PATCHING GAPS IN LLM REASONING WITH INTERVENTIONAL TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning (RL) training of large language models (LLMs) is limited by the policy's ability to generate rollouts with non-zero rewards: without such rewards, the policy is not updated and learning is stalled on hard problems, which are problems that the policy consistently fails to sample any correct rollouts for. We find that many hard problems remain unsolved due to the repeated generation of incorrect intermediate steps in a long reasoning trace; identifying and fixing these requires performing better *credit assignment*. But existing approaches for credit assignment are either impractical or impose a substantial data-writing burden on oracles (*e.g.*, humans). In this paper, we introduce **Interventional Training** (InT), a framework that leverages single-step oracle interventions to improve LLM reasoning. Given a reasoning attempt and ground-truth answer, the oracle detects and then provides language feedback on a single intermediate reasoning step, which is much cheaper than obtaining a full reasoning trace. **InT** then *patches* the LLM by running supervised fine-tuning on the on-policy rollout up to the error, followed by the correction from the oracle. RL on this patched model now generates counterfactual traces and with merely $\approx$100 interventions from the oracle, **InT** solves 16% more hard test problems that were previously unsolved (only zero rewards) and also improves performance across multiple standard evals.

1 INTRODUCTION

Post-training large language models (LLMs) with reinforcement learning (RL) has proven to be a highly effective strategy for improving their reasoning capabilities. In a typical RL recipe, we update the current policy on a given problem by first sampling from the policy multiple rollouts conditioned on the problem, and then positively reinforcing the policy on correct rollouts while down-weighting the likelihood of incorrect ones (DeepSeek-AI et al., 2025). However, if no rollout ends up being correct, all traces receive zero reward. This poses a barrier to scaling RL to harder problems where the policy fails to acquire any non-zero reward. When faced with such a scenario, a practitioner could modify pre-training or mid-training run typically before RL to retrain a base model capable of attaining reward, but the effect of these procedures on subsequent RL is still not well understood.

What do the reasoning traces on such problems unsolved during training look like? While several such problems lie far outside the scope of the pre-trained model's capabilities, we find that a substantial (**near 30%**) chunk of problems that we cannot train on today are those where the model makes an *execution* mistake in its trace, which derails subsequent reasoning, and results in an incorrect answer even with several parallel attempts. Such mistakes are more likely to occur at long lengths that are composed of many steps. This raises a key question: *how can we still train models on hard problems on which they fail to sample correct traces largely due to execution errors?*

In principle, addressing this challenge requires addressing the problem of *credit assignment*: if we could pinpoint the intermediate step at which a reasoning trace goes astray, and feed in a more accurate alternate step, we could train the model to correct its course from that point onward. However, credit assignment in long reasoning traces is notoriously difficult. One way of identifying an incorrect step is to sample multiple rollouts from each prefix in the long trace, and find the step that drops the probability of success, but this can be a prohibitively high sampling cost for long length reasoning models (Kazemnejad et al., 2024). Another option is to train process reward models (PRMs) (Setlur et al., 2024b; Lightman et al., 2023b) on oracle data and use them to directly evaluate intermediate reasoning steps. Training accurate PRMs often is quite sample inefficient (Luo et al., 2024), and can

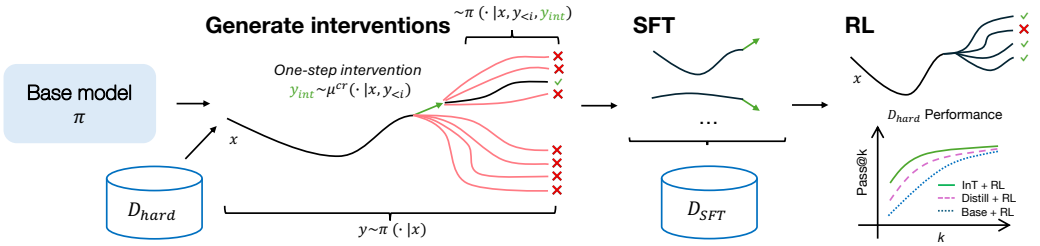


Figure 1: ***Intervention training for patching LLMs during RL.*** Instead of relying on full expert reasoning traces or attempting to find rare correct rollouts during RL, InT introduces *single-step*, oracle interventions that "patch" incorrect intermediate steps in model-generated reasoning traces. Conditioned on these localized corrections, the model can generate counterfactual continuations that succeed where the original failed. We then distill these interventions into the model via supervised fine-tuning before resuming RL, enabling effective credit assignment and continued progress even on problems that were previously unsolvable with standard RL.

be particularly hard for long reasoning traces (Kim et al., 2025). As a result, existing approaches to credit assignment remain impractical for long reasoning traces. That said, we can improve over these approaches if we assume access to an oracle (*e.g.*, a human) that provides some kind of intermediate feedback on *just* the incorrect step, and if the number of calls to such an oracle is forced to be limited.

Motivated by this, we introduce **Interventional Training (InT)** (Figure 1), an approach for effective credit assignment in long reasoning traces from a model that has plateaued during RL. Here, instead of requesting complete and long expert reasoning traces, InT solicits *single-step interventions* from an oracle (*e.g.*, a human, another LLM, or a specialized verifier). Given a model-generated reasoning attempt and the ground-truth answer, the oracle replaces exactly one critical and incorrect intermediate step with a corrected version (*e.g.*, fixing an incorrect approximation in a long math answer, see Fig. 2). Conditioned on this intervention, the model can then generate alternate counterfactual traces that may succeed where the original failed. In this way, interventions provide a low-cost mechanism to discover correct reasoning traces. Next, **InT** internalizes these interventions into the model weights by running supervised fine-tuning on the interventions, and then continues the RL run that had previously plateaued. By "patching" the model on single-step interventions in this manner, **InT** makes it possible for the model to attain non-zero reward signals even on otherwise unsolvable problems, enabling effective training on problems that are inaccessible with RL.

If we are using the oracle for localized credit assignment, it is reasonable to ask why can't we get the oracle to give us the entire reasoning trace on the hard problem, that we can now clone. Requesting localized interventions requires far less data-writing effort than producing full ground-truth solutions in the model's output format, and can flexibly incorporate feedback from humans, specialized tools that an LLM interacts with, or other oracle models. Moreover, we find that even if we had full length reasoning traces, since they typically lie outside our base model's distribution, cloning them leads to memorization and completely distorts the base model's next-token distribution.

By running RL on a set of 64 training problems, InT achieves a 6.09% (3.12% to 9.03%) gain in pass@64 and solves 14% more problems on a challenging held out test set. In contrast, simply distilling the oracle or running SFT on oracle data achieves only a 3.51% gain in pass@64. On standard reasoning benchmarks, **InT** leads to an average improvement of 1.92% across 7 standard math reasoning benchmarks, showing that patching the model with **InT** does not degrade existing model capabilities. These results show that **InT** offers a simple, deterministic way to patch model behavior, improving performance on new problems while preserving or improving existing reasoning capabilities.

2 Related Work

Credit assignment in LLM reasoning. The effectiveness of long length RL with outcome rewards (DeepSeek-AI et al., 2025) is often crippled by credit assignment: it is unclear which intermediate steps in a long response should be "credited" for the outcome reward. While one might surmise that sampling enough rollouts should address this problem, note that the difficulty of credit assignment also greatly increases with the horizon (Setlur et al., 2025a). While most methods reward each token with the outcome level advantage (Yu et al., 2025), others use process reward models (PRMs) to assign dense token or step-level rewards (Lightman et al., 2023a; Wang et al., 2024; Qu et al., 2025b) that can reinforce correct steps and promote unlearning of incorrect ones. Although PRMs may

improve RL compute efficiency (Setlur et al., 2024b;a), we often require costly rollouts (Luo et al., 2024; Kazemnejad et al., 2024) to reliably estimate these reward signals. We instead leverage oracles to detect individual mistakes, bypassing the compute required to train entirely new PRMs or to perform full credit assignment on the full reasoning trace; even when the oracle is itself a reasoning model, it is given the ground-truth response and only needs to perform a comparative analysis to identify an intervention. While our method shares the general idea of using generative models as verifiers (Zhang et al., 2024; Liu et al., 2025; Kim et al., 2025; Khalifa et al., 2025), it is distinct in that we task the oracle with explicitly pointing out the location of a single, key mistake, rather than verifying every step and judging the solution. Finally, we use the outputs of the oracle to improve RL training rather than inference-time methods (e.g., best-of-N search), which prior works focus on.

Learning from natural language feedback. Another related line of work explores utilizing natural language feedback to improve RL training. Such works typically leverage the feedback to refine rollouts that are then used to improve the policy. Chen et al. (2024) combine human feedback and a separate refinement model to improve policy-generated outputs that are then distilled back into the policy via SFT. Yan et al. (2025) use a teacher model to generate correction trajectories for off-policy RL, while Zhang et al. (2025b) conduct critique-guided self-refinement to generate correction trajectories, again for use in off-policy RL. Unlike these works, our work considers generating short, targeted natural language feedback to correct individual steps within what are otherwise purely on-policy trajectories. As we discuss later on, this allows us to achieve substantial improvements without making significant changes to the standard RL training recipe.

Intervention training outside of LLMs. Applying interventions at exact points of failure has been explored in domains outside of LLM training, for example, in dexterous manipulation (Hu et al., 2025) and imitation learning (Kelly et al., 2019). The class of intervention methods across these different domains generally shows improved data efficiency and faster convergence of reward curves compared against naive behavior cloning method due to superior credit assignment. In our work, we examine whether applying such ideas to language model RL training can reap similar benefits.

3 Preliminaries and Problem Statement

Reasoning LLMs still struggle to solve certain training problems. One might expect that training on hard problems during RL helps improve the model's success rate on analogous hard problems at test time, which makes it important to be able to derive learning signal on some such problems. To better understand the failure modes of LLMs on difficult reasoning questions, we inspect rollouts from the reasoning LLM Qwen3-4B-Instruct and find that a substantial number of model failures stem from a failure of *execution* and an inability to ***recover*** from failure: even though the model often takes a correct high-level approach, it gets derailed during an intermediate step when attempting to execute on this high-level approach. We demonstrate some examples of these failures in Figure 2 and Appendix B. As shown in Figure 4 and as noted by Sinha et al. (2025), failure in execution (vs. high-level strategy) is a common occurrence in long reasoning traces, especially later in the trajectory. Our objective is to improve model performance on challenging reasoning tasks in which such execution failures result in scarce positive rewards for RL.

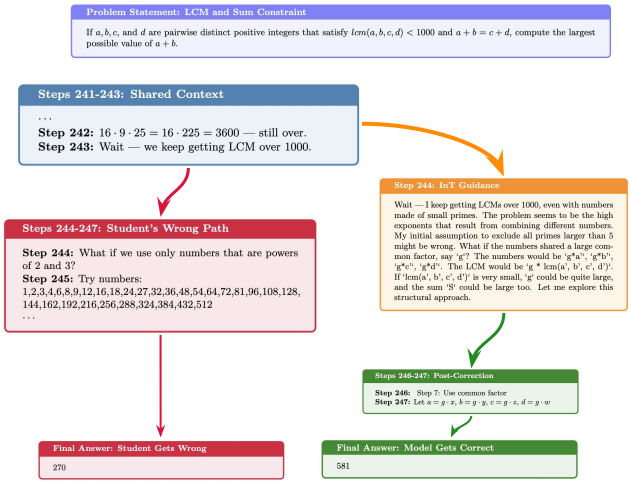


Figure 2: ***Example execution error of Qwen3-4B-Instruct.*** Note that the model deduces an incorrect conclusion from going over 1000 and continues trying small primes, which an oracle intervention (via our approach, **InT**) addresses. We find that 40.6% stem from execution errors. This showcases the potential of fixing execution errors directly for RL, without needing to rerun pre-training or mid-training.

Notation. To build our approach, we first define some relevant notation. LLM training for reasoning typically involves an LLM π, a binary reward $r(\mathbf{x}, \mathbf{y}) \in \{0, 1\}$ for inputs $\mathbf{x} \sim \rho$, and model outputs $\mathbf{y} \sim \pi(\cdot \mid \mathbf{x})$. Common paradigms for training LLMs include supervised fine-tuning (SFT) and online policy gradient methods (henceforth referred to as RL). SFT fine-tunes a model by maximizing likelihood on a dataset (ideally consisting of correct traces $\mathbf{y}$ for problems $\mathbf{x}$). RL, in contrast, samples candidate rollouts from its own policy distribution (i.e., on-policy) and trains to maximize the reward of these rollouts. In its simplest form, RL training for LLMs uses the policy gradient:

$$\theta' \leftarrow \theta + \alpha \cdot \mathbb{E}_{\mathbf{y} \sim \tilde{\pi}(\cdot|\mathbf{x})} \Big[r(\mathbf{x}, \mathbf{y}) \cdot \nabla_\theta \log \pi_\theta(\mathbf{y} \mid \mathbf{x}) \Big], \tag{1}$$

where $\tilde{\pi}$ is the policy used to generate samples that go in for training. In RL methods such as GRPO, $\tilde{\pi} = \pi_{\text{old}}$ is a periodically updated copy of π, and the reward $r(\mathbf{x}, \mathbf{y})$ is normalized by subtracting a baseline to form the advantage, which serves as the multipler to $\nabla_\pi \log \pi$ instead of r:

$$A_i(\mathbf{x}, \mathbf{y}_i) = r(\mathbf{x}, \mathbf{y}_i) - {}^1\!/_n \sum_i \bar{r}(\mathbf{x}, \mathbf{y}_i).$$

Credit assignment. As discussed above, a substantial fraction of errors made by models on the training distribution correspond to execution errors. This naturally means that while some tokens/steps in a model response are on the right track towards solving the problem at hand, a different subset of tokens derails the model onto not attaining the right final answer. If we could identify the identity of these tokens/steps and the extent of their influence (i.e., "credit") in affecting the correctness of the final answer, then this problem could be solved as long as the model is able to find alternative steps that do actually succeed. This process is called ***credit assignment*** (Setlur et al., 2024a). RL algorithms rely on self-generated rollouts to guide the process of credit assignment.

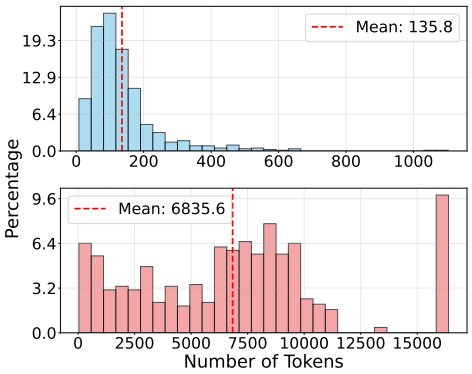


Figure 3: ***Interventions are short.*** Top: Lengths of interventions typically span under 200 tokens, while full solutions are much longer (bottom).

However, performing credit assignment solely from outcome-level rewards is highly challenging. When the advantage is positive, every token probability $\pi(\mathbf{y}_i \mid \mathbf{x}, \mathbf{y}_{<i})$ across the sequence is equally reinforced; when it is negative, all tokens are equally discouraged. Over long reasoning traces, such uniform updates are not effective at performing credit assignment: tokens that played no role in reaching the solution may still be upweighted ("higher credit"), while tokens that were correct but followed by later mistakes may be suppressed ("lower credit"). Even when we know which tokens failed at a particular problem, we need to search for other steps to pursue to get to a correct answer. On hard problems, this noisecan overwhelm the signal, hindering the model from making progress. Correctly assigning credit is therefore crucial.

Problem Statement

A significant fraction (40.6%) of mistakes made by LLMs on hard problems stem from execution errors. We aim to address these errors by performing better credit assignment (reinforcing correct steps, unlearning incorrect steps, and finding alternative completions).

4 INT: INTERVENTIONAL TRAINING FOR CREDIT ASSIGNMENT

Given a problem, when a reasoning LLM repeatedly falters, one of the most direct approaches of "patching" it on this problem is to behavior clone oracle (*e.g.*, human) generated off-policy solutions to this problem. However, oracle full-length solutions to problems are highly off-policy with respect to the policy itself. Prior works argue (Zhang et al., 2025a; Setlur et al., 2025a; Kang et al., 2024), and we further show, that distilling such data interferes with the model's reasoning capabilities, especially on out-of-distribution problems. *Is there a simpler, more data-efficient, and effective way to leverage oracles for improving credit assignment?* In this section, we develop our approach for doing so.

Our key idea, **InT** (interventional training), is to structure oracle feedback as single-step corrective interventions on the base model's (model plateaued during RL training) output reasoning trace: each intervention provides short natural language feedback at an intermediate step proposed by the base model, guiding the subsequent completion, particularly in ways different from simply continuing the

rollout (see illustration in Fig 1). These interventions are easy to obtain and avoid the type mismatch problem described above since they are simply $\sim$100 tokens long (Fig. 3). Our experiments show that interventions provide asample-efficient mechanism for incorporating oracle feedback to do credit assignment. With $\leq$**80 interventions**, we improve accuracy on tasks where the base model previously obtained no reward, and also advance state-of-the-art across various reasoning benchmarks.

4.1 INT INTERACTION PROTOCOL

To instantiate **InT**, we collect data that pinpoints where a model's reasoning trace first goes wrong and provides a corrective step at that location. In principle, the corrective step of the intervention comes from an oracle, which we simulate using a larger, proprietary reasoning model denoted by μ^{cr}. This oracle does not need to be capable of solving the problem by itself, but it should be capable of comparing the base LLM π's solution trace $\mathbf{y}$, with the ground truth solution $\mathbf{y}^*$ on problem $\mathbf{x}$. Assuming that $\mathbf{y}$ can be segmented into k_0 reasoning steps (based on simple keywords like "wait", "maybe", or "\n \n" based segmentation), μ^{cr} identifies the first step $i \in [k_0]$ where an error occurs that itself is not corrected by the base LLM, and outputs a corrective intervention $\mathbf{y}_{\text{int},i}$. This intervention may either prevent the error from arising at step i or repair it immediately afterward. This approach requires the oracle $\mu^{\text{cr}}(\cdot|\mathbf{y}^*, \mathbf{y}, \mathbf{x})$ to simply perform step-by-step comparison operations until the *first* index i where the responses are incorrect, and then return a short intervention.

An example of this interaction protocol between an LLM and the oracle is shown in Figure 2. We implement this idea with small 1.7B and 4B reasoning LLMs as our base models, and Gemini 2.5 Pro as the oracle, and note that most interventions occur mid-trace rather than at the beginning. The position of interventions also demonstrates that **InT** is fundamentally different from "hint-conditioning" approaches (Qu et al., 2025a), which prepend a "hint" before reasoning begins.

Empirically we find that continuing rollouts from just before the identified error location $\pi(\cdot \mid \mathbf{x}, \mathbf{y}_{<i})$ significantly improves the likelihood of reaching a correct solution compared to rollouts from the start of the problem $\mathbf{x}$. Appending the corrective step from the oracle to the base model-generated steps before the error and then continuing to sample from the base model $\pi(\cdot \mid \mathbf{x}, \mathbf{y}_{<i}, \mathbf{y}_{\text{int},i})$ yields further gains, confirming that both localizing mistakes and injecting short corrective interventions are individually useful. Some concrete examples of interventions are shown in Figure 2, with additional cases in Appendix B. We report the aggregate performance and number of problems solved by conditioning on interventions in Table 1.

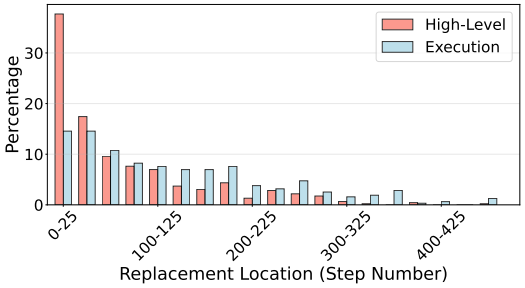


Figure 4: ***Histogram of intervention locations by error category.*** High-Level errors occur early on in the trace but execution errors persist throughout model rollouts.

4.2 PATCHING THE BASE LLM WITH ORACLE INTERVENTIONS

Having established the intervention protocol, the next step is to "patch" the gaps in the base LLM's reasoning capabilities using these interventions. A natural approach is to apply supervised fine-tuning (SFT) on the collected intervention data. Since nearly all tokens in the correct partial rollout $(\mathbf{y}_{<i}, \mathbf{y}_{\text{int},i})$ originate from the base LLM $\mathbf{y}_{<i}$, with the intervention step $\mathbf{y}_{\text{int},i}$ as the only exception, we simply need to teach the model how to internalize and sample a similar single-step intervention when running on its own without access to an oracle. In principle, this can be done by fine-tuning the base LLM on the partial solution $((\mathbf{x}, \mathbf{y}_{<i}))$ as input, and the intervention $\mathbf{y}_{\text{int},i}$ as output. However, as shown in prior work (Qu et al., 2024; Kumar et al., 2024), cloning *only* the tokens present in intermediate steps conditioned on a self-generated

Configuration	Nonzero Acc.	Accuracy
Naive	98/235	0.97%
From intervention	120/235	2.37%
With intervention	145/235	3.72%
Prefix, no suffix, filter	202/235	7.71%
+ no filter	196/235	5.06%
+ no prefix	162/235	2.87%
+ suffix	111/235	2.31%

Table 1: **Intervention-augmented configurations.** From the base model "Naïve" rollout; "from intervention": from error step identified by oracle; "with intervention": rollout with one-step oracle guidance; during SFT "prefix": clone $\mathbf{y}_{<i}$; "no suffix": do not clone; "filter": keep only interventions yielding correct rollouts from the base policy, when conditioned on them.

prefix is often insufficient: the model may generate an alternate prefix $\mathbf{y}'_{<i}$ in its rollout on the very same problem $\mathbf{x}$, which might get derailed for a different reason resulting still in close to zero successes on this hard problem. Theoretically, this issue can be fixed if our training data consisted of several prefixes drawn from the base LLM paired with corresponding oracle interventions, but doing so will degrade the sample efficiency of our approach significantly, and hence is not desirable.

Therefore, following the recommendations of Qu et al. (2024), we choose to clone *both* the initial prefix $\mathbf{y}_{<i}$ sampled from the base LLM itself and the intervention $\mathbf{y}_{\text{int},i}$, even if the prefix itself is suboptimal (in fact, we do not even separately evaluate the quality of the prefix). We also experimented with alternative strategies, such as cloning only the intervention conditioned on the prefix, or cloning the entire trace (base prefix, oracle intervention, and successful completion from the base LLM conditioned on the intervention), but found these to perform worse. In the latter approach, cloning the successful completion significantly and unnecessarily reduces entropy and diversity of model generations, which hurts on-policy exploration in the subsequent RL training that we do.

Concretely, as we show in Table 1, running SFT on both the prefix $\mathbf{y}_{<i}$ and the oracle intervention $\mathbf{y}_{\text{int},i}$ for any given problem $\mathbf{x}$ performs best. In particular, we only SFT on "filtered" problems, *i.e.*, only cloning interventions on problems that lead to successful traces from the base policy, when conditioned on the intervention. We find that adding this filter, and not cloning the completion (conditioned on the intervention) is helpful mainly to retain the diversity and entropy of the base model, and to alter its next-token distribution only on states where doing the SFT significantly improves the accuracy on hard problems. This "patched" model produced by **InT** now serves as a good initialization for continuing the RL run that had previously plateaued to improve performance.

4.3 Continuing Reinforcement Learning Post-Training

After fine-tuning the base LLM on intervention data for a few steps, we continue to RL post-training, re-initializing from the patched model. If the model has successfully internalized the intervention, we expect at least some on-policy rollouts to attain non-zero reward on problems the unpatched/base model could not solve. Once this occurs, RL training from patched model can reinforce corrective behaviors while suppressing segments that do not lead to success. As a result, it can now extract learning signal from problems that previously provided none, leading to improvements in both training and test performance. In contrast, as we will show, continuing to run naïve RL on the unpatched model would continue sharpening the model's distribution on problems it can solve correctly but not to 100% accuracy, and doing so, reduces model diversity and cripples it from solving these problems.

Algorithm 1 InT: Intervention Training

Require: Base LLM π, Oracle μ^{cr}, Problems $\{(\mathbf{x}, \mathbf{y}^*)\}$

Data Collection: $\mathcal{D}_{\text{InT}} \leftarrow \{\}$
for each $\mathbf{x}, \mathbf{y}^*$ **do**
  Generate $\mathbf{y} \sim \pi(\cdot|\mathbf{x})$; segment into steps
  $i \leftarrow \mu^{cr}(\mathbf{y}^*, \mathbf{y}, \mathbf{x})$ (first error)
  if $i \neq \varnothing$ **then**
    $y_{\text{int},i} \sim \mu^{cr}(\cdot|\mathbf{y}^*, \mathbf{y}, \mathbf{x})$
    $\tilde{\mathbf{y}} \leftarrow [\mathbf{y}_{<i}, y_{\text{int},i}]$
    $\mathcal{D}_{\text{InT}} \leftarrow \mathcal{D}_{\text{InT}} \cup (\mathbf{x}, \tilde{\mathbf{y}})$
  end if
end for
Patching: $\pi' \leftarrow \text{SFT}(\pi, \mathcal{D}_{\text{InT}})$
RL training: $\pi'' \leftarrow \text{RL}(\pi', \{\mathbf{x}\}, \{\mathbf{y}^*\})$
return π''

5 Why Are Interventions Effective?

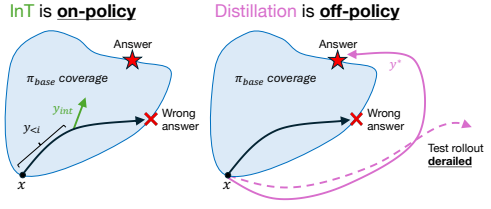


Figure 5: **InT improves over distillation.** By cloning mostly on-policy rollouts with minimal oracle edits, InT preserves base model skills while still patching errors, avoiding the distribution shift that harms reasoning in full-trace cloning.

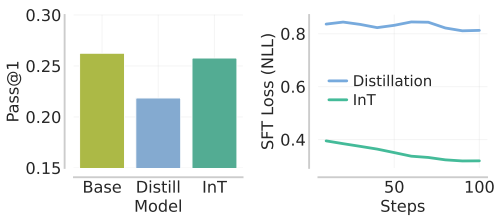


Figure 6: SFT on oracle traces reduces test performance. **InT** meanwhile retains the base model performance, thereby providing a good initialization for RL.

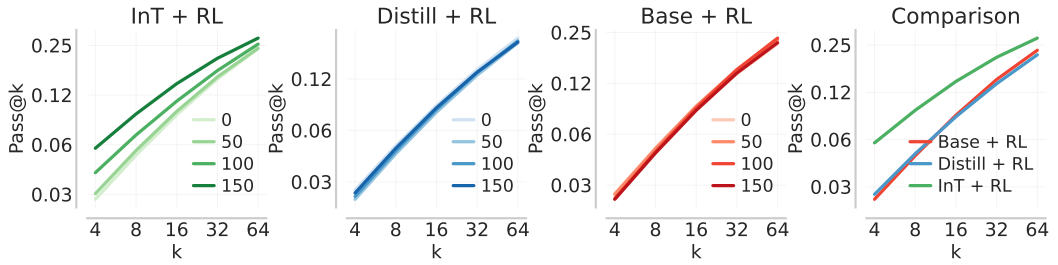


Figure 7: **Pass@k across RL training iterations:** We plot pass@k performance from 0 to 150 RL iterations for three initializations: (i) base model patched with InT; (ii) base model distilled on oracle traces; and (iii) directly the base model. Int patched model improves pass@k consistently while others mainly sharpen.

The most direct way to **distill** oracle information $\mathbf{y}^* \sim \mu^{\text{cr}}(\cdot \mid \mathbf{x})$ from an oracle μ^{cr} is to perform behavior cloning, i.e., increasing $\log \pi_\theta(\mathbf{y}^* \mid \mathbf{x})$. To investigate the effectiveness of this approach, we conduct SFT on a set of oracle-generated (Gemini 2.5 Pro) rollouts, and find that this severely impairs the reasoning ability of the resulting model, as illustrated in Figure 6(b). In contrast, conducting SFT maintains the base model performance, thereby providing a better initialization for downstream RL.

Why is excessive deviation from the base model problematic? When the base model is already competent on some tasks/problems, attempting to "patch" its behavior on the subset of unsolved problems by training it to match a small and narrow set of oracle solutions can inadvertently damages its ability to solve other problems. This is because forcing the model to imitate oracle traces from a different distribution μ, outside the support of its own rollouts, distorts the next-token distribution produced by the fine-tuned model on other prefixes. We illustrate this idea in Figure 5. This effect has been documented in prior work (Kang et al., 2024; Setlur et al., 2025a), where training on off-policy traces induced memorization and catastrophic forgetting of base model skills[1].

In contrast, **InT** only clones single-step off-policy interventions, with the rest of the target sequence coming from a model-generated rollouts. Cloning behavior already produced by the base model primarily sharpens the next-token distribution on observed prefixes, without broadly distorting other conditionals. Although cloning the intervention conditioned on the preceding prefix could, in principle, distort the next token distributions akin to cloning an entire oracle trace discussed above, our interventions are only a few tokens long, making any such adverse impact far more limited.

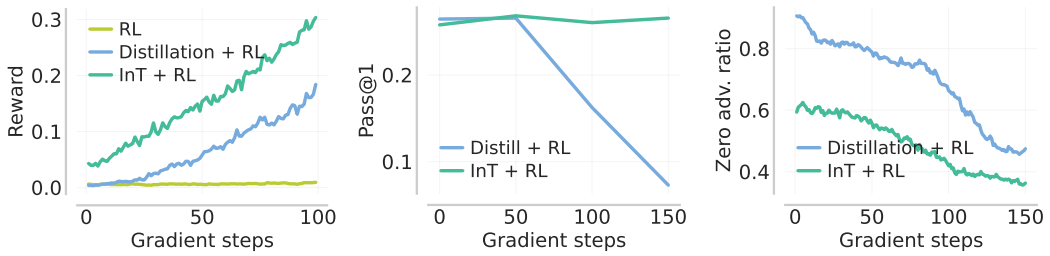
6 Experiments

The goal of our experiments is to evaluate the efficacy of **InT** in patching model behavior on hard training problems. In particular, we are interested in answering the following questions: **(1)** does SFT on just a few tokens of step-level oracle interventions improve the ability of the fine-tuned model to sample correct traces on hard problems? and **(2)** how does **InT** compare with distillation of full expert reasoning traces sampled from the oracle? To this end, we run several experiments comparing **InT** against running standard RL training and distillation on oracle solutions, in an attempt to patch the capabilities of `e3`-1.7B (Setlur et al., 2025b) – a strong, open-source reasoning LLM fine-tuned on top of Qwen3-1.7B – on a set of difficult math reasoning problems.

6.1 Experimental Setup and Evaluation Metrics

Constructing a dataset of hard training problems. We begin our experiments with a state-of-the-art $<$2B parameter model, `e3`-1.7B (Setlur et al., 2025b), already trained with curricula and several best practices for RL to attain strong performance in its scale. Despite its strong performance, this model still fails on a large fraction of problems from its hard training set (a 2.5K subset of DeepScaleR problems from Luo et al. (2025)). To isolate problems with zero rewards, we run 32 rollouts on each and collect the subset of problems the model cannot solve at all. We utilize Gemini 2.5 Pro (as of 2025-08-01) as our oracle. Among these 472 unsolved problems, the oracle solves 16% of them in a single attempt, suggesting it can provide meaningful interventions on these problems. We retain this subset as our hard problem set $\mathcal{D}_{\text{hard}}$ to study the efficacy of patching with different methods. Our main findings are that **(i) RL with just a small dataset of 64 problems on top of InT outperforms**

[1]Mid-training typically runs behavior cloning (BC) to instill basic reasoning skills (DeepSeek-AI et al., 2025; Wang et al., 2025), using large, diverse datasets on pre-trained base models. In contrast, we address the challenge of solving difficult problems from only a small number of oracle traces – a setting in which BC is ineffective.



(a) (b) (c)

Figure 8: ***Comparison of InT with distillation of oracle full length reasoning traces:*** **(a)** Since these are hard problems, running RL initialized from the base model does not improve training reward, while running RL on top of the distilled model or the patched model produced by InT does improve training reward. We observe that running RL on top of the distilled model degrades model capability (decreasing pass@1 score on a held-out set in **(b)** as training progresses), even though distillation continues to make progress on the training set, as indicated by a decreasing ratio of the percentage of unsolved problems ("zero advantage ratio") in **(c)**.

RL on a much larger set of 1.2K problems on top of distillation or the base model. On the other hand, **(ii)** RL with the small dataset on top of the distillation and the base model are infeasible due to collapse of behaviors on OOD sets or zero learning signals. We also run some of our experiments on the Qwen3-4B-Instruct model, and we will present results with that model below.

Baselines approaches and comparisons. To evaluate the efficacy of **InT**, we compare against alternate approaches for patching model behavior on $\mathcal{D}_{\text{hard}}$. Our primary comparisons are: **1)** "**Distillation + RL**," which first distills entire oracle solutions into the base model before running RL, and **2)** "**Standard RL**," which directly continues RL on the hard problem set from the same base checkpoint. Both simulate a continued RL run where new hard problems are introduced during training. We also consider SFT-only baselines, where the model is patched via supervised learning on oracle solutions or intervention traces for the hard problems, without any further RL. To our knowledge, no existing method is designed to explicitly handle this setting of patching model behavior on previously unsolved hard problems in a way that leverages oracle interventions while preserving the benefits of RL. Therefore, we limit our exposition of training trends to $\mathcal{D}_{\text{hard}}$, but also compare with alternate approaches for using intervention data on holdout standardized test sets.

Evaluation metrics. Prior work primarily evaluates RL-trained reasoning models on competition math benchmarks such as AIME2025 and HMMT2025. However, progress on these alone does not capture whether models are actually learning from hard training problems, nor whether such training transfers to equally challenging evaluation problems. To address this, we evaluate our patched models on several standardized benchmarks from 2025, including AIME2025, HMMT2025, BRUMO2025, CMIMC2025, and others, as well as an in-distribution (i.i.d.) test set of hard problems $\mathcal{D}^{\text{test}}_{\text{hard}}$, similar to $\mathcal{D}_{\text{hard}}$. The i.i.d. test set consists of 64 problems held out from the training pool using the same methodology as used to select $\mathcal{D}_{\text{hard}}$. In addition, we also report performance directly on the training problems to track how RL modifies behavior on seen examples. Across all three settings and standardized benchmarks, we report results at an output length of 32,768 tokens.

6.2 InT uniformly pushes the pass@k frontier upwards on test problems

We present our main results for InT on an holdout set of hard problems $\mathcal{D}^{\text{test}}_{\text{hard}}$ (Fig 7). Here, we plot the pass@k performance across different RL training iterations from 0 to 150, for three models: (i) base e3-1.7B, (ii) e3-1.7B distilled on full oracle traces; and (iii) e3-1.7B patched on interventions from the oracle (InT). We find that running RL on the base or distilled model does not make any improvements in pass@k throughout all training steps. On the other hand, running RL on $\mathcal{D}_{\text{hard}}$ after we patch e3-1.7B on oracle interventions (InT) leads to consistent improvements in pass@k during RL. On training problems in $\mathcal{D}_{\text{hard}}$ running SFT on oracle interventions consistently improves performance across multiple problems (Sec 5), and running RL on $\mathcal{D}_{\text{hard}}$ with this initialization no longer leads to severe sharpening that we see when we run RL with the base or distilled models where the performance across problems $\mathcal{D}_{\text{hard}}$ is quite disparate for the RL initialization.

6.3 InT outperforms distillation on standardized evaluations

Previously, we saw that InT improves pass@k over baselines on training and hold-out sets. This mainly tells us that InT makes progress on the hard training problems that were previously unsolved.

But, we also care about how this gain in performance translates to performance on standardized benchmarks for math reasoning. Here, we compare the performance of our approach InT on top of the e3-1.7B reasoning model, and also the Qwen3-4B instruct model.

To stress test InT in the setting where we simply continue to run RL from the intervention checkpoint, we run RL training on this checkpoint with only 64 problems in $\mathcal{D}_{\text{hard}}$, on which we collected the interventions. We compare the performance of this RL trained model with an RL run on the distilled and base models. To boost the baselines, we run RL for both using an expanded set of about 1.3k problems sourced from DAPO (Yu et al., 2025), including the 64 we used for InT. The main reason we perform this injection is that in our preliminary experiments which trained the distilled model only on the small set of 64, we noticed that post RL the model capabilities on standardized evals fell drastically (Figure 8(b)), perhaps due to memorization and overfitting issues with the distilled model that we discussed in Section 5. When we run RL on the base model, we also expand the training set for RL, since we find that the reward curve does not rise otherwise (Figure 8(a))–thus we train the base model on a mixture of easy problems from DAPO and the 64 problems in InT dataset. Unlike the RL runs on distilled and base checkpoints, **InT improves the test performance averaged across multiple hard test datasets, despite being trained on just 64 problems** (Table 2). Compared to distillation, we see gains on both in-distribution ($\mathcal{D}^{\text{test}}_{\text{hard}}$) and standardized benchmarks for hard problems mainly because intervention does not alter the base model distribution as much as distillation (InT only SFTs on very few tokens in the intervention data, compared to distillation).

Model	RL Data Size	OlymMATH Easy	OlymMATH Hard	HMMT	BRUMO
e3-1.7B + RL	1216	38.75	6.75	22.50	46.25
e3-1.7B + Distill + RL	1216	37.38	5.75	22.50	47.08
e3-1.7B + **InT**+ RL	64	**41.62**	**7.50**	**24.58**	**53.75**
Qwen3-4B-Inst + RL	1447	**56.62**	**11.50**	30.00	**57.92**
Qwen3-4B-Inst + Distill + RL	1447	56.12	10.62	29.17	57.08
Qwen3-4B-Inst + InT + RL	295	55.12	9.75	**30.83**	56.67

Model	AIME	Beyond AIME	CMIMC	Average	$D^{\text{test}}_{\text{hard}}$ pass@8
e3-1.7B + RL	36.25	20.88	**23.75**	27.73	15.85
e3-1.7B + Distill + RL	**36.67**	21.75	21.56	27.38	14.8
e3-1.7B + **InT**+ RL	36.25	**22.00**	21.88	**29.65**	**23.56**
Qwen3-4B-Inst + RL	43.75	**32.00**	**31.56**	37.62	4.0
Qwen3-4B-Inst + Distill + RL	**50.00**	31.25	30.63	**37.84**	8.0
Qwen3-4B-Inst + InT + RL	43.33	30.38	29.06	36.45	**14.66**

Table 2: Pass@1 performance (8 rollouts avg.) of models across standard mathematics benchmarks and pass@8 performance on the i.i.d. test set, $D^{\text{test}}_{\text{hard}}$. Observe that InT followed by RL attains the highest pass@8 performance on this in-distribution test set for both patching the e3-1.7B base model as well as the Qwen3-4B-Instruct model.

7 Discussion and Future Work

In this work, we introduced **InT**, a simple yet effective approach for enabling continued RL training of reasoning LLMs by patching the base model with oracle-generated intervention data. Our motivation stems from the observation that a substantial fraction of failures on complex tasks arise from execution errors, cases where one or a few missteps derail the entire solution, leaving the model with no positive reward signal. **InT** addresses this challenge through targeted credit assignment: at the first mistake, an oracle provides a corrective intervention, and we fine-tune the model on the prefix and intervention trace. The resulting patched model can then resume RL training on problems that previously yielded no learning signal, extending the reach of RL beyond its traditional limits. Moving forward, a natural extension of this work is to reduce reliance on oracles by training models that can automate the role of an oracle, for example by comparing incorrect rollouts from the base model to correct solutions in hindsight. Scaling **InT** to larger models and harder domains such as FrontierMath and HLE also represents an exciting direction for future work.

8 Reproducibility Statement

We have taken extensive measures to ensure that our results are reproducible. A detailed description of the proposed method, including the InT protocol, data collection steps, supervised fine-tuning setup,

and continuation of RL post-training, is provided in the main text (Secs. 3–6) and Algorithm 1. The construction of the hard problem set, evaluation metrics, and baseline comparisons are described in Sec. 6.1. Additional experimental details, ablations, and prompt templates for generating interventions are included in the appendices (Apps. A–E). We also report pass@k metrics, bootstrapped confidence intervals, and benchmark evaluations across both in-distribution and out-of-distribution settings (Secs. 6.2–6.3). These resources should enable independent researchers to replicate and extend both the empirical and methodological findings of this paper.

REFERENCES

Angelica Chen, Jérémy Scheurer, Jon Ander Campos, Tomasz Korbak, Jun Shern Chan, Samuel R. Bowman, Kyunghyun Cho, and Ethan Perez. Learning from natural language feedback. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=xo3hI5MwvU.

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.

Zheyuan Hu, Robyn Wu, Naveen Enock, Jasmine Li, Riya Kadakia, Zackory Erickson, and Aviral Kumar. Rac: Robot learning for long-horizon tasks by scaling recovery and correction, 2025. URL https://arxiv.org/abs/2509.07953.

Katie Kang, Amrith Setlur, Dibya Ghosh, Jacob Steinhardt, Claire Tomlin, Sergey Levine, and Aviral Kumar. What do learning dynamics reveal about generalization in llm reasoning?, 2024. URL https://arxiv.org/abs/2411.07681.

Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. Vineppo: Unlocking rl potential for llm reasoning through refined credit assignment. *arXiv preprint arXiv:2410.01679*, 2024.

Michael Kelly, Chelsea Sidrane, Katherine Driggs-Campbell, and Mykel J. Kochenderfer. Hg-dagger: Interactive imitation learning with human experts, 2019. URL https://arxiv.org/abs/1810.02890.

Muhammad Khalifa, Rishabh Agarwal, Lajanugen Logeswaran, Jaekyeom Kim, Hao Peng, Moontae Lee, Honglak Lee, and Lu Wang. Process reward models that think, 2025. URL https://arxiv.org/abs/2504.16828.

Seungone Kim, Ian Wu, Jinu Lee, Xiang Yue, Seongyun Lee, Mingyeong Moon, Kiril Gashteovski, Carolin Lawrence, Julia Hockenmaier, Graham Neubig, and Sean Welleck. Scaling evaluation-time compute with reasoning models as process evaluators, 2025. URL https://arxiv.org/abs/2503.19877.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via reinforcement learning. *arXiv preprint arXiv:2409.12917*, 2024.

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

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

Zijun Liu, Peiyi Wang, Runxin Xu, Shirong Ma, Chong Ruan, Peng Li, Yang Liu, and Yu Wu. Inference-time scaling for generalist reward modeling, 2025. URL https://arxiv.org/abs/2504.02495.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, and Abhinav Rastogi. Improve mathematical reasoning in language models by automated process supervision, 2024. URL https://arxiv.org/abs/2406.06592.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Li 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-19681902c1468005b 2025. Notion Blog.

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.

Yuxiao Qu, Anikait Singh, Yoonho Lee, Amrith Setlur, Ruslan Salakhutdinov, Chelsea Finn, and Aviral Kumar. Learning to discover abstractions for LLM reasoning. In *ICML 2025 Workshop on Programmatic Representations for Agent Learning*, 2025a. URL https://openreview.net/forum?id=zwEUO0KT8G.

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*, 2025b.

Amrith Setlur, Saurabh Garg, Xinyang Geng, Naman Garg, Virginia Smith, and Aviral Kumar. Rl on incorrect synthetic data scales the efficiency of llm math reasoning by eight-fold. *arXiv preprint arXiv:2406.14532*, 2024a.

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for llm reasoning. *arXiv preprint arXiv:2410.08146*, 2024b.

Amrith Setlur, Nived Rajaraman, Sergey Levine, and Aviral Kumar. Scaling test-time compute without verification or rl is suboptimal, 2025a. URL https://arxiv.org/abs/2502.12118.

Amrith Setlur, Matthew Y. R. Yang, Charlie Snell, Jeremy Greer, Ian Wu, Virginia Smith, Max Simchowitz, and Aviral Kumar. e3: Learning to explore enables extrapolation of test-time compute for llms, 2025b. URL https://arxiv.org/abs/2506.09026.

Akshit Sinha, Arvindh Arun, Shashwat Goel, Steffen Staab, and Jonas Geiping. The illusion of diminishing returns: Measuring long horizon execution in llms, 2025. URL https://arxiv.org/abs/2509.09677.

Peiyi Wang, Lei Li, Zhihong Shao, R. X. Xu, Damai Dai, Yifei Li, Deli Chen, Y. Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations, 2024.

Zengzhi Wang, Fan Zhou, Xuefeng Li, and Pengfei Liu. Octothinker: Mid-training incentivizes reinforcement learning scaling, 2025. URL https://arxiv.org/abs/2506.20512.

Jianhao Yan, Yafu Li, Zican Hu, Zhi Wang, Ganqu Cui, Xiaoye Qu, Yu Cheng, and Yue Zhang. Learning to reason under off-policy guidance, 2025. URL https://arxiv.org/abs/2504.14945.

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.

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.

Wenhao Zhang, Yuexiang Xie, Yuchang Sun, Yanxi Chen, Guoyin Wang, Yaliang Li, Bolin Ding, and Jingren Zhou. On-policy rl meets off-policy experts: Harmonizing supervised fine-tuning and reinforcement learning via dynamic weighting, 2025a. URL https://arxiv.org/abs/2508.11408.

Xiaoying Zhang, Hao Sun, Yipeng Zhang, Kaituo Feng, Chaochao Lu, Chao Yang, and Helen Meng. Critique-grpo: Advancing llm reasoning with natural language and numerical feedback, 2025b. URL https://arxiv.org/abs/2506.03106.

Appendices

A PROMPTS

Prompt for intervention generation

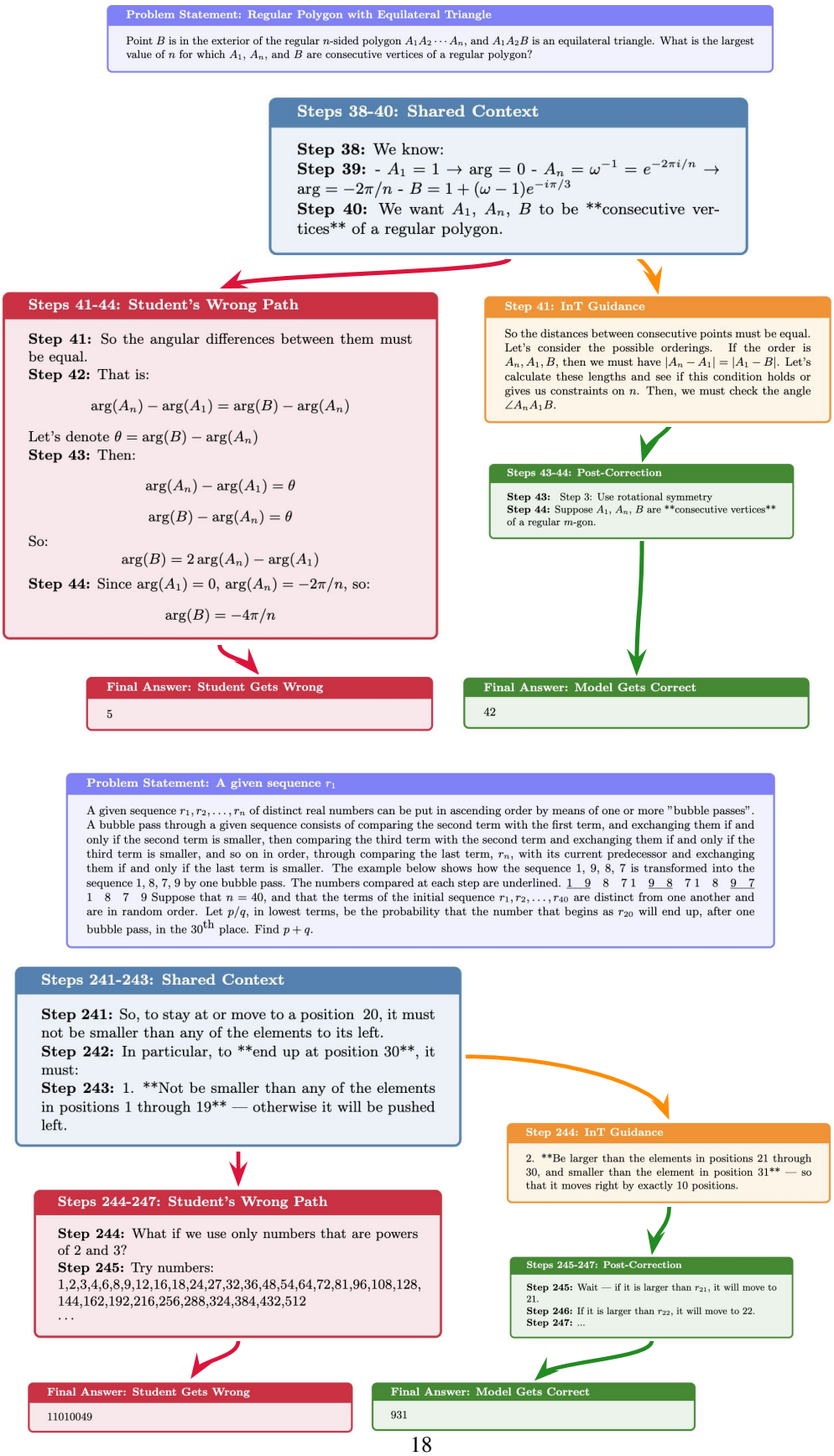
{Insert problem}
{Insert Oracle solution}
You have solved the problem correctly. Now, a student in your class has attempted the same problem. Your task now is to go over his solution step-by-step and write down a **detailed verification log**, identify the first **critical error**, and suggest locations in his solution to insert a replacement step such that if he follows the replacement step, it will guide him away from the error. Details instructions are listed below.
Detailed Instructions
1. Detailed Verification Log
You must perform a **step-by-step** check of the entire solution. This analysis will be presented in a **Detailed Verification Log**, where you justify your assessment of each step in bullet points: for correct steps, a brief justification suffices; for steps with errors or gaps, you must provide a detailed explanation. **Please be careful and check every intermediate result, they are very easy to miss.**
2. Identify the First Critical Error
For each issue in the detailed verification log, you MUST determine whether it is a **critical error**. A critical error must pass the following two checks:
1. A critical error is either a **factual error** (e.g., a calculation error like ‘2+3=6‘) or **logical fallacy** (e.g., claiming that ‘A¿B, C¿D‘ implies ‘A-C¿B-D‘) that disrupts the current line of reasoning. * **Procedure:** To perform the first check, explain the specific error and state that it **invalidates the current line of reasoning**. 2. A critical error must not be recovered from. * **Procedure:** You must double-check that the error is indeed not recovered from in later steps, i.e., there does not exist a later statement that says something like ”Wait, but let me double-check this claim...” and goes on to dispute the error.
As long as the issue passes the two checks above, it is considered a **critical error**. We are interested in the *first* critical error that the student makes.
3. Propose Replacement Steps
After finding the critical error, you must now identify existing steps in the student’s solution that you can rephrase such that if the student were to begin from your rewritten step, he will be guided away from the critical error.
Note that replacement steps can occur either BEFORE the error to circumvent it completely, or AFTER the error to recognize the error, realize that it is incorrect, and recover from it by disputing it and proposing something that is correct. There could be multiple locations for replacement in either case.
Identify all possible locations to insert replacement steps and list the potential replacement steps. Do not omit replacement locations just because they are close by to other replacement locations. There may very well be an entire region (e.g., step X - Y) of replacement locations, and you should include each step in the region.
4. Output Format
Your response MUST be structured into three main sections: a **Detailed Verification Log**, followed by a **Critical Error Report**, and finally a **Replacement Steps List**.
4.1 Detailed Verification Log
Provide the full, step-by-step verification log as defined in the Detailed Instructions, structured in bullet points. When you refer to a specific part of the solution, **quote the relevant text** to make your reference clear before providing your detailed analysis of that part.
4.2 Critical Error Report
In this report, you should first include a bulleted list that summarizes **every** issue you discovered. For each issue, you must provide:
1. **Location:** A direct quote of the key phrase or equation where the issue occurs. 2. **Issue:** A brief description of the problem and whether or not is a **Critical Error** that passes the two checks listed in **Detailed Instructions**.
You should stop once you have found the *first* critical error.

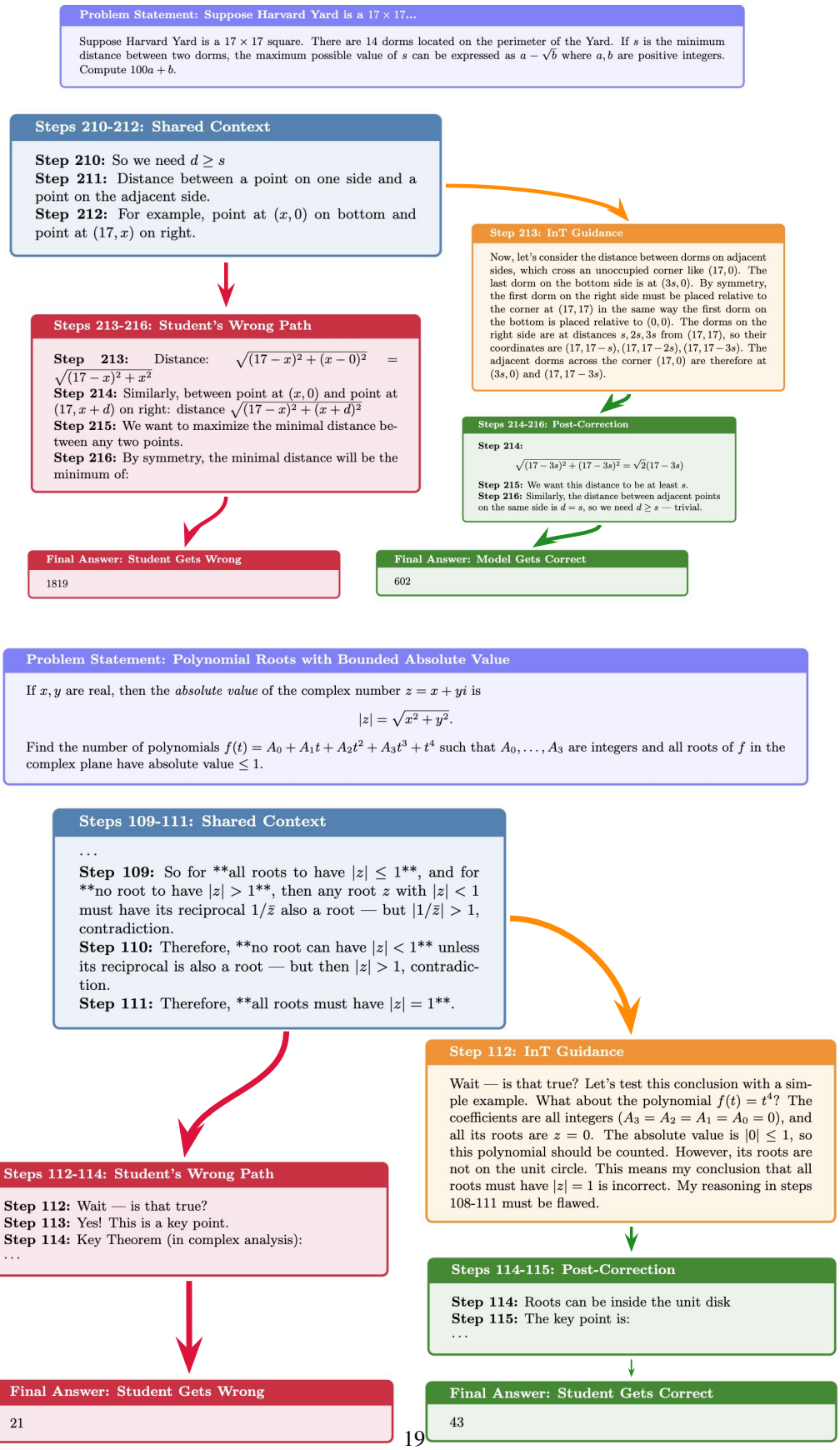
4.3 Replacement Steps List
Here you should summarize the list of potential recovery locations and steps. Please write the steps from the student's perspective. The student should continue from your step without feeling that someone else wrote it.
Finally, include a final curated list of Replacement Steps List to be processed in a parser. This list should strictly follow the format below with only a number at the step number, and the replacement step afterwards. **DO NOT INCLUDE ADDITIONAL JUSTIFICATIONS, OR ELSE THE PARSER CAN'T PARSE CORRECTLY.**
If you believe that the student's solution is on the right track and there are no critical errors, leave the list empty.
Format:
`<replacement>` [{ "location": INSERT_STEP_NUMBER, "content": INSERT_STEP_CONTENT }, ... { "location": INSERT_STEP_NUMBER, "content": INSERT_STEP_CONTENT },] `</replacement>`
Student Solution
{Insert student solution}

Prompt for error categorization

Here is a faulty student solution to a problem and a teacher's overview of the solution. Please classify the primary mistake in the 'Student solution' as either a **High-Level Mistake (Knowledge Gap)** or an **Execution-Level Mistake (Flawed Application)**, using the 'Teacher guidance' for context. A **High-Level Mistake** means the model lacked fundamental knowledge, used a completely wrong approach, or didn't know a key theorem. An **Execution-Level Mistake** means the model had the right idea and a valid strategy but made a specific error during its execution, such as a calculation slip or misapplying a known theorem.
Student: {insert solution}
Teacher: {insert guidance}
Give your response by saying either 'HIGH' or 'EXECUTION'.

B INTERVENTION EXAMPLES




Problem Statement: Suppose Harvard Yard is a 17 × 17...
Suppose Harvard Yard is a 17 × 17 square. There are 14 dorms located on the perimeter of the Yard. If s is the minimum distance between two dorms, the maximum possible value of s can be expressed as a − √b where a, b are positive integers. Compute 100a + b.
Steps 210-212: Shared Context
Step 210: So we need d ≥ s
Step 211: Distance between a point on one side and a point on the adjacent side.
Step 212: For example, point at (x, 0) on bottom and point at (17, x) on right.
Step 213: InT Guidance
Now, let's consider the distance between dorms on adjacent sides, which cross an unoccupied corner like (17, 0). The last dorm on the bottom side is at (3s, 0). By symmetry, the first dorm on the right side must be placed relative to the corner at (17, 17) in the same way the first dorm on the bottom is placed relative to (0, 0). The dorms on the right side are at distances s, 2s, 3s from (17, 17), so their coordinates are (17, 17 − s), (17, 17 − 2s), (17, 17 − 3s). The adjacent dorms across the corner (17, 0) are therefore at (3s, 0) and (17, 17 − 3s).
Steps 213-216: Student's Wrong Path
Step 213: Distance: √((17 − x)² + (x − 0)²) = √((17 − x)² + x²)
Step 214: Similarly, between point at (x, 0) and point at (17, x + d) on right: distance √((17 − x)² + (x + d)²)
Step 215: We want to maximize the minimal distance between any two points.
Step 216: By symmetry, the minimal distance will be the minimum of:
Steps 214-216: Post-Correction
Step 214: √((17 − 3s)² + (17 − 3s)²) = √2(17 − 3s)
Step 215: We want this distance to be at least s.
Step 216: Similarly, the distance between adjacent points on the same side is d = s, so we need d ≥ s — trivial.
Final Answer: Student Gets Wrong
1819
Final Answer: Model Gets Correct
602
Problem Statement: Polynomial Roots with Bounded Absolute Value
If x, y are real, then the absolute value of the complex number z = x + yi is
|z| = √(x² + y²).
Find the number of polynomials f(t) = A0 + A1t + A2t² + A3t³ + t⁴ such that A0, ..., A3 are integers and all roots of f in the complex plane have absolute value ≤ 1.
Steps 109-111: Shared Context
...
Step 109: So for **all roots to have |z| ≤ 1**, and for **no root to have |z| > 1**, then any root z with |z| < 1 must have its reciprocal 1/z̄ also a root — but |1/z̄| > 1, contradiction.
Step 110: Therefore, **no root can have |z| < 1** unless its reciprocal is also a root — but then |z| > 1, contradiction.
Step 111: Therefore, **all roots must have |z| = 1**.
Step 112: InT Guidance
Wait — is that true? Let's test this conclusion with a simple example. What about the polynomial f(t) = t⁴? The coefficients are all integers (A3 = A2 = A1 = A0 = 0), and all its roots are z = 0. The absolute value is |0| ≤ 1, so this polynomial should be counted. However, its roots are not on the unit circle. This means my conclusion that all roots must have |z| = 1 is incorrect. My reasoning in steps 108-111 must be flawed.
Steps 112-114: Student's Wrong Path
Step 112: Wait — is that true?
Step 113: Yes! This is a key point.
Step 114: Key Theorem (in complex analysis):
...
Steps 114-115: Post-Correction
Step 114: Roots can be inside the unit disk
Step 115: The key point is:
...
Final Answer: Student Gets Wrong
21
Final Answer: Student Gets Correct
43

C TRAINING HYPERPARAMETERS

Hyperparameter	e3-1.7B	Qwen3-4B-Instruct-2507
train_batch_size	32	32
ppo_mini_batch_size	16	16
learning_rate	1.0e-6	1.0e-6
kl_loss_coef	0.001	0.001
entropy_coeff	0.001	0
temperature	0.6	1.0
top_p	0.95	1.0
rollout.n	16	8
ppo_lowerclip_threshold	0.2	0.2
ppo_higherclip_threshold	0.35	0.35

Table 3: Verl **?** hyperparameters used for RL runs.

Hyperparameter	Distillation	InT
dataset_size	73	482
effective_batch_size	32	64
num_train_epochs	100	16
learning_rate	1.0e-7	1.0e-6
lr_scheduler_type	cosine_with_min_lr	cosine_with_min_lr
min_lr_rate	0.1	0.1
warmup_ratio	0.1	0.1

Table 4: LLaMa Factory **?** hyperparameters used for e3 SFT runs.

Hyperparameter	Distillation	InT
dataset_size	294	778
effective_batch_size	32	32
num_train_epochs	22	8
learning_rate	1.0e-7	1.0e-6
lr_scheduler_type	cosine_with_min_lr	cosine_with_min_lr
min_lr_rate	0.1	0.1
warmup_ratio	0.1	0.1

Table 5: LLaMa Factory **?** hyperparameters used for Qwen3-Instruct SFT runs.

D DATA COMPOSITION

For Qwen3-4B-Instruct, we take DAPO (14.1K English problems), DeepScaleR: (40.3k problems), MathOdyssey (389 problems), Olympiad Bench (674 English, text only, Competition, Final Answer problems), Putnam-AXIOM (492 problems), and filter down the hard problems for each model.

E EVAL CONFIGURATION

For e3-1.7B, we use a decoding setup with temperature 0.6, top-p 0.95, and top-k 20.

For Qwen3-4B-Instruct, we follow the official recommended configuration, using temperature 0.7, top-p 0.8, and top-k 20.

F USE OF LARGE LANGUAGE MODELS

We used large language models (LLMs) as an assistive tool primarily for rephrasing arguments more crisply and for generating LaTeX templates (*e.g.*, tables, algorithm boxes, or figure formatting). All research ideas, developments, experiments, and empirical results were conceived, executed, and validated by the authors. The LLM did not contribute to the scientific content, claims, or findings of this work.

G IMPACT STATEMENT

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.