. Main Performance Results for *RLAD*

We evaluate ***RLAD*** in three settings: (1) **w/o abs**,without abstractions; (2) **w/ abs (avg)**, average performance over generations conditioned on 4 proposed abstractions per problem; and (3) **w/ abs (best)**: using the best-performing abstraction (in a set of 4 proposed abstractions per problem).

Observe that ***RLAD*** consistently outperforms the base model and variant fine-tuned with RL on the same prompts via DAPO [51], but without any abstractions, across all settings and benchmarks (Table 1). This highlights that ***RLAD*** can propose and leverage abstractions to improve its reasoning performance. We also note that these performance gains are not limited to abstraction-conditioned inference: even in the **w/o abs** setting, where no abstraction is provided during inference, ***RLAD*** improves over the prior methods, when trained with abstractions via ***RLAD***. This suggests that exposure to diverse abstractions during training enhances the model's general reasoning ability. We observe similar trends on additional benchmarks, including AIME 2024 and HMMT 2025 (see Appendix B.2), where ***RLAD*** improves in the w/o abs setting.

Approach	**AIME 2025**			**DeepScaleR [Hard]**			**AMC 2023**		
	No Abs	Abs (Mean)	Abs (Best)	No Abs	Abs (Mean)	Abs (Best)	No Abs	Abs (Mean)	Abs (Best)
Qwen-3-1.7B	33.75	36.25	40.00	20.21	22.14	32.50	86.41	78.01	84.53
+ DAPO	37.92	34.90	39.79	21.67	21.88	33.54	86.41	81.99	88.44
+ ***RLAD***	38.04	42.45	48.33	23.54	24.84	35.54	87.25	88.35	91.72

Table 1: **Pass@1 accuracy across three math reasoning benchmarks. *RLAD*** achieves consistent gains in both abstraction-conditioned and w/o abstraction settings.

In Appendix C, we also measure the performance of ***RLAD*** when different budgets are allowed for reasoning – while Table 1 measures performance at a budget of 32K tokens, we also measure performance at 8K and 16K budgets and find ***RLAD*** to be more effective compared to the comparisons.

6.2. Understanding Properties of *RLAD*

Compute tradeoffs between abstraction and solution generation. We now study how to allocate compute between generating diverse abstractions and sampling solutions conditioned on them to attain maximal performance within a given budget on the total sampling allowed. This corresponds to a "compute-optimal strategy" [42] for partitioning compute between abstraction and solution generation. If the model typically fails by making small local errors in its computations, then additional concise abstractions may not help it as much as simply trying again. In contrast, if the model tends to pursue a seemingly plausible but incorrect approach and is unable to recover or switch to a better one, then conditioning on diverse abstractions can help by offering alternative high-level approaches toward the correct answer. In other words, when the model tends to explore "depth" over "breadth" of solution strategies, abstractions can help improve performance. With this intuition, we hypothesize that when the compute budget permits only a limited number of samples, allocating more compute to sampling multiple solutions will enable the model to succeed at least once. In other words, sampling multiple solutions for the same abstraction will result in a higher pass@k performance. However, once pass@k for a single abstraction begins to saturate, performance gains are more likely to come from scaling the diversity of abstractions, which enables the model to explore qualitatively different regions of the solution space.

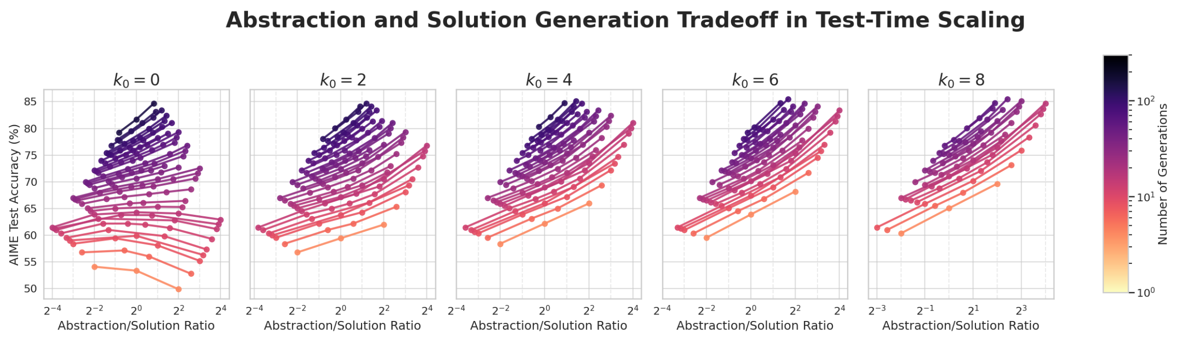
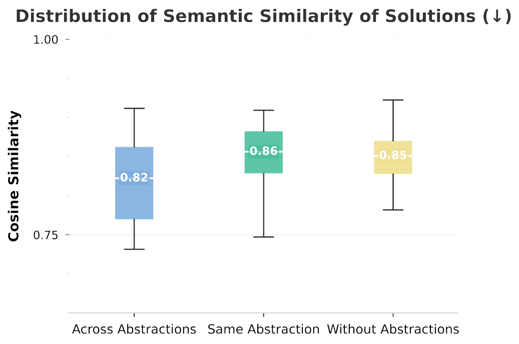


Figure 6: ***Tradeoff of abstraction and solution generation on AIME 2025.*** As the compute budget increases, we find better performance efficiency when allocating our budget to abstraction generation rather than solution generation, for all values of normalization offset k_0.



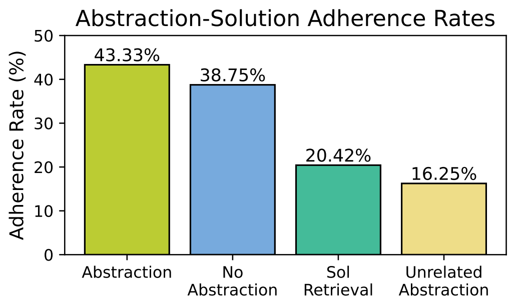


Figure 7: **Abstraction-conditioned solution generation analysis.** ***RLAD*** produces solutions with **(left)** greater semantic diversity across different abstractions and **(right)** higher abstraction adherence than baselines.

To visualize this tradeoff, we plot *iso-compute* scaling curves under a fixed compute budget, where multiple abstractions are generated and multiple solutions are sampled per abstraction. Specifically, we denote the number of abstractions as m and the number of solutions sampled per abstraction as k. To better isolate the effect of abstraction diversity, we introduce a normalization offset k_0, which accounts for performance gains that do not stem from new strategies, but arise from local modifications in the solution and the model's stochasticity (e.g., small edits that do not require new abstractions). Figure 6 shows multiple *iso-compute* frontiers, one for each total compute budget. Each curve corresponds to a fixed total number of abstraction-conditioned samples, with compute defined as $m \times (k - k_0)$, where m is the number of abstractions, k is the number of solutions per abstraction, and k_0 offsets for solutions. This formulation captures the number of "meaningful" samples that go beyond the model's local neighborhood. The x-axis plots the ratio between abstractions and adjusted solutions, $m/(k - k_0)$ We observe in Figure 6 that across $k_0 \in \{0, 2, 4, 6, 8\}$, shifting compute toward abstractions consistently yields greater performance improvements than allocating the same additional compute to solution refinements. ***This supports the conclusion that*** *once local errors in the chain-of-thought have been addressed, it is more effective to increase the breadth of the search through abstraction conditioning rather than to continuing to scale up sampling alone.*

Understanding behavior of the abstraction-conditioned solution generator. A desirable property of the solution generator is the ability to follow proposed abstractions. To study this, we prompt a strong reasoning model `o4-mini` to classify whether a particular solution trace produced by a trained solution generator closely adheres to a given abstraction. We ask for a binary decision on each pair of hint and solution, and measure the adherence rate across 200 pairs. In Figure 7 (right), we report adherence rates under four conditions: `Abstraction` (solution generated with the intended abstraction), `No Abstraction` (solution with only question), `Retrieval` (a semantically similar past solution is retrieved), and `Unrelated Abstraction` (solution conditioned on an abstraction from a different problem). We find that the `Abstraction` condition achieves the highest adherence rate, outperforming all control variants on average. Intuitively, this means that the trained solution generator is more likely to follow the strategy or guidance of a given abstraction. Additionally, we measure the semantic similarity of solutions generated without abstraction conditioning, conditioned on the same abstraction, and across abstractions. Here, across abstractions, the semantic similarity of solutions is lower, indicating that abstractions allow for higher solution diversity.

Categorizing abstractions. As outlined in Appendix C, we classify each model-generated abstraction into four mutually-exclusive categories: (1) `Caution Alert` that warns the solver to avoid a specific approach; (2) `Productive Launchpoint` that suggests strategic framings or problem reformulations that open high-potential solution paths; (3) `Blind-Follow Trajectory` that prescribes repeatable, step-by-step procedures executable without further insight; and (4) `Structural Shortcut` that leverages abstract insights or invariants to collapse multiple reasoning steps into a single leap. In Figure 8, we show that after training via ***RLAD***, the distribution over these categories shifts, with a notable increase in blind-follow abstractions, which a stronger reasoning model classifies as an effective reasoning path to a successful solution as seen in Appendix C.

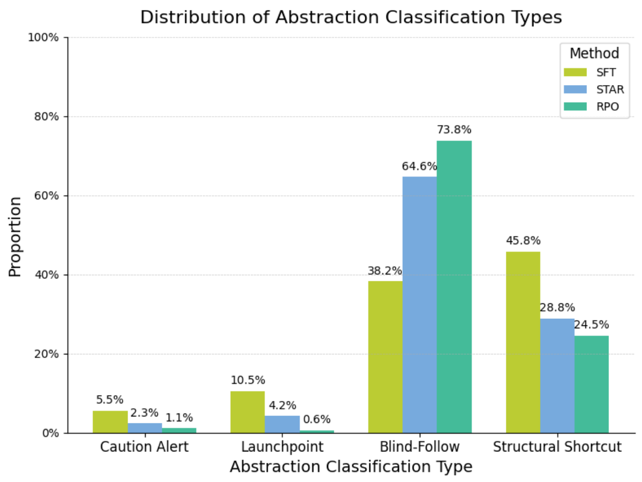


Figure 8: **Abstraction Categorization *RLAD*** produces a diverse characterization of abstractions, which we characterize by prompting `o4-mini`.

Takeaways: Experimental Results

RLAD outperforms RL fine-tuning approaches that do not propose or leverage abstractions on math reasoning. Jointly scaling the number of abstractions and solution samples enables continued performance gains even when scaling solutions alone begins to saturate.

7. Discussion, Conclusion, Limitations, and Societal Impacts

We introduce reasoning abstractions: concise representations of procedural and factual knowledge expressed in natural language, as a means to broaden the reasoning strategies of LLMs. Our method, ***RLAD***, instantiates a two-player training framework that trains an abstraction generator and an abstraction-conditioned solution generator. ***RLAD*** yields consistent improvements across several mathematical reasoning benchmarks, outperforming existing methods for training LLMs to reason. Moreover, we demonstrate that allocating compute toward generating diverse abstractions, rather than increasing solution sampling alone, yields greater performance gains. This highlights abstractions as a complementary axis for scaling test-time compute. While longer chain-of-thoughts and parallel solution sampling provide existing ways to scale compute, using abstractions provides an orthogonal axis to improve performance.

While we demonstrated that abstractions can be helpful, our evaluation is limited to mathematical reasoning tasks, leaving open-ended reasoning unexplored. Finally, ***RLAD*** incurs additional computational overhead, and training a single model that can both generate abstractions and solutions is open for future work.

From a societal perspective, this work has the potential to enable more reliable and interpretable reasoning in high-stakes applications such as education, scientific discovery, and decision support. However, stronger reasoning abilities could also be misused to generate more persuasive misinformation or automate complex manipulation.

References

[1] Neel Alex, Eli Lifland, Lewis Tunstall, Abhishek Thakur, Pegah Maham, C Jess Riedel, Emmie Hine, Carolyn Ashurst, Paul Sedille, Alexis Carlier, et al. Raft: A real-world few-shot text classification benchmark. *arXiv preprint arXiv:2109.14076*, 2021.

[2] Anonymous. Optimizing inference-time reasoning in LLMs via retrieval-augmented reflection, 2025. URL https://openreview.net/forum?id=ElYRG3pJcv.

[3] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR, 2022.

[4] Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, page 173–180, USA, 2005. Association for Computational Linguistics. doi: 10.3115/1219840.1219862. URL https://doi.org/10.3115/1219840.1219862.

[5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

[6] 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.

[7] Xidong Feng, Ziyu Wan, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. Alphazero-like tree-search can guide large language model decoding and training, 2024. URL https://arxiv.org/abs/2309.17179.

[8] Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*, 2023.

[9] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*, 2023.

[10] Neel Guha, Julian Nyarko, Daniel Ho, Christopher Ré, Adam Chilton, Alex Chohlas-Wood, Austin Peters, Brandon Waldon, Daniel Rockmore, Diego Zambrano, et al. Legalbench: A collaboratively built benchmark for measuring legal reasoning in large language models. *Advances in Neural Information Processing Systems*, 36:44123–44279, 2023.

[11] Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model, 2023. URL https://arxiv.org/abs/2305.14992.

[12] Namgyu Ho, Laura Schmid, and Se-Young Yun. Large language models are reasoning teachers, 2023. URL https://arxiv.org/abs/2212.10071.

[13] Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. Training language models to self-correct via reinforcement learning, 2024. URL https://arxiv.org/abs/2409.12917.

[14] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

[15] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large language model society, 2023. URL https://arxiv.org/abs/2303.17760.

[16] Qintong Li, Leyang Cui, Xueliang Zhao, Lingpeng Kong, and Wei Bi. Gsm-plus: A comprehensive benchmark for evaluating the robustness of llms as mathematical problem solvers. *arXiv preprint arXiv:2402.19255*, 2024.

[17] Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models, 2025. URL https://arxiv.org/abs/2501.05366.

[18] Kevin Lin, Charlie Snell, Yu Wang, Charles Packer, Sarah Wooders, Ion Stoica, and Joseph E Gonzalez. Sleep-time compute: Beyond inference scaling at test-time. *arXiv preprint arXiv:2504.13171*, 2025.

[19] Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Tianjun Zhang, Erran Li, Raluca Ada Popa, and Ion Stoica. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl. https://pretty-radio-b75.notion.site/DeepScaleR-Surpassing-O1-Preview-with-a-1-5B-Model-by-Scaling-RL-19681902c1468005bed8ca303013a4 2025. Notion Blog.

[20] Jingyuan Ma, Damai Dai, Lei Sha, and Zhifang Sui. Large language models are unconscious of unreasonability in math problems. *arXiv preprint arXiv:2403.19346*, 2024.

[21] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative Refinement with Self-Feedback. *arXiv e-prints*, art. arXiv:2303.17651, March 2023. doi: 10.48550/arXiv.2303.17651.

[22] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594, 2023.

[23] Mathematical Association of America. 2025 American Invitational Mathematics Examination (AIME) I: Problems and Solutions, feb 2025. URL https://artofproblemsolving.com/wiki/index.php/2025_AIME_I. Art of Problem Solving Wiki entry.

[24] Rakesh R. Menon, Sayan Ghosh, and Shashank Srivastava. Clues a benchmark for learning classifiers using natural language explanations. 2022.

[25] Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*, 2024.

[26] Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show your work: Scratchpads for intermediate computation with language models, 2021. URL https://arxiv.org/abs/2112.00114.

[27] Jiayi Pan, Xiuyu Li, Long Lian, Charlie Snell, Yifei Zhou, Adam Yala, Trevor Darrell, Kurt Keutzer, and Alane Suhr. Learning adaptive parallel reasoning with language models. *arXiv preprint arXiv:2504.15466*, 2025.

[28] Richard Yuanzhe Pang, Weizhe Yuan, Kyunghyun Cho, He He, Sainbayar Sukhbaatar, and Jason Weston. Iterative reasoning preference optimization. *arXiv preprint arXiv:2404.19733*, 2024.

[29] Ivo Petrov, Jasper Dekoninck, Lyuben Baltadzhiev, Maria Drencheva, Kristian Minchev, Mislav Balunović, Nikola Jovanović, and Martin Vechev. Proof or bluff? evaluating llms on 2025 usa math olympiad. *arXiv preprint arXiv:2503.21934*, 2025.

[30] Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with" gradient descent" and beam search. *arXiv preprint arXiv:2305.03495*, 2023.

[31] Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. Recursive Introspection: Teaching Language Model Agents How to Self-Improve. *arXiv e-prints*, art. arXiv:2407.18219, July 2024. doi: 10.48550/arXiv.2407.18219.

[32] Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. Recursive introspection: Teaching language model agents how to self-improve. *arXiv preprint arXiv:2407.18219*, 2024.

[33] Yuxiao Qu, Matthew YR Yang, Amrith Setlur, Lewis Tunstall, Edward Emanuel Beeching, Ruslan Salakhutdinov, and Aviral Kumar. Optimizing test-time compute via meta reinforcement fine-tuning. *arXiv preprint arXiv:2503.07572*, 2025.

[34] Qwen Team. Qwen3 technical report. Technical report, Qwen, May 2025. URL https://huggingface.co/Qwen. Available at: https://huggingface.co/Qwen, https://modelscope.cn/organization/qwen, https://github.com/QwenLM/Qwen3.

[35] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741, 2023.

[36] Marlin B. Schäfer, Frank Ohme, and Alexander H. Nitz. Detection of gravitational-wave signals from binary neutron star mergers using machine learning. *Physical Review D*, 102(6), September 2020. ISSN 2470-0029. doi: 10.1103/physrevd.102.063015. URL http://dx.doi.org/10.1103/PhysRevD.102.063015.

[37] Amrith Setlur, Matthew YR Yang, Charlie Victor Snell, Jeremiah Greer, Ian Wu, Virginia Smith, Max Simchowitz, and Aviral Kumar. e3: Learning to explore enables extrapolation of test-time compute for llms. In *ICML 2025 Workshop on Long-Context Foundation Models*, 2025.

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

[39] Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H. Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 31210–31227. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/shi23a.html.

[40] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

[41] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters. *arXiv e-prints*, art. arXiv:2408.03314, August 2024. doi: 10.48550/arXiv.2408.03314.

[42] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

[43] Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. *arXiv preprint arXiv:2504.07952*, 2025.

[44] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. *arXiv preprint arXiv:2212.10509*, 2022.

[45] Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback, 2022. URL https://arxiv.org/abs/2211.14275.

[46] Prakhar Verma, Sukruta Prakash Midigeshi, Gaurav Sinha, Arno Solin, Nagarajan Natarajan, and Amit Sharma. Plan$\times$RAG: Planning-guided Retrieval Augmented Generation. *arXiv e-prints*, art. arXiv:2410.20753, October 2024. doi: 10.48550/arXiv.2410.20753.

[47] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023. URL https://arxiv.org/abs/2203.11171.

[48] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*, 2023.

[49] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. URL https://arxiv.org/abs/2305.10601.

[50] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.

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

[52] Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Chuanqi Tan, and Chang Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.

[53] Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *arXiv preprint arXiv:2504.13837*, 2025.

[54] Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639(8055):609–616, 2025.

[55] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

[56] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. Star: Bootstrapping reasoning with reasoning, 2022. URL https://arxiv.org/abs/2203.14465.

[57] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642, 2024.

[58] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations*, 2022.

Appendices

A. Experimental Details

A.1. Pseudocode for *RLAD*

Algorithm 1 Joint RL Training of π_θ^{abs} and π_θ^{sol}

```
Require: Policies $\pi_\theta^{\text{abs}}(\mathbf{z} \mid \mathbf{x})$, $\pi_\theta^{\text{sol}}(\tilde{\mathbf{y}} \mid \mathbf{x}, \mathbf{z})$ Datasets $\mathcal{D}_{\pi_\theta^{\text{abs}}}$, $\mathcal{D}_{\pi_\theta^{\text{sol}}}$
Require: Learning rates $\alpha_{\pi_\theta^{\text{abs}}}, \alpha_{\pi_\theta^{\text{sol}}}$; Batch sizes $N, M$; Epochs $E$
 1: Initialize $\pi_\theta^{\text{abs}}, \pi_\theta^{\text{sol}}$
 2: for $e = 1$ to $E$ do                                          ▷ Update abstraction policy
 3:     for $\{\mathbf{x}_i\}_{i=1}^N \sim \mathcal{D}_{\pi_\theta^{\text{abs}}}$ do
 4:         $\mathbf{z}_i \sim \pi_\theta^{\text{abs}}(\cdot|\mathbf{x}_i)$
 5:         $r_i \leftarrow r_{\pi_\theta^{\text{sol}}}(\mathbf{x}_i, \mathbf{z}_i)$
 6:         $\pi_\theta^{\text{abs}} \leftarrow \pi_\theta^{\text{abs}} - \alpha_{\pi_\theta^{\text{abs}}} \nabla_{\pi_\theta^{\text{abs}}} \mathcal{L}_{\text{STAR/RPO}}(\pi_\theta^{\text{abs}}; \mathbf{x}_i, \mathbf{z}_i, r_i)$
 7:     end for                                                    ▷ Update solution policy
 8:     for $\{\mathbf{x}_j\}_{j=1}^M \sim \mathcal{D}_{\pi_\theta^{\text{sol}}}$ do
 9:         $\mathbf{z}_j \sim \pi_\theta^{\text{abs}}(\cdot|\mathbf{x}_j)$,    $\tilde{\mathbf{y}}_j \sim \pi_\theta^{\text{sol}}(\cdot|\mathbf{x}_j, \mathbf{z}_j)$
10:         $r_j \leftarrow r(\mathbf{x}_j, \mathbf{z}_j, \tilde{\mathbf{y}}_j)$
11:         $\pi_\theta^{\text{sol}} \leftarrow \pi_\theta^{\text{sol}} - \alpha_{\pi_\theta^{\text{sol}}} \nabla_{\pi_\theta^{\text{sol}}} \mathcal{L}_{\text{GRPO}}(\pi_\theta^{\text{sol}}; \mathbf{x}_j, \mathbf{z}_j, \tilde{\mathbf{y}}_j, r_j)$
12:     end for
13: end for
```

A.2. Hyperparameters

B. Additional Experimental Results

B.1. Abstraction on Diverse Text Classification

Hyperparameter	Value
algorithm	DaPO [51]
training steps	100
epochs	10
train batch size	128
max prompt length	3072
max response length	16384
max extrapolation length	32768
learning rate	1e-6
PPO mini batch size	64
PPO micro batch size	64
clip ratio (low / high)	0.2 / 0.5
entropy coefficient	0.001
KL loss coefficient	0.001
KL loss type	low_var_kl
sampling temperature (train / val)	0.6 / 0.6
samples per prompt (train / val)	16 / 8
max batched tokens	32768

Table 2: Key training hyperparameters used in ***RLAD***.

B.2. *RLAD*'s w/ abs performance on AIME 2024 and HMMT 2025

In this section, we evaluate the performance of the base model (Qwen-3-1.7B), GRPO-enhanced model, and our proposed method ***RLAD*** on two math reasoning benchmarks: AIME 2024 and HMMT 2025. As shown in Table 4, our method achieves the best performance across both datasets.

It is important to note that ***RLAD*** is trained using access to abstractions, yet it also generalizes better even when evaluated without abstraction. This suggests that ***RLAD*** does not merely overfit to the abstraction format but instead learns to effectively leverage high-level procedural guidance, leading to better generalization on challenging reasoning benchmarks.

B.3. Design Choice Ablations

In this section, we run run some ablation experiments to better understand the contributions of individual components of ***RLAD*** in attaining good performance. In particular, we are interested in understanding the role of **(a)** inclusion of prompts that are not annotated with an abstraction, **(b)** reward masking on these prompts if they are included, and **(c)** training via a curriculum approach, following the protocol in Setlur et al. [37].

We present our results in Table 5. The first experiment we run focuses on understanding how important it is to include a small fraction of prompts with no abstractions in training of π_θ^{sol}. Observe in

Curriculum training refers to a staged training process where the model first learns from simpler problems and gradually transitions to harder examples. We borrow this idea from concurrent work Setlur et al. [37] (which we also attach in the supplementary material) as it showed that this approach led to better

Dataset	**Zero-shot**	**Best Abstraction**	**Average Abstraction**
UCI Dry Bean	0.00	0.65	0.51
Wikipedia Proteinogenic Acid	0.22	0.78	0.58
UCI Student Performance	0.25	0.45	0.28
UCI Website Phishing	0.25	0.25	0.22
UCI Teaching Assistant Evaluation	0.25	0.45	0.33
UCI Contraceptive Method Choice	0.30	0.60	0.43
UCI Vertebral Column	0.30	0.75	0.64
UCI Shill Bidding	0.30	1.00	0.95
Kaggle Job Change	0.30	0.85	0.83
UCI Caesarian Section	0.38	0.75	0.64
Wikipedia Coin Face Value	0.40	1.00	0.88
UCI Wine	0.40	0.95	0.85
UCI Tic-Tac-Toe Endgame	0.40	0.80	0.42
Kaggle Campus Placement	0.40	0.85	0.72
Wikipedia Driving Championship Points	0.40	1.00	0.74
UCI Mammographic Mass	0.45	0.90	0.82
UCI Banknote Authentication	0.45	1.00	0.78
Kaggle Engineering Placement	0.50	0.85	0.79
RAFT One Stop English	0.50	0.40	0.36
LegalBench Function of Decision Section	0.54	0.72	0.61
Kaggle Entrepreneur Competency	0.55	0.65	0.58
UCI Indian Liver Patient	0.55	0.80	0.68
LegalBench International Citizenship Questions	0.56	0.74	0.63
LegalBench Abercrombie	0.56	0.80	0.67
Wikipedia Color Luminance	0.60	1.00	1.00
RAFT Twitter Hate Speech	0.60	0.90	0.76
Wikipedia Award Nomination Result	0.64	1.00	0.76
UCI Car Evaluation	0.65	0.75	0.64
Kaggle Water Potability	0.65	0.50	0.38
Kaggle Travel Insurance	0.65	0.70	0.59
UCI Internet Firewall	0.70	1.00	0.97
RAFT ADE Corpus	0.70	1.00	0.89
UCI Somerville Happiness Survey	0.70	0.80	0.68
UCI Mushroom	0.75	1.00	0.95
UCI Occupancy Detection	0.80	1.00	0.92
Kaggle Stroke Prediction	0.85	0.90	0.90
LegalBench Corporate Lobbying	0.88	0.94	0.88
Average	0.50	0.80	0.68

Table 3: Evaluation of abstractions on diverse collection of 37 domains. We sampled 10 abstractions by prompting `o4-mini`, and measure test set accuracy while prompting `GPT-4o-mini` with each abstraction. We report both the average performance of the 10 abstractions and the best abstraction. **We find that the average and best abstractions outperform standard prompting by 18.0% and 30.0% on average, respectively.**

Approach	AIME 2024	HMMT 2025
Qwen-3-1.7B	48.54	22.50
+ GRPO	44.17	23.13
+ ***RLAD***	51.46	23.75

Table 4: ***RLAD*'s w/ abs performance on AIME 2024 and HMMT 2025.**

performance without any abstractions, for just direct math problem-solving. In contrast, non-curriculum training mixes problems of all difficulties throughout training. As shown in the table, when training with abstractions as well, curriculum training improves both average and best-case abstraction-conditioned performance (0.41 and 0.48 vs. 0.38 and 0.43).

Second, we explore whether **including no-abstraction prompts** during training helps the solution-generator pay attention to the abstractions. We find that including these abstractions minorly improves the average performance from 0.37 to 0.38, in isolation when curriculum is not utilized.

Lastly, we study the effect of masking the problem-solving reward on no-abstraction prompts. We apply **reward masking** to prevent updates that might cause the solution-generator to ignore abstractions altogether. Specifically, we zero out the advantage (i.e., no policy reward) for completions from no-abstraction prompts, while retaining the KL penalty to maintain regularization. This design discourages the model from over-optimizing on no-abstraction examples, which could otherwise lead it to bypass abstractions entirely, a shortcut that may yield improved performance on the training set but hinders generalization to test problems when abstractions are provided. Empirically, we find reward masking is helpful.

Approach	**Design Choice**			**AIME 2025**	
	curriculum training	including no-abstraction prompt	reward masking	w/ abs (avg)	w/ abs (best)
variant 1	✗	✓	✗	36.51	42.29
variant 2	✗	✗	-	37.08	42.50
variant 3	✗	✓	✓	37.50	43.33
RLAD	✓	✓	✓	**42.45**	**48.33**

Table 5: **Ablation of Design Choices in *RLAD*.** We isolate the effects of curriculum training, no-abstraction inclusion, and reward masking. The full method achieves the strongest performance under abstraction-conditioned evaluation.

C. Qualitative Examples of Math Reasoning Abstractions

C.1. Prompt for Abstraction Classification

We prompt `GPT-4o-mini` with the following classifier prompt to classify each abstraction into one of four categories.

Post-hoc abstraction classifier prompt

```
You are a abstraction classifier. You will be given a problem-solving heuristic or abstraction
used for mathematical reasoning. Your task is to classify it into exactly one of the following
mutually exclusive categories, based on the primary cognitive function the heuristic serves.

(A) Caution alert: any abstraction that warns the reader to double-check a specific aspect
of their solution or to not take a specific approach to the problem.
(B) Productive launchpoint: an early move or framing that opens up high-potential trajectories.
Examples include clever reformulations or symmetries.
(C) Blind-follow trajectory: a description of a repeatable, sequential path that can be
reliably followed to solve the problem. Examples include plug-and-play formulas that can be
followed blindly, without insight. Do not choose this is further reasoning is required to
solve the problem.
(D) Structural shortcut: a conceptual move that collapses multiple graph paths into a single
jump via insight or abstraction. This can include introducing invariants.
(E) Other: a abstraction that does not fit into the above categories.
Give a 1-2 sentence explanation for your classification, and end your answer with exactly one
of: (A), (B), (C), (D), or (E).

_
abstraction:
{abstraction}
```

C.2. Example for Each Abstraction Category

Below, we show examples of abstractions classified into the four categories above.

Examples of (A) Caution alert

```
<description>Always record forbidden values from denominators before and after manipulation.
After solving the polynomial, discard any roots that make a denominator zero or that do not
satisfy the original equation, to avoid extraneous solutions.</description>
<example>In the equation (x+2)/(2x−1) = x−3, 2x−1 cannot be zero (so x is not ½). If solving
yields x=½ or any root that makes any denominator zero, reject it. Then verify the accepted
roots in the original equation.</example>

<description>Keep units consistent when moving between area and length or summing lengths.
After extracting a length from an area (via square root), ensure subsequent arithmetic stays
in the same unit to avoid scaling errors. </description>
<example>If a square's area is 10000 cm², its side is sqrt(10000) = 100 cm. To express in
meters, convert 100 cm to 1 m. All later distances computed with that side length must be in
meters to remain consistent.</example>
```

Examples of (B) Productive launchpoint

<description>Translate comparative statements into algebraic equations using the chosen variables. Phrases like “twice as many” or “one less than” correspond to multiplication or addition/subtraction expressions. This step captures the core relationship in a solvable form.</description>
<example>If the problem states “Group A has twice as many as Group B,” write the equation x = 2y. For “Group B has three fewer than Group C,” you would write y = z - 3.</example>

<description>Select one variable as a parameter (often setting it to 1 or keeping it symbolic) to express all other variables in terms of it. This reduces the number of independent symbols and streamlines substitutions.</description>
<example>Given p/q = 3 and r/q = 2, choose q as the base variable. Write p = 3q and r = 2q, so all expressions involving p and r can be handled through q alone.</example>

Examples of (C) Blind-follow trajectory

<description>Logarithms offer a streamlined way to compute floor-based digit counts: for y>0, the number of integer digits is floor(log10 y) + 1. Use this to handle arbitrary exponents without juggling large powers explicitly.</description>
<example>To count digits of y = x^7, compute d = floor(7 * log10 x) + 1. If x=2.5, then d = floor(7 * log10(2.5))+1 = 2+1 = 3 digits.</example>

<description>The mean of a set equals its total sum divided by its number of elements. Use this to move between sums and averages when counts or totals are known. It works because “average” is defined as that ratio.</description>
<example>Suppose a subset has k items with mean m. Then its total sum is S = k·m. Conversely, if you know the sum S and the count k, the mean is m = S/k. For instance, if 5 items average to 10, their total is 5×10 = 50, and if you later learn the total is 60 for 6 items, the new mean becomes 60/6 = 10.</example>

Examples of (D) Structural shortcut

<description>When the same distance appears in multiple geometric roles (e.g., as radius to a vertex and to a tangen t point), express it in different algebraic forms and equate them. Solving the resulting equation produces the unknown variable, which then gives the desired length.</description>
<example>If r is both the distance from O to a vertex (r = sqrt[x^2 + $(L/2)^2$]) and the distance from O to the tangent point (r = f(x)), set sqrt[x^2 + $(L/2)^2$] = f(x). Solving this equation for x and back-substituting determines r explicitly, closing the geometric problem with an algebraic solution.</example>

<description>Use the perimeter constraint a+b+c=P to eliminate one variable, e.g. set c=P-a-b, reducing the problem to two degrees of freedom. This simplification turns the three-variable Heron expression into a function of a and b alone, facilitating analysis or enumeration.</description>
<example>For a target perimeter P=10, one writes c=10-a-b. Substituting into Heron’s formula yields A(a,b)=sqrt[5 * (5-a) * (5-b) * (a+b-5)], which is now a two-variable function to study instead of three.</example>