GLoRe: When, Where, and How to Improve LLM Reasoning via Global and Local Refinements

Alex Havrilla[1,2,*], **Sharath Raparthy**[1], **Christoforos Nalmpantis**[1], **Jane Dwivedi-Yu**[1], **Maksym Zhuravinskyi**[3], **Eric Hambro**[1], **Roberta Raileanu**[1]

[1]FAIR at Meta, [2]Georgia Institute of Technology, [3]StabilityAI

[*]Work done during Meta internship

State-of-the-art language models can exhibit impressive reasoning refinement capabilities on math, science or coding tasks. However, recent work demonstrates that even the best models struggle to identify *when and where to refine* without access to external feedback. Outcome-based Reward Models (**ORMs**), trained to predict correctness of the final answer indicating when to refine, offer one convenient solution. However, when used to indicate where to refine, we find that ORMs tend to be *overly-pessimistic* when used to assess intermediate reasoning steps, resulting in excessive refinement of valid solutions. Process Based Reward Models (**PRMs**), trained to predict correctness of intermediate steps indicating where to refine, have been used to improve LLM reasoning ability via rejection sampling or reinforcement learning (RL) fine-tuning. But they are expensive to train, requiring extensive human annotations. In this paper, we propose Stepwise ORMs (**SORMs**) which are trained, only on synthetic data, to approximate the expected future reward of the optimal policy or $V^{\star}$. More specifically, SORMs are trained to predict the correctness of the final answer when sampling the current policy many times (rather than only once as in the case of ORMs). Our experiments show that SORMs can more accurately detect incorrect reasoning steps compared to ORMs, thus improving downstream accuracy when doing refinements. We then train *global* refinement models, which take only the question and a draft solution as input and predict a corrected solution, and *local* refinement models which also take as input a critique indicating the location of the first reasoning error. We generate training data for both models synthetically by reusing data used to train the SORM. We find combining global and local refinements, using the ORM as a reranker, significantly outperforms either one individually, as well as a best of three sample baseline. With this strategy we can improve the accuracy of a LLaMA-2 13B model (already fine-tuned with RL) on GSM8K from 53% to 65% when greedily sampled.

Date: February 20, 2024
Correspondence: Alex Havrilla at ahavrilla3@gatech.edu

1 Introduction

State-of-the-art large language models (**LLMs**) exhibit a wide range of downstream capabilities after pre-training. This includes the ability to refine their reasoning on math, science, or coding problems (OpenAI, 2023; Touvron et al., 2023; Chowdhery et al., 2022). However, under close inspection, this refinement ability is quite brittle, often unable to even identify when a solution needs refinement (Huang et al., 2023). When LLMs do produce successful refinements on hard reasoning tasks this is often due to the incorporation of external forms of feedback, e.g. feedback from humans or code, stronger models, or other tools (Zhou et al., 2023; Gou et al., 2023). In this work, we carefully examine and improve the self-refinement abilities of LLMs on reasoning tasks without any external feedback other than the ground truth answers of the training problems. Notably, this means we make no use of data or feedback from humans or stronger models. To do so we start by heuristically decomposing the refinement problem into three parts: firstly deciding *when* to refine, then *where* to refine, and finally *how* to refine.

Outcome Based Reward Models (**ORMs**) (Cobbe et al., 2021), first introduced as an estimator of final answer correctness given a question to do solution reranking, are a natural choice for addressing step one. For deciding where to refine, we carefully examine the generalization of ORMs to intermediate steps. We find the accuracy of the underlying data generating policy π directly affects the ORM's ability to learn correctness of intermediate solutions steps. This leads to the ORM often under-estimating the solvability of a problem from an intermediate step S_i. The result is high false-negative rates when used to classify steps with errors. Process Based Reward Models (**PRMs**) instead are trained to directly estimate the correctness of each step. Yet this requires extensive human labeling of model-generated solution steps as valid or invalid. In an effort to improve our ability to give intermediate step feedback, we introduce the Stepwise ORMs (**SORMs**) which explicitly predict labels at each step indicating the presence of an error. We generate SORM training data by sampling a student policy π many times at a step S_i in solution S, labeling S_i as valid if we successfully reach the final answer. From an RL perspective, this can be interpreted as learning (a lower bound of) the optimal value function V^* of the reasoning task via approximation of the optimal policy π^* with rejection sampling. The resulting SORM gives better intermediate step-level feedback, allowing us to give information to the refinement model about both *when* and *where* to refine. The refinement model must then only decide *how* to refine.

We initially train *global* refinement models capable of refining the entire reasoning trace without any feedback beyond an initial draft solution D. The training data is generated synthetically, by pairing correct solutions with incorrect solutions as in Welleck et al. (2022). An evaluation of the global refinement model confirms its inability to correctly identify when to refine, demonstrating the need for an ORM. Reusing the SORM training data, we train a *local* refinement model which uses the feedback given by the SORM to identify the first incorrect reasoning step. We then compare the performance of global versus local refinements on a test set of incorrect solution drafts, finding similar refinement accuracy but on largely disjoint sets of problems. In this sense the global and local refinement models are complementary, with local refinements often able to solve problems global refinements cannot and vice versa. To obtain our best results we combine both global and local refinements, using the ORM to choose the most promising one by acting as a reranker of both plus the initial draft. Using this strategy, we can improve the accuracy of an already strong RL fine-tuned Llama-2 13B mode from 53% to 65% when greedily sampled.

In summary we make the following contributions:

- Decompose the refinement problem into three parts, namely deciding *when, where, and how* to refine a solution by leveraging reward models (RMs).
- Highlight the limitations of ORMs in judging the correctness of intermediate steps, despite their ability to judge the correctness of the final answer.
- Introduce the step-wise ORM (SORM) to refine which is trained only on synthetic data and can more accurately evaluate intermediate steps than the ORM.
- Propose a new method for refining LLM reasoning that decides *when* to refine using an ORM, *where* to refine using a SORM, and *how* to refine using both global and local refinements. We find the two types of refinement are complementary, each able to solve a large class of problems the other cannot.
- Demonstrate performance improvements of up to 12% on GSM8K for a 13B LLaMA-2 model using our approach.

2 Background

Reasoning: We define a reasoning task τ as a distribution of (natural language) question/answer pairs $(Q, A) \sim \tau$. The answer could be either a single final answer, typically a numerical value in case of math problems for ease of evaluation, or include a CoT style solution trace justifying a numerical final answer. We often further write the answer A as consisting of atomic steps $A = (S_1, ..., S_L)$ with the final answer being given on step L. The notion of a start of a new "step" is problem dependent but in our case always corresponds to a newline token.

Reward Modeling: Given a reinforcement learning (RL) environment, a reward model can be trained to approximate the reward coming from an action a in state s (Christiano et al., 2017). In the language setting, reward models are trained to approximate the reward given to a response generated by a LLM (Ouyang et al., 2022). The reward is generally sparse and given at the end of a generation as in the case of RLHF (Christiano et al., 2017; Ziegler et al., 2019) where a contrastive preference model is learned for RL and rejection sampling.

Similar to this is the *Outcome-based Reward Model* (**ORM**) first proposed as a final answer verifier used to rerank GSM8K solutions (Cobbe et al., 2021). Formally, we say the ORM estimates $p(\texttt{is\_correct}(A)|Q, A)$ where Q is a question and A is a model generated answer. Training data for the ORM is generated by sampling an underlying student model π many times on questions from a reasoning task τ. The ORM is then trained to predict $p(\texttt{is\_correct}(A)|Q, P_i)$ where P_i is prefix of intermediate steps $(S_1, ..., S_i)$ and A is any hypothetical continuation of P_i sampled from π. i.e., at intermediate steps we may interpret the ORM as estimating the probability of P_i leading to the correct final answer. We may sometimes write ORM_π to emphasize the ORM's dependence on its data generating student model π. More recently, *Process-based Reward Models* (**PRMs**) have been proposed to directly supervise the correctness of each step in a solution $A = (S_1, ..., S_L)$ (Lightman et al., 2023; Uesato et al., 2022). Formally, we write a PRM predicts $p(\texttt{is\_correct}(S_i)|P_i, Q)$ where S_i is the last step of P_i.

Refinement: We define a refinement of a draft solution A_D and question Q as a new solution A_R generated by conditioning on both Q and A_D. We consider both global refinement models, which take as input only Q, A_D and predict $p(A_R|Q, A_D)$, and local refinement models, which take as input an extra parameter E indicating the location of an error in A_D, to predict $p(A_R|Q, A_D, E)$.

Notation: For the rest of the paper we refer to the pre-trained LLM fine-tuned for downstream tasks as the *base model*. We fine-tune the base model, either on supervised data or using RL, to produce a student model that generates answers A given a question Q. Sometimes we may also write the student model as a policy π implicitly depending on learnable parameters θ. $\mathcal{D}_{\text{TASK}}$ will be used to denote a dataset for TASK τ with train split $\mathcal{D}_{\text{TASK}}^{\text{train}}$ and test split $\mathcal{D}_{\text{TASK}}^{\text{test}}$ being implicit. We will use Q to denote a question and $A_1, ..., A_k$ to denote solution traces. Sometimes we will write $A = (S_1, ..., S_L)$ which decomposes the solution trace A into intermediate steps S_i. $P_i = (S_1, ..., S_i)$ will be used to denote the prefix of steps up to S_i. Additionally we will sometimes use A_{GR} and A_{LR} to represent global and local refinements of A_D. V^π denotes the value function of policy π. V^* denotes the optimal value function with dependence on the background task implicit.

3 Related Works

LLM Reasoning: State-of-the-art (SOTA) large language models (**LLMs**) (OpenAI, 2023; Touvron et al., 2023; Bai et al., 2022; Chowdhery et al., 2022) demonstrate increasingly impressive abilities on hard reasoning tasks as studied by a wide range of math, science, and code benchmarks (Cobbe et al., 2021; Hendrycks et al., 2021b; Sawada et al., 2023; Liang et al., 2022; Srivastava et al., 2022; Rein et al., 2023; Mialon et al., 2023; Chollet, 2019; Hendrycks et al., 2021a; Austin et al., 2021; Mishra et al., 2022; Patel et al., 2021; Gao et al., 2021). *Chain of thought* (**CoT**) (Wei et al., 2022) and related techniques (Chen et al., 2022; Yao et al., 2023a; Besta et al., 2023) have emerged as dominant methods significantly boosting LLM performance on these types of tasks. CoT methods allow LLMs to defer giving their final answer by first generating a "chain of thought" involving intermediate computations needed to correctly solve the problem.

LLM Refinement: Intimately related to reasoning ability is a model's ability to refine previous answers. This work studies the ability of large language models to self-refine their CoT solutions to math reasoning tasks. Several works (Yao et al., 2022; Madaan et al., 2023; Zhou et al., 2023) demonstrate SOTA LLM self-refining and self-critiquing abilities on a range of tasks via prompting and/or tool usage. However, recent work (Huang et al., 2023) argues even for the strongest models such techniques struggle on hard, open-ended reasoning tasks where the model itself must decide when to stop refinement.

Other papers use hand-crafted data augmentation (Paul et al., 2023) or gather human data (Wang et al., 2023b; Chen, 2023; Lee et al., 2023; Saunders et al., 2022; Schick et al., 2022) while still others use techniques from reinforcement learning to generate critiques (Akyurek et al., 2023; Yao et al., 2023b) for larger models. Most related to us is (Welleck et al., 2022) which trains global refinement models in an implicit reinforcement learning like manner by pairing *low-value* rollouts with *high-value* rollouts.

Process-based reward modeling (**PRMs**) (Uesato et al., 2022; Lightman et al., 2023) gives a denser, step-by-step reward for the "correctness" of a particular step without explicitly modeling the step's impact on the correctness of the final answer. Both ORMs and PRMs are most often used as rerankers over large numbers of candidate solutions, with PRMs generally outperforming ORMs (Lightman et al., 2023). However, PRMs areexpensive to train, requiring extensive human annotation of each step. Uesato et al. (2022) directly compares the performance of a 70B ORM vs PRM on GSM8K, finding both performing similarly when used as a reward for RL and for reranking. They qualitatively note the ORM appears to somewhat generalize to intermediate steps in a manner similar to a PRM but do not quantitatively ablate this observation over multiple models or tasks. Li et al. (2022) attempt to train synthetic stepwise verifiers similar to a PRM which are then used for Monte Carlo Tree Search. Concurrent work (Wang et al., 2023a) proposes training a synthetic process based reward model in a manner similar to our SORM. They then use the RM downstream for RL fine-tuning and rejection sampling.

In contrast to the above works we conduct a careful comparison of ORM/SORM verification abilities at the step level. We then propose to utilize the ORM/SORM for refinement. We accomplish this by generating fully synthetic stepwise labels which allow us to train both the SORM and refinement models.

4 Method

We start by decomposing the refinement problem into three stages: First, learning *when* a draft D is correct and when it needs refinement. Second, learning *where* to begin refinement by identifying the first incorrect step. Third, learning *how* to correct the initial draft. We can naturally address step one by using the ORM which is trained to predict the probability of a draft being correct. This alleviates some of the difficulty, now only requiring the refiner to identify where and when to refine. Additionally, when doing local refinement, we propose using the (S)ORM to localize the position of the first error. This simplifies the task even more, as now the local refiner must only decide how to fix the error and continue from there.

Localizing errors with Reward Models: To identify errors at the step level we can leverage the ORM by taking its intermediate prediction $ORM_\pi(Q, P_i)$ at a step S_i where $P_i = (S_1, ..., S_i)$ is the prefix of all steps up to S_i. Recall the ORM is trained to predict the likelihood a solution with prefix P_i results in a correct final answer. Importantly, the likelihood inferred from this training data is heavily dependent on the data generating policy π. For this reason we sometimes include the subscript ORM_π, omitting it when not needed.

To best understand the behavior of the ORM's prediction at an intermediate step S_i, we can interpret it as the *value function* of π. Recall the value function V^π of a policy π is computed as $V^\pi(S) = \mathbb{E}_{\tau \sim \pi(S)} R(\tau)$ i.e. the mean return of the policy π from the state S. In the context of reasoning problems, the states we consider are of the form $S = (Q, S_1, ..., S_i)$ with question Q and intermediate steps S_j. In our setting by default there is only a sparse reward of $+1$ given at the terminal state for a correct final answer.

We can write $ORM_\pi(Q, P_i) \approx p(\texttt{is\_correct(A)}|Q, P_i, \pi)$ where $P_i = (S_1, ..., S_i)$ is the prefix of all prior steps and `is_correct(A)` is the event that a full solution A sampled from π with prefix P_i has the correct final answer. We can then write $\mathbb{E}_{A \sim \pi(Q,P_i)} R(A) = \mathbb{E}_{A \sim \pi(Q,P_i)} 1_{\texttt{is\_correct(A)}} = p(\texttt{is\_correct(A)}|Q, P_i, \pi)$. Therefore, an approximation to the value function of a policy π is predicting exactly the same thing as the outcome-based reward model at an intermediate step S. So we may treat the **ORM as approximating a value function for the student model** π used to generate its training data.

Ideally we might want to use the ORM to identify where a mistake was made by finding the first step S_i such that $ORM(Q, P_i) \leq 0.5$ i.e. P_i is likely to result in the wrong answer. However, because the ORM is acting as a value function for π, it tends to **hallucinate error steps** simply because it expects the data generating student π to fail. For example, if π almost always fails problems involving division, the ORM will assign low probability of success to a division problem even before the student takes its first step. In these cases we say the ORM is *overly pessimistic*. This is not ideal when using the ORM to identify the location of mistakes.

Learning a Step-Wise ORM (SORM): Another natural candidate which could be used to identify mistakes at each step is a Process Based Reward Model (PRM) (Lightman et al., 2023). A PRM estimates the probability of correctness of a step S_i, $p(S_i \texttt{ correct}|Q, S_1, S_2, ..., S_i)$ independently of its impact on the final answer. However, this would be expensive, requiring collecting human annotated samples. Instead, we propose to approximate the *optimal value function* V^* of the reasoning task. V^* corresponds to the value function of the

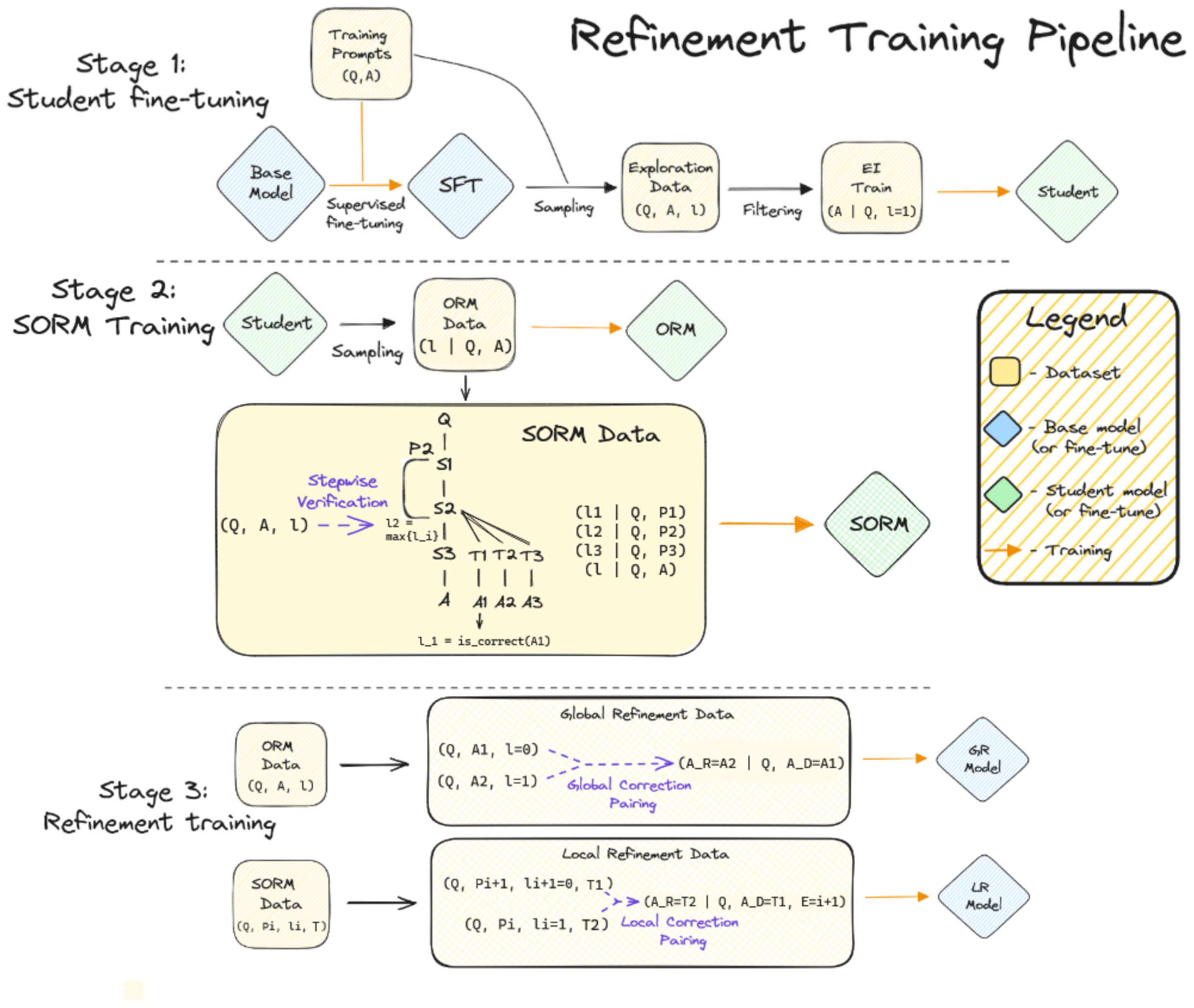


Figure 1 Diagram for three-stage refinement training pipeline. First we RL fine-tune the base model to produce a strong student policy π. Then we generate ORM/SORM training data by sampling π on the training data. Finally, we generate refinement data by pairing together incorrect rollouts with correct rollouts globally and locally. Note, (Q, A, l) denotes a question, answer pair with binary correctness label l. A SORM training sample (Q, P_i, l_i, T) includes a prefix of steps $(S_1, ..., S_i)$, a binary correctness label l_i for the prefix, and the set of verification rollouts $T_1, ..., T_K$ from P_i verifying correctness of P_i. Global correction pairing is used to produce global refinement training data by pairing incorrect ORM rollouts with correct ORM rollouts. Analgously, local correction pairing pairs incorrect verifictions T_- of (incorrect) P_{i+1} with correct verifications T_+ of P_i. This then produces a label $E = i + 1$ indicating an error at step $i + 1$ in the initial draft $A_D = T_-$ and refinement $A_R = T_+$.

optimal policy which is able to successfully solve the reasoning task from any logically valid intermediate state S_j. Such an optimal value function would have $V^*(Q, S_1, ..., S_i) = 1$ for a solution prefix with no mistakes, and $V^*(Q, S_1, ..., S_i) = 0$ if the prefix already contains a mistake which will result in an incorrect final answer. We call models we train to directly approximate V^* stepwise ORMs or **SORMs**.

As discussed in Uesato et al. (2022), the ORM possesses some knowledge of intermediate solution correctness, allowing it to approximate a PRM. However, we find in practice this property is dependent on the size of the base model and the difficulty of the task τ, with ORMs trained on data from larger students and easier tasks giving better approximations to a PRM. When interpreting the ORM as a value function V^π of the data generating student, this makes sense. A larger, more capable student will better approximate the optimal policy π^*, resulting in a better approximation of the ORM to V^*.

4.1 Training pipeline

Recall, we assume no access to data from humans or better models for fine-tuning. Thus we must generate all training data synthetically for both global and local refinement. Additionally we must generate data for both the ORM and SORM. We divide our proposed training pipeline in three steps. See Figure 1 for a diagram outlining each step.

Step 1: Fine-tuning a student model

To produce base checkpoints from which we can generate ORM/SORM training data and initial refinement drafts A_D we fine-tune models using Expert Iteration (**EI**) (Silver et al., 2017). This is done by sampling the student model $K = 96$ times per question and filtering out rollouts with incorrect final answers. De-duplication is then performed on the remaining samples to construct a new finetuning dataset $\mathcal{R}_1$. We then combine this with any available SFT data producing $\mathcal{D}_1$ which we use to again fine-tune the pre-trained model. This process is repeated until the maj@1 score of each subsequent fine-tune converges. Note, the fine-tuning dataset used at step i is $\mathcal{D}_i = R_i \cup \mathcal{D}_{i-1}$: the union of rollouts generated at the ith step with previously generated training data ($\mathcal{D}_0 = \emptyset$ or SFT). In the case of GSM8K we first fine-tune each pre-trained model on the given supervised fine-tuning (**SFT**) data. For SVAMP, which has no CoT SFT data, we 1-shot prompted the pretrained model to generate solutions used to construct an initial EI dataset. We call the resulting model the student model or student policy π. For more details of this training process and resulting models see Section B in the appendix.

Step 2: Training the ORM/SORM

We generate ORM training data by sampling the RL fine-tuned student policy π K times per prompt. As usual, we then label each intermediate step S_i as correct if the final answer is correct and incorrect otherwise. To generate training data for our SORM we sample an approximation of the optimal policy π^* at each step S_i in a model generated solution and check correctness of the final answer. We aim to approximate π^* via rejection sampling of our student policy π^*. Concretely, to produce a training label for a step S_i in model generated rollout S, we sample the student policy π for K rollouts starting from the prefix $P_i = (S_1, ..., S_i)$. This produces verifying traces $T_1, ..., T_K$ with correct final answers indicated by $l_1, ..., l_K$. We then label S_i as `positive` if $\max_j l_j = 1$ i.e. we can find the correct final answer starting from S_i. In practice we sample $K = 8$ rollouts per step, each generating at most 300 tokens. Otherwise we label S_i as `negative`. We then train the SORM in exactly the same manner as the ORM, predicting the appropriate label after each step in a solution. See Section G for a comparison of the labels assigned by this process to ground truth human labels.

SORM data post-processing To improve our approximation to the optimal policy via rejection sampling we apply several post-processing steps: **1)** If a step S_i has a positive label l_i we set $l_j = 1$ for $j \leq i$. I.e. all steps before a positive steps are labeled as positive. This accounts for particularly hard problems where the student is able to find the solution with K samples from the step S_i but not any prior step S_j, $j < i$. **2)** We enforce a *consistency constraint* on the verifying rollouts, requiring each intermediate result R_i computed on step S_i of the solution to be used later on. This helps prevent false positives by requiring a verification to make full use of the previous steps it's verifying. In practice we implement this by checking for each R_i as a string in the suffix after P_i. **3)** We balance the number of positive and negative labels at each prefix length in the training dataset. This is crucial, as otherwise there is an imbalance of positive labels towards the start of solutions

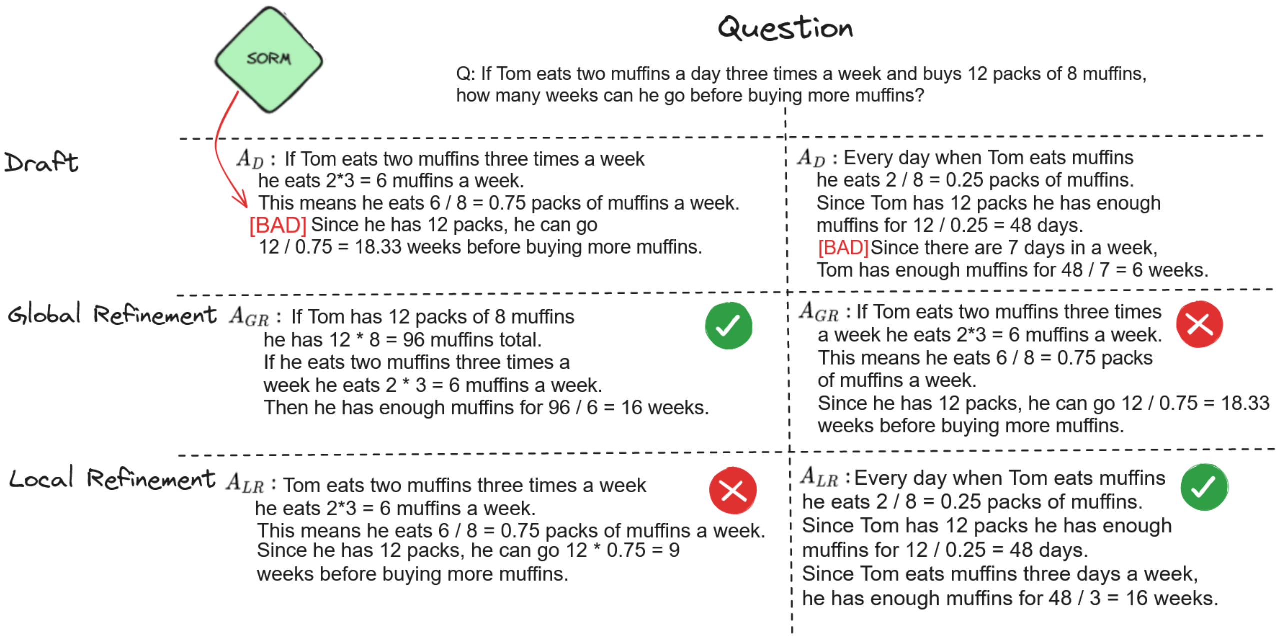

SORM
Question
Q: If Tom eats two muffins a day three times a week and buys 12 packs of 8 muffins, how many weeks can he go before buying more muffins?
Draft
A_D : If Tom eats two muffins three times a week he eats 2*3 = 6 muffins a week. This means he eats 6 / 8 = 0.75 packs of muffins a week. [BAD] Since he has 12 packs, he can go 12 / 0.75 = 18.33 weeks before buying more muffins.
A_D : Every day when Tom eats muffins he eats 2 / 8 = 0.25 packs of muffins. Since Tom has 12 packs he has enough muffins for 12 / 0.25 = 48 days. [BAD]Since there are 7 days in a week, Tom has enough muffins for 48 / 7 = 6 weeks.
Global Refinement
A_{GR} : If Tom has 12 packs of 8 muffins he has 12 * 8 = 96 muffins total. If he eats two muffins three times a week he eats 2 * 3 = 6 muffins a week. Then he has enough muffins for 96 / 6 = 16 weeks.
A_{GR} : If Tom eats two muffins three times a week he eats 2*3 = 6 muffins a week. This means he eats 6 / 8 = 0.75 packs of muffins a week. Since he has 12 packs, he can go 12 / 0.75 = 18.33 weeks before buying more muffins.
Local Refinement
A_{LR} : Tom eats two muffins three times a week he eats 2*3 = 6 muffins a week. This means he eats 6 / 8 = 0.75 packs of muffins a week. Since he has 12 packs, he can go 12 * 0.75 = 9 weeks before buying more muffins.
A_{LR} : Every day when Tom eats muffins he eats 2 / 8 = 0.25 packs of muffins. Since Tom has 12 packs he has enough muffins for 12 / 0.25 = 48 days. Since Tom eats muffins three days a week, he has enough muffins for 48 / 3 = 16 weeks.

Figure 2 Example of local and global refinements on a math word problem. **Left:** The local refinement does poorly with a student which struggles dividing by a fraction. Although all prior steps leading up to the fractional division are valid, the local refinement model is forced to either attempt the difficult operation again or choose the wrong operation entirely. In contrast, the global refinement model may attempt to solve the problem with an entirely new approach. **Right:** In this draft, the model is very close to the final answer, only making a simple mistake at the end. The local refinement is able to correct this simple mistake. In contrast, the global refinement must start from scratch.

and negative labels towards the end. This imbalance is easy for SORMs to exploit, leading to models which almost always predict a `positive` label in the first few steps a `negative` label towards the end.

As an additional baseline we consider the **Balanced ORM** which simply balances the number of positives and negatives per question in the ORM training dataset. This is done in an attempt to mitigate the overly pessimisstic behavior of the ORM described earlier.

Our SORM approximation is motivated by observations from concurrent work which shows our student π does not need to engage in too much exploration, i.e. sampling, to solve most problems sufficiently in distribution of pretraining data. This suggests rejection sampling to be capable of providing a decent approximation to the optimal policy. Additionally, the deterministic dynamics of the reasoning environment allows us to only sample once from the optimal policy π^* to compute V^* at a prefix P_i. This further reduces our sampling requirements, while also allowing us to conclude that if rejection sampling can solve the problem from a prefix P_i, then π^* will also solve the problem from P_i. Note of course rejection sampling will be weaker than π^*, resulting in the SORM being an under-approximation of V^*.

Step 3: Training refinement models

To train a local refinement model we need a dataset of the form (Q, A_D, A_R, E) where Q is a question, A_D is an initial draft, E labels the location of the first error in A_D indicating where to refine, and A_R is a refinement with the correct final answer. In pratice, E is communicated to the local refinement as a "[BAD]" token prefixing the incorrect step S_i in the draft. Then, at test time, we need a model predicting $p(E|Q, A_D)$ to localize errors in the draft. Conveniently, we explicitly train the SORM to predict the correctness of each step in A_D. Thus, to produce E we infer the SORM on all steps and return the index of the first step with predicted correctness below a threshold T. Further, **we can construct a refinement training dataset with error annotations using the SORM dataset**. Given an incorrect model rollout $A = (S_1, S_2, ..., S_L)$ we can locate step S_i as containing the first error by identifying $l_i = 0$ as the first zero label in the trace. We then pair A with a correct verifying trace T from the previous (correct) step S_{i-1}. This creates a training pair (A, T) where we label the first error in A as $E = i$. See Figure 2 for an example.

We construct a dataset for global refinement similarly using the ORM training dataset. This is done by pairing incorrect rollouts $A_{\text{incorrect}}$ with correct rollouts A_{correct} for the same question Q. This constructs a training tuple $(Q, A_{\text{incorrect}}, A_{\text{correct}})$. To maintain a format similar to local refinement, we put a $[BAD]$ token at the very start of the incorrect rollout. We combine both refinement datasets to train a model capable of both global and local refinement.

4.2 Evaluation

We construct a test set for both the ORM/SORM and refinement models by sampling the student model greedily on test questions Q from the task τ. For each benchmark this gives us a test set with prompts of the form (Q, A_D) where Q is the problem and A_D is an initial draft. For both benchmarks we refer to this as the (Q, D) test set. To generate intermediate step labels we use the same process as used to generate SORM training data. We evalaute the ORM and SORM on this test set by comparing their predictions to these ground truth labels.

To evaluate the global refinement performance we greedily infer the refiner on each (Q, A_D) sample and compare the resulting refinement A_{GR} to the ground truth. To evaluate the local refinement model we first annotate each (Q, A_D) pair with the location of its first error using the ORM or SORM. This forms a (Q, A_D, E) triplet which we use to greedily sample the local refiner.

For our best results, we propose to sample both a global refinement A_{GR} and a local refinement A_{LR} for a draft A_D and choose the best solution using the ORM reranker. This strategy stems from our observation that global and local refinements each solve complementary, partially non-overlapping subsets of problems the student initially fails on. Thus combining both refinements with the draft significantly expands the set of problems we can solve. Additionally, using the ORM to rerank refinements allows for a cleaner comparison against a best-of-three baseline from the draft-generating student π. See Figure 3 for a diagram of the evaluation pipeline.

We also highlight more exploratory work in the appendix. In the main body we consider only *process-based* local refinement, which relies on locating reasoning errors in a solution trace. One drawback of this approach is its agnosticism to the abilities of the student model doing refinement. Alternatively, we consider *value-based* refinement which relies on feedback identifying the step in a solution from which the model has the best chance of succeeding. A comparison to process-based refinement is done in appendix Section J. Additionally, in appendix Section C, we compare refinement training using expert iteration to other RL algorithms with various reward schemes.

5 Results

We evaluate our refinement pipeline on the GSM8K (Cobbe et al., 2021) and SVAMP (Patel et al., 2021) math word problem benchmarks. We fine-tune Llama-2 7B and 13B to produce all downstream models including the ORM, SORM, and refinement models. Note, the evaluation of each model size is self-contained, not utilizing any data or feedback from models of a different size. maj@1 model scores via greedy sampling will be used to evaluate model performance. Hyperparamters for each phase of training are supplied in Section A of the appendix.

5.1 Evaluting the ORM and SORM

SORMs are better than ORMs at evaluating intermediate answers: On GSM8K the SORM improves over the intermediate step accuracy of the ORM by up to 8% from 73% to 81% (See Table 1). This confirms the ORM does a reasonable job estimating intermediate step correctness but can still be improved, particularly for smaller models on a hard tasks like GSM8K. We'll see this difference in label accuracy also translates into a difference in refinement final accuracy, where it is critical for the ORM/SORM to reliably identify locations of

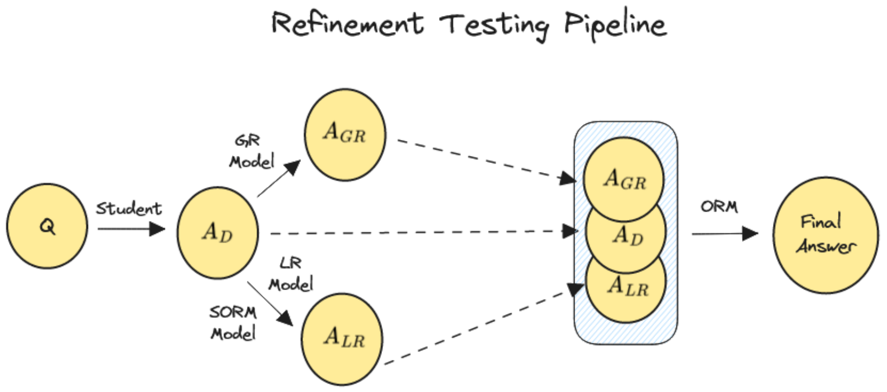


Figure 3 **Evaluation Pipeline** for global and local refinement models. We first sample a draft A_D from the student model then sample global and local refinements. The ORM is then used to determine which response to select as the final answer among these three candidate solutions.

	GSM8K		SVAMP	
	7B	13B	7B	13B
ORM	0.74	0.73	0.77	0.85
Balanced ORM	0.73	0.74	0.77	0.83
SORM	**0.79**	**0.81**	**0.78**	**0.87**

Table 1 Step-level accuracy of 7B/13B ORM and SORM on test set labels. Note: the test sets are well balanced with positive labels representing 45%-55% of samples. The SORM has better step level accuracy than ORM on the harder GSM8K benchmark but comparable step level accuracy on SVAMP.

	GSM8K		SVAMP	
	7B	13B	7B	13B
ORM	**0.82**	**0.85**	**0.75**	**0.82**
Balanced ORM	0.8	0.82	0.73	0.79
SORM	0.79	0.8	0.74	0.79

Table 2 Final answer accuracy of 7B/13B ORM and SORM on test set labels. Note: the test sets are well balanced with positive labels representing 45%-55% of samples. The ORM has better accuracy than the SORM at predicting final answer correctness.

mistakes. In comparison, the balanced ORM underperforms, having comparable intermediate accuracy to the ORM. This is despite qualitiatively appearing to fix the ORM's over-pessimism, as the balanced ORM assigns roughly 50% chance of success to all questions. We also examine the types of errors models make, finding the SORMs to have a balanced numbers of false positives and negatives when using a 0.5 as the classification threshold.

ORMs better approximate V^* on easier tasks: On SVAMP the ORM has better step accuracy than on GSM8K (see Table 1), particularly the 13B model. As a result the SORM offers less improvement. Most questions in GSM8K are relatively more difficult, requiring at least 4 steps to solve. In contrast, most questions in SVAMP require at most three key steps. This small number of steps likely makes it easier for the ORM to generalize. Additionally, the EI models trained on SVAMP reach on average 15% higher accuracy than the same sized model on GSM8K. This makes the base student model a closer approximation to π^* on SVAMP, making the ORM a closer approximation to V^*.

The importance of a strong data generating student π is further highlighted by the difference in accuracies between 7B and 13B models on SVAMP. The 7B student EI model gets an accuracy of 58%, whereas the 13B model gets an accuracy of 70%. Correspondingly, the 13B ORM model performs much better at on intermediate steps than the 7B model. Yet in contrast the 13B ORM on GSM8K performs slightly worse at intermediate steps than 7B. This is perhaps partially explained by the performance of the 13B EI student on GSM8K which only improves 5% over the 7B student.

ORMs are better than SORMs at evaluating final answers: Despite the SORM being generally better at predicting intermediate steps, it is slightly worse at predicting final answer correctness compared to the ORM. This is true for both benchmarks, with the 13B SORM on GSM8K lagging by 5% (See Table 2). However, part of this difference is likely due to statistical biases the ORM is able to exploit, improving final answer accuracy at the cost of over-pessimism. For example, if the problem involves division, the ORM knows the student is likely to fail and immediately predicts a low probability of success. In contrast the SORM is forced to be more optimistic, attempting to carefully examine the correctness of each intermediate step.

Unfortunately, the inaccuracy of the SORM as a final answer predictor also makes it slightly worse as a final answer reranker. For this reason we always use the ORM whenever reranking candidate drafts and refinements. A more detailed comparison of reranking accuracies on GSM8K is done in Figure 4. Note, this comparison is done using ORMs and SORMs derived from a student model trained using only supervised fine-tuning on GSM8K. Rerank accuracies are computed by sampling the student K times and scoring each rollout with the ranker. The rollout with the highest score is then chosen as the final answer.

Figure 4 also plots rerank accuracies for SORM models trained on data without additional postproccessing. The best performing SORM uses only consistent verifying rollouts and per-step balanced labels, justifying these as good postprocessing choices.

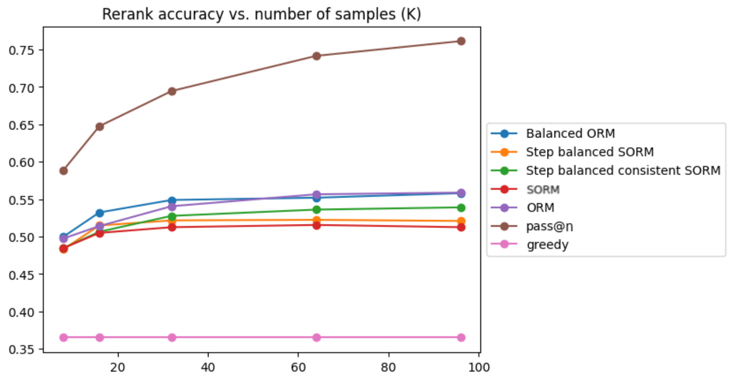


Figure 4 Plot of ORM, balanced ORM, and SORM rerank accuracies with the same SFT student (maj@1 = 0.36). Note: SORM by itself does not use balanced step labels or consistent verifiers as additional pre-processing steps as described in Section 4. When we add in both steps, reranking performance significantly improves to nearly match the ORM's performance.

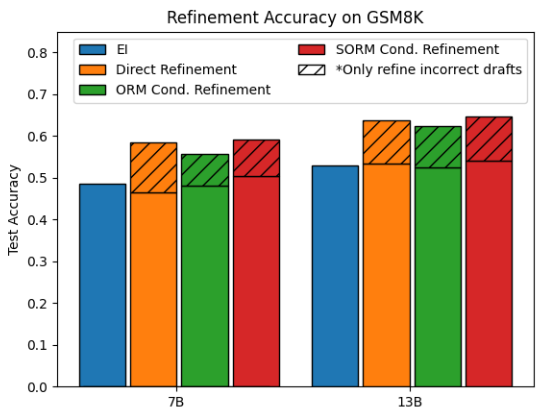
5.2 Evaluating global and local refinements

Now, with a better understanding of our SORMs' capabilities, we can apply them for refinement. Recall that to decide *when* to accept a refinement A_R we use the ORM as a reranker on the draft A_D and refinement A_R. When performing local refinement we can additionally use both the ORM and SORM to identify the location of the first mistake in A_D. For the ORM we do this by labeling the first step S_i such that $ORM(S_i) \leq T = 0.5$ where T is a threshold hyperparameter. We identify the first error analogously with the SORM. We report results on both GSM8K and SVAMP (Q, D) test sets in Figure 5. Note, we being evaluation without using the ORM as a reranker. This is done to confirm others' observations that refiners struggle knowing when to refine on their own.

Both global and local refinement models struggle with knowing when to refine: On both benchmarks global and local refinements show little improvement to overall model accuracy. GSM8K 7B global refinements even decreases overall accuracy, with the other models improving by at most 1%. The local refinements improve overall accuracy more, likely due to the presence of the "[BAD]" token indicating the location (and therefore presence) of the first mistake. This underscores the importance of an ORM for choosing when to refine an incorrect draft. We also note that bigger models produce better refinements.

Global and local refinements fix similar percentages of incorrect drafts: To understand how well our refiners perform when refinement is needed we also report results when applying refinement to only incorrect drafts from the test set in Figure 5. In this case both global and local refinements do much better, improving overall accuracy by an average of 10% on GSM8K and 8% on SVAMP. This demonstrates the refiners have learned how to refine, they simply often do not know when.

It is initially somewhat surprising global refinements are able to fix a similar percentage of drafts as local refinements. Local refinements receive extra information from E, presumably strictly improving performance over the global refiner. In reality, the provided E is noisy as it must be predicted by an imperfect ORM/SORM. We see that even the difference in label accuracy bewteen the ORM and SORM results in a nontrivial difference



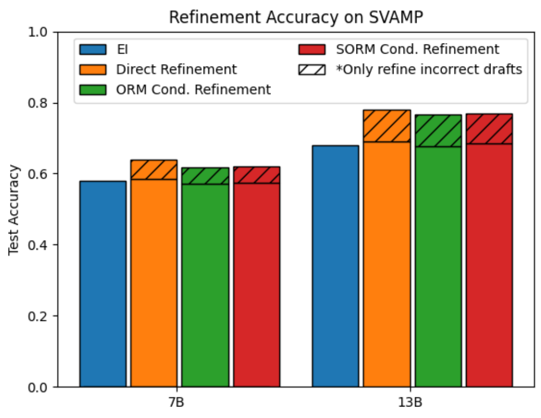


Figure 5 Refinement accuracies on GSM8K and SVAMP. All refinement models struggle identifying correct drafts which do not need refinement. Significant improvements are seen when only refining incorrect drafts.

	GSM8K 7B	GSM8K 13B	SVAMP 7B	SVAMP 13B
Global Refinement	0.203	0.281	0.14	0.255
Local Refinement + ORM	0.182	0.262	0.09	0.229
Local Refinement + SORM	0.211	0.283	0.11	0.237
Global Refinement + Local Refinement + ORM	0.252	0.384	0.173	0.35
Global Refinement + Local Refinement + SORM	**0.280**	**0.412**	**0.19**	**0.37**

Table 3 Refinement accuracy on incorrect model answers. Local refinement + SORM denotes using the SORM to highlight the first incorrect reasoning step for the local refinement model. We find refining both globally and locally with the SORM can fix up to 41% of problems the model previously failed.

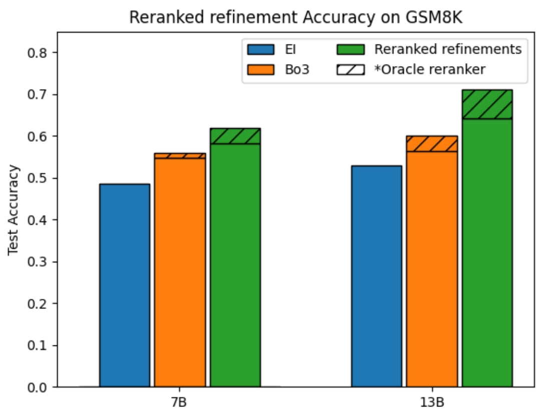
in refinement accuracy.

Additionally, global refinements have the advantage of optionally restarting a solution from scratch. A local refinement model is trained to reuse the prefix of a solution preceding a "[BAD]" token under the assumption this prefix has no errors. However, even if this prefix has valid reasoning, it may be a *low-value* solution path for the student. For example, a student who often fails to correctly divide may benefit from starting the problem from scratch in a way that doesn't require any use of division. global refinements can take advantage of this, whereas local refinements may be commited to valid reasoning with a low chance of successfully completing. See Figure 2 for examples illustrating this point.

Global and local refinements solve partially disjoint, complementary sets of problems: To better understand how global and local refinements compare we examine the overlap between the problems they correctly solve. The last two rows of Table 3 show that, **when combined, global and local refinements can fix 41% of incorrect GSM8K drafts** from the 13B student. Alone, global refinement and local refinement with the SORM fixes only 28% of problems. Yet, when taking the best of both types of refinement for the same question, we significantly improve performance across all combinations of benchmarks and model sizes. This shows local refinement is able to solve a large set of problems global refinement cannot, and vice versa. Best performance at test time can then be achieved if we have a way of selecting which of the two refinements is appropriate.

Fortunately, we can use the ORM as a reranker for exactly the task of choosing between global and local refinements. Additionally, we can consider the initial draft as a third possible option as a way of deciding if we want to refine at all. Figure 6 shows the results of reranking the draft, global, and local refinement for each question. Since we are effectively sampling three times, we include as a baseline the best of three (**Bo3**) samples from the EI student. We additionally report overall accuracy if we had a perfect reranker capable of always choosing the correct solution.

Reranking the draft + refinements improves over the draft accuracy by on average 8% across models and



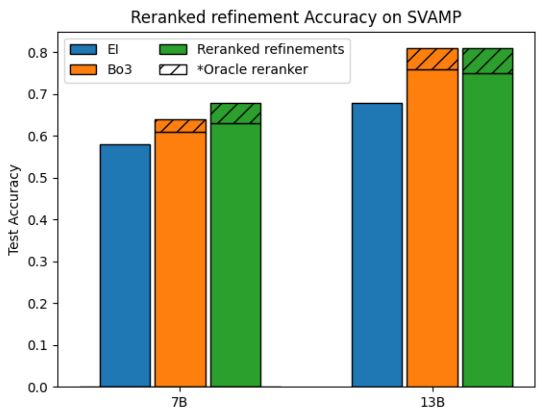


Figure 6 Accuracy of reranked refinements on all drafts compared to greedy and best of 3 samples from the student (Bo3) baselines. On GSM8K, reranking refinements using the ORM improves over the Bo3 baseline by up to 9% and up to 13% with a perfect reranker.

benchmarks. When comparing with the Bo3 baseline we still see significant improvements of around 8% on GSM8K. On SVAMP, reranked Bo3 is a much more competitive baseline, itself giving a large improvement over the draft accuracy. An even bigger improvement can be seen when using an oracle reranker, with the 13B refiner improving 11% over even Bo3 on GSM8K.

6 Conclusion and Future Work

In this paper we study the use of reward models for both identifying *when* to refine and *where* to refine LLM reasoning. We found ORM models generalize to some extent to evaluating the accuracy of intermediate steps on easier reasoning tasks but struggle on harder tasks where the training data generating policy π is further from π^*. We then propose to approximate the optimal policy π^* via rejection sampling and post-processing, allowing us to generate training labels for intermediate steps S_i used to train SORM models. We find the SORM generalizes better on intermediate test steps than the ORM, but at the cost of final answer accuracy. We then reused the ORM/SORM training data to train a global/local refinement models. We found each type of refinement strategy helped solve a largely unique set of problems, allowing us to combine both via ORM reranking for best performance.

Future work can be classified as either: 1) improving the reliability and verbosity of local error critiques E by providing more information on *how* to refine or 2) augmenting the type of information local refiners use to generate correct solutions. Our study of both ORMs and SORMs reveals large room for improvement when verifying step level reasoning. Allowing verifier models to generate chains of thought appears to offer some benefit (Dhuliawala et al., 2023). Further augmenting verifying CoT with tools (Zhou et al., 2023) allows GPT-4 to effectively solve MATH (Hendrycks et al., 2021a). But it remains unclear how much GPT-4 relies on the tool to solve the problem versus actually uses the tool to augment its own understanding of *why* a step is wrong.

Another promising direction treats iterative refinement as a form of *in-context exploration* similar in spirit to ideas from algorithm distillation (Laskin et al., 2022). Here, the aim is to minimize the number of in-context model rollouts needed to figure out *how* to refine. This also closely relates to work aiming to augment the exploration abilities of SOTA LLMs, a direction we believe is critical to future success. The right iterative local self-refinement strategies might hopefully allow models to access complex behaviors previously inaccessible with naieve iid repeated sampling.

References

Afra Feyza Akyurek, Ekin Akyürek, Aman Madaan, A. Kalyan, Peter Clark, D. Wijaya, and Niket Tandon. Rl4f: Generating natural language feedback with reinforcement learning for repairing model outputs. In *Annual Meeting of the Association for Computational Linguistics*, 2023. URL https://api.semanticscholar.org/CorpusID:258685337.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program synthesis with large language models. *ArXiv*, abs/2108.07732, 2021. URL https://api.semanticscholar.org/CorpusID:237142385.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, John Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, E Perez, Jamie Kerr, Jared Mueller, Jeff Ladish, J Landau, Kamal Ndousse, Kamilė Lukoiūtė, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noem'i Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, T. J. Henighan, Tristan Hume, Sam Bowman, Zac Hatfield-Dodds, Benjamin Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom B. Brown, and Jared Kaplan. Constitutional ai: Harmlessness from ai feedback. *ArXiv*, abs/2212.08073, 2022. URL https://api.semanticscholar.org/CorpusID:254823489.

Maciej Besta, Nils Blach, Ale Kubek, Robert Gerstenberger, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Michal Podstawski, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. *ArXiv*, abs/2308.09687, 2023. URL https://api.semanticscholar.org/CorpusID:261030303.

Angelica Chen. Improving code generation by training with natural language feedback. *ArXiv*, abs/2303.16749, 2023. URL https://api.semanticscholar.org/CorpusID:257804798.

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

François Chollet. On the measure of intelligence. *ArXiv*, abs/1911.01547, 2019. URL https://api.semanticscholar.org/CorpusID:207870692.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam M. Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Benton C. Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathleen S. Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways. *J. Mach. Learn. Res.*, 24:240:1–240:113, 2022. URL https://api.semanticscholar.org/CorpusID:247951931.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *ArXiv*, abs/2110.14168, 2021. URL https://api.semanticscholar.org/CorpusID:239998651.

Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. Chain-of-verification reduces hallucination in large language models. *ArXiv*, abs/2309.11495, 2023. URL https://api.semanticscholar.org/CorpusID:262062565.

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, Jason Phang, Laria Reynolds, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, September 2021. URL https://doi.org/10.5281/zenodo.5371628.

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*, abs/2305.11738, 2023. URL https://api.semanticscholar.org/CorpusID:258823123.

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

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Xiaodong Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *ArXiv*, abs/2103.03874, 2021b. URL https://api.semanticscholar.org/CorpusID:232134851.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. *ArXiv*, abs/2310.01798, 2023. URL https://api.semanticscholar.org/CorpusID:263609132.

Michael Laskin, Luyu Wang, Junhyuk Oh, Emilio Parisotto, Stephen Spencer, Richie Steigerwald, DJ Strouse, Steven Stenberg Hansen, Angelos Filos, Ethan Brooks, Maxime Gazeau, Himanshu Sahni, Satinder Singh, and Volodymyr Mnih. In-context reinforcement learning with algorithm distillation. *ArXiv*, abs/2210.14215, 2022. URL https://api.semanticscholar.org/CorpusID:253107613.

Ariel N. Lee, Cole J. Hunter, and Nataniel Ruiz. Platypus: Quick, cheap, and powerful refinement of llms. *ArXiv*, abs/2308.07317, 2023. URL https://api.semanticscholar.org/CorpusID:260886870.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, B. Chen, Jian-Guang Lou, and Weizhu Chen. Making language models better reasoners with step-aware verifier. In *Annual Meeting of the Association for Computational Linguistics*, 2022. URL https://api.semanticscholar.org/CorpusID:259370847.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Cosgrove, Christopher D. Manning, Christopher Ré, Diana Acosta-Navas, Drew A. Hudson, E. Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue Wang, Keshav Santhanam, Laurel J. Orr, Lucia Zheng, Mert Yüksekgönül, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri S. Chatterji, O. Khattab, Peter Henderson, Qian Huang, Ryan Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. Holistic evaluation of language models. *ArXiv*, abs/2211.09110, 2022. URL https://api.semanticscholar.org/CorpusID:263423935.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *ArXiv*, abs/2305.20050, 2023. URL https://api.semanticscholar.org/CorpusID:258987659.

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, 2023.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann André LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. 2023. URL https://api.semanticscholar.org/CorpusID:265351664.

Swaroop Mishra, Pan Lu, and A. Kalyan. Lila: A unified benchmark for mathematical reasoning. 2022. URL https://api.semanticscholar.org/CorpusID:257405677.

OpenAI. Gpt-4 technical report. *ArXiv*, abs/2303.08774, 2023. URL https://api.semanticscholar.org/CorpusID:257532815.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke E. Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Francis Christiano, Jan Leike, and Ryan J. Lowe. Training language models to follow instructions with human feedback. *ArXiv*, abs/2203.02155, 2022. URL https://api.semanticscholar.org/CorpusID:246426909.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are nlp models really able to solve simple math word problems?, 2021.

Debjit Paul, Mete Ismayilzada, Maxime Peyrard, Beatriz Borges, Antoine Bosselut, Robert West, and Boi Faltings. Refiner: Reasoning feedback on intermediate representations. *ArXiv*, abs/2304.01904, 2023. URL https://api.semanticscholar.org/CorpusID:257921623.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *ArXiv*, abs/2311.12022, 2023. URL https://api.semanticscholar.org/CorpusID:265295009.

William Saunders, Catherine Yeh, Jeff Wu, Steven Bills, Ouyang Long, Jonathan Ward, and Jan Leike. Self-critiquing models for assisting human evaluators. *ArXiv*, abs/2206.05802, 2022. URL https://api.semanticscholar.org/CorpusID:249626555.

Tomohiro Sawada, Daniel Paleka, Alex Havrilla, Pranav Tadepalli, Paula Vidas, Alexander Kranias, John J. Nay, Kshitij Gupta, and Aran Komatsuzaki. Arb: Advanced reasoning benchmark for large language models. *ArXiv*, abs/2307.13692, 2023. URL https://api.semanticscholar.org/CorpusID:260155126.

Timo Schick, Jane Dwivedi-Yu, Zhengbao Jiang, Fabio Petroni, Patrick Lewis, Gautier Izacard, Qingfei You, Christoforos Nalmpantis, Edouard Grave, and Sebastian Riedel. Peer: A collaborative language model. *ArXiv*, abs/2208.11663, 2022. URL https://api.semanticscholar.org/CorpusID:251765117.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, L. Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *ArXiv*, abs/1712.01815, 2017. URL https://api.semanticscholar.org/CorpusID:33081038.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R. Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, Agnieszka Kluska, Aitor Lewkowycz, Akshat Agarwal, Alethea Power, Alex Ray, Alex Warstadt, Alexander W. Kocurek, Ali Safaya, Ali Tazarv, Alice Xiang, Alicia Parrish, Allen Nie, Aman Hussain, Amanda Askell, Amanda Dsouza, Ambrose Slone, Ameet Annasaheb Rahane, Anantharaman S. Iyer, Anders Andreassen, Andrea Madotto, Andrea Santilli, Andreas Stuhlmuller, Andrew M. Dai, Andrew La, Andrew Kyle Lampinen, Andy Zou, Angela Jiang, Angelica Chen, Anh Vuong, Animesh Gupta, Anna Gottardi, Antonio Norelli, Anu Venkatesh, Arash Gholamidavoodi, Arfa Tabassum, Arul Menezes, Arun Kirubarajan, Asher Mullokandov, Ashish Sabharwal, Austin Herrick, Avia Efrat, Aykut Erdem, Ayla Karakacs, B. Ryan Roberts, Bao Sheng Loe, Barret Zoph, Bartlomiej Bojanowski, Batuhan Ozyurt, Behnam Hedayatnia, Behnam Neyshabur, Benjamin Inden, Benno Stein, Berk Ekmekci, Bill Yuchen Lin, Blake Stephen Howald, Bryan Orinion, Cameron Diao, Cameron Dour, Catherine Stinson, Cedrick Argueta, C'esar Ferri Ram'irez, Chandan Singh, Charles Rathkopf, Chenlin Meng, Chitta Baral, Chiyu Wu, Chris Callison-Burch, Chris Waites, Christian Voigt, Christopher D. Manning, Christopher Potts, Cindy Ramirez, Clara E. Rivera, Clemencia Siro, Colin Raffel, Courtney Ashcraft, Cristina Garbacea, Damien Sileo, Daniel H Garrette, Dan Hendrycks, Dan Kilman, Dan Roth, Daniel Freeman, Daniel Khashabi, Daniel Levy, Daniel Mosegu'i Gonz'alez, Danielle R. Perszyk, Danny Hernandez, Danqi Chen, Daphne Ippolito, Dar Gilboa, David Dohan, David Drakard, David Jurgens, Debajyoti Datta, Deep Ganguli, Denis Emelin, Denis Kleyko, Deniz Yuret, Derek Chen, Derek Tam, Dieuwke Hupkes, Diganta Misra, Dilyar Buzan, Dimitri Coelho Mollo, Diyi Yang, Dong-Ho Lee, Dylan Schrader, Ekaterina Shutova, Ekin Dogus Cubuk, Elad Segal, Eleanor Hagerman, Elizabeth Barnes, Elizabeth P. Donoway, Ellie Pavlick, Emanuele Rodolà, Emma Lam, Eric Chu, Eric Tang, Erkut Erdem, Ernie Chang, Ethan A. Chi, Ethan Dyer, Ethan J. Jerzak, Ethan Kim, Eunice Engefu Manyasi, Evgenii Zheltonozhskii, Fanyue Xia, Fatemeh Siar, Fernando Mart'inez-Plumed, Francesca Happ'e, François Chollet, Frieda Rong, Gaurav Mishra, Genta Indra Winata, Gerard de Melo, Germán Kruszewski, Giambattista Parascandolo, Giorgio Mariani, Gloria Wang, Gonzalo Jaimovitch-L'opez, Gregor Betz, Guy Gur-Ari, Hana Galijasevic, Hannah Kim, Hannah Rashkin, Hannaneh Hajishirzi, Harsh Mehta, Hayden Bogar, Henry Shevlin, Hinrich Schutze, Hiromu Yakura, Hongming Zhang, Hugh Mee Wong, Ian Ng, Isaac Noble, Jaap Jumelet, Jack Geissinger, John Kernion, Jacob Hilton, Jaehoon Lee, Jaime Fernández Fisac, James B. Simon, James Koppel, James Zheng, James Zou, Jan Koco'n, Jana Thompson, Janelle Wingfield, Jared Kaplan, Jarema Radom, Jascha Narain Sohl-Dickstein, Jason Phang, Jason Wei, Jason Yosinski, Jekaterina Novikova, Jelle Bosscher, Jennifer Marsh, Jeremy Kim, Jeroen Taal, Jesse Engel, Jesujoba Oluwadara Alabi, Jiacheng Xu, Jiaming Song, Jillian Tang, Jane W Waweru, John Burden, John Miller, John U. Balis, Jonathan Batchelder, Jonathan Berant, Jorg Frohberg, Jos Rozen, José Hernández-Orallo, Joseph Boudeman, Joseph Guerr, Joseph Jones, Joshua Tenenbaum, Joshua S. Rule, Joyce Chua, Kamil Kanclerz, Karen Livescu, Karl Krauth, Karthik Gopalakrishnan, Katerina Ignatyeva, Katja Markert, Kaustubh D. Dhole, Kevin Gimpel, Kevin Omondi, Kory Wallace Mathewson, Kristen Chiafullo, Ksenia Shkaruta, Kumar Shridhar, Kyle McDonell, Kyle Richardson, Laria Reynolds, Leo Gao, Li Zhang, Liam Dugan, Lianhui Qin, Lidia Contreras-Ochando, Louis-Philippe Morency, Luca Moschella, Luca Lam, Lucy Noble, Ludwig Schmidt, Luheng He, Luis Oliveros Col'on, Luke Metz, Lutfi Kerem cSenel, Maarten Bosma, Maarten Sap, Maartje ter Hoeve, Maheen Farooqi, Manaal Faruqui, Mantas Mazeika, Marco Baturan, Marco Marelli, Marco Maru, Maria Jose Ram'irez Quintana, Marie Tolkiehn, Mario Giulianelli, Martha Lewis, Martin Potthast, Matthew L. Leavitt, Matthias Hagen, M'aty'as Schubert, Medina Baitemirova, Melody Arnaud, Melvin Andrew McElrath, Michael A. Yee, Michael Cohen, Michael Gu, Michael Ivanitskiy, Michael Starritt, Michael Strube, Michal Swkedrowski, Michele Bevilacqua, Michihiro Yasunaga, Mihir Kale, Mike Cain, Mimee Xu, Mirac Suzgun, Mitch Walker, Monica Tiwari, Mohit Bansal, Moin Aminnaseri, Mor Geva, Mozhdeh Gheini, T MukundVarma, Nanyun Peng, Nathan A. Chi, Nayeon Lee, Neta Gur-Ari Krakover, Nicholas Cameron, Nicholas Roberts, Nick Doiron, Nicole Martinez, Nikita Nangia, Niklas Deckers, Niklas Muennighoff, Nitish Shirish Keskar, Niveditha Iyer, Noah Constant, Noah Fiedel, Nuan Wen, Oliver Zhang, Omar Agha, Omar Elbaghdadi, Omer Levy, Owain Evans, Pablo Antonio Moreno Casares, Parth Doshi, Pascale Fung, Paul Pu Liang, Paul Vicol, Pegah Alipoormolabashi, Peiyuan Liao, Percy Liang, Peter Chang, Peter Eckersley, Phu Mon Htut, Pi-Bei Hwang, P. Milkowski, Piyush S. Patil, Pouya Pezeshkpour, Priti Oli, Qiaozhu Mei, Qing Lyu, Qinlang Chen, Rabin Banjade, Rachel Etta Rudolph, Raefer Gabriel, Rahel Habacker, Ramon Risco, Raphael Milliere, Rhythm Garg, Richard Barnes, Rif A. Saurous, Riku Arakawa, Robbe Raymaekers, Robert Frank, Rohan Sikand, Roman Novak, Roman Sitelew, Ronan Lebras, Rosanne Liu, Rowan Jacobs, Rui Zhang, Ruslan Salakhutdinov, Ryan Chi, Ryan Lee, Ryan Stovall, Ryan Teehan, Rylan Yang, Sahib Singh, Saif M. Mohammad, Sajant Anand, Sam Dillavou, Sam Shleifer, Sam Wiseman, Samuel Gruetter, Samuel R. Bowman, Samuel S. Schoenholz, Sanghyun Han, Sanjeev Kwatra, Sarah A. Rous, Sarik Ghazarian, Sayan Ghosh, Sean Casey, Sebastian Bischoff, Sebastian Gehrmann, Sebastian Schuster, Sepideh Sadeghi, Shadi S. Hamdan, Sharon Zhou, Shashank Srivastava, Sherry Shi, Shikhar Singh, Shima Asaadi, Shixiang Shane Gu, Shubh Pachchigar, Shubham Toshniwal, Shyam Upadhyay, Shyamolima Debnath, Siamak Shakeri, Simon Thormeyer, Simone Melzi, Siva Reddy, Sneha Priscilla Makini, Soo-Hwan Lee, Spencer Torene, Sriharsha Hatwar, Stanislas Dehaene, Stefan Divic, Stefano Ermon, Stella Biderman, Stephanie Lin, Stephen Prasad, Steven T Piantadosi, Stuart M. Shieber, Summer Misherghi, Svetlana Kiritchenko, Swaroop Mishra, Tal Linzen, Tal Schuster, Tao Li, Tao Yu, Tariq Ali, Tatsunori Hashimoto, Te-Lin Wu, Theo Desbordes, Theodore Rothschild, Thomas Phan, Tianle Wang, Tiberius Nkinyili, Timo Schick, Timofei Kornev, Titus Tunduny, Tobias Gerstenberg, Trenton Chang, Trishala Neeraj, Tushar Khot, Tyler Shultz, Uri Shaham, Vedant Misra, Vera Demberg, Victoria Nyamai, Vikas Raunak, Vinay Venkatesh Ramasesh, Vinay Uday Prabhu, Vishakh Padmakumar, Vivek Srikumar, William Fedus, William Saunders, William Zhang, Wout Vossen, Xiang Ren, Xiaoyu Tong, Xinran Zhao, Xinyi Wu, Xudong Shen, Yadollah Yaghoobzadeh, Yair Lakretz, Yangqiu Song, Yasaman Bahri, Yejin Choi, Yichi Yang, Yiding Hao, Yifu Chen, Yonatan Belinkov, Yu Hou, Yu Hou, Yuntao Bai, Zachary Seid, Zhuoye Zhao, Zi Fu Wang, Zijie J. Wang, Zirui Wang, and Ziyi Wu. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. 2022. URL https://api.semanticscholar.org/CorpusID:263625818.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, L. Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback. *ArXiv*, abs/2211.14275, 2022. URL https://api.semanticscholar.org/CorpusID:254017497.

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. *ArXiv*, abs/2312.08935, 2023a. URL https://api.semanticscholar.org/CorpusID:266209760.

Tianlu Wang, Ping Yu, Xiaoqing Ellen Tan, Sean O'Brien, Ramakanth Pasunuru, Jane Dwivedi-Yu, Olga Golovneva, Luke Zettlemoyer, Maryam Fazel-Zarandi, and Asli Celikyilmaz. Shepherd: A critic for language model generation, 2023b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Huai hsin Chi, F. Xia, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *ArXiv*, abs/2201.11903, 2022.

Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. Generating sequences by learning to self-correct. *ArXiv*, abs/2211.00053, 2022. URL https://api.semanticscholar.org/CorpusID:253244506.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *ArXiv*, abs/2210.03629, 2022. URL https://api.semanticscholar.org/CorpusID:252762395.

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. *ArXiv*, abs/2305.10601, 2023a. URL https://api.semanticscholar.org/CorpusID:258762525.

Weiran Yao, Shelby Heinecke, Juan Carlos Niebles, Zhiwei Liu, Yihao Feng, Le Xue, Rithesh Murthy, Zeyuan Chen, Jianguo Zhang, Devansh Arpit, Ran Xu, Phi Thi Mui, Haiquan Wang, Caiming Xiong, and Silvio Savarese. Retroformer: Retrospective large language agents with policy gradient optimization. *ArXiv*, abs/2308.02151, 2023b. URL https://api.semanticscholar.org/CorpusID:260611249.

Aojun Zhou, Ke Wang, Zimu Lu, Weikang Shi, Sichun Luo, Zipeng Qin, Shaoqing Lu, Anya Jia, Linqi Song, Mingjie Zhan, and Hongsheng Li. Solving challenging math word problems using gpt-4 code interpreter with code-based self-verification. *ArXiv*, abs/2308.07921, 2023. URL https://api.semanticscholar.org/CorpusID:260900008.

Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

	Expert Iteration	(S)ORM	Refiners
Epochs	4	1	1
max lr	2e-5	2e-6	2e-5
min lr	2e-7	2e-7	2e-7
Batch size	128	256	128

Table 4 Hyperparameters for all training jobs. A cosine decay lr schedule is used in all cases.

	maj@1	maj@96	Rerank@96	pass@96
GSM8K				
SFT 7B	0.41	0.47	0.54	0.72
SFT 13B	0.48	0.55	0.68	0.84
EI_2 7B	0.485	0.55	0.64	0.8
EI_2 13B	0.53	0.59	0.71	0.88
SVAMP				
EI_5 7B	0.58	0.6	0.62	0.70
EI_5 13B	0.69	0.75	0.78	0.93

Table 5 Performance metrics for EI fine-tuned models on GSM8K and SVAMP. The subscript n denotes the number of rounds of expert iteration until convergence of the maj@1 score. Reranking done via an ORM trained with samples from the model being reranked.

A Hyperparamters

See Table 4 for a list of training hyperparameters used in each training job.

B RL for Reasoning

In order to start from the best student possible we RL fine-tune the base-model using Expert Iteration. See Table 5 for maj@1 (greedy), maj@96, Rerank@96 and pass@96 scores for EI fine-tuned models.

C RL for (global) refinement

Setup: We compare the utility of PPO versus EI for refinement on the GSM8K benchmark. To train EI models we sample the SFT^2 model trained in Section **??** $K = 96$ times per prompt in the train set. We then pair together all incorrect solutions A_{wrong} with correct solutions A_{correct} for a fixed question Q to form training tuples $(Q, A_{\text{wrong}}, A_{\text{correct}})$. We then fine-tune from Llama-2 7B to predict $p(A_{\text{correct}}|Q, A_{\text{wrong}})$ with the standard cross-entropy loss for a single epoch. We use an initial learning rate of 5e-5 decaying to 5e-7.

We initialize the PPO model from the SFT^2 checkpoint used in Section 2 above and use the same PPO parameters as when fine-tuning from the SFT checkpoint on GSM8K. A single example, included in the appendix for reference, is used to prompt the model for refinement. During training the student model is given a question Q and draft D, where the draft is generated by the SFT model, and tasked with generating a refinement R. We give $R = \mathbf{1}_{\texttt{is\_correct(R)}} - \mathbf{1}_{\texttt{is\_correct(D)}}$ as a sparse reward at the end of the rollout. We additionally experimented with mixing in scores from an ORM, giving a final reward of $R = max(\mathbf{1}_{\texttt{is\_correct(R)}} - \mathbf{1}_{\texttt{is\_correct(D)}}, ORM(R) - ORM(D))$.

C.0.1 Results for global refinement

We evaluate all refinement models on a test set with questions from GSM8K test and drafts generated by SFT^2. Results are reported in Table 6. Sampling at test time is done greedily, so only maj@1 accuracy is reported. We additionally report a 1-shot prompted Llama-2 7B as a baseline.

	Accuracy
SFT	0.36
Prompted	0.15
PPO	0.36
ORM PPO	0.36
EI	0.39

Table 6 global refinement accuracies for prompted, PPO, and EI models. Note, maj@1 accuracy is reported for entire test set containing both correct and incorrect SFT generated drafts.

All models struggle learning when to refine Our best refinement model is the EI model. However, EI improves over the SFT baseline by only 3%, with PPO showing no improvement at all. This is because both models struggle correctly deciding when to refine. Often the EI model chooses to incorrectly refine correct drafts. The prompted pretrained model does even worse, having not been trained on GSM8K and thus struggling with when to refine and how to produce correct alternative refinements.

The PPO model collapses to simply returning the draft final answer, at least avoiding any negative rewards from incorrectly refining a correct draft. The prompted baseline also exhibits this copying behavior, accounting for the majority of its nontrivial accuracy. We experiment with alternative RL setups for preventing this degenerate behavior by using the ORM as an additional reward, removing the penalty for refining correct drafts, and/or having the student generate both the draft D and refinement R. However, in all cases the model's copy bias continues to limit exploration, causing a collapse of the refinement to the initial draft.

Discussion: The results above highlight several failure modes. Firstly, models struggle with determining **when to refine**, often defaulting to never refining at all. Secondly, when the model does correctly choose where to refine it still struggles with knowing **where to refine**. Finally, even knowing when and where to refine, the model still must decide **how** to refine.

In order to improve model refinement capability we propose to decompose the problem by using unique models to solve each failure mode. Fortunately, deciding when to refine can naturally be handled by the ORM which is explicitly trained to predict when a final answer is correct. Additionally, when doing local refinement, we can use the SORM to identify where to refine. This now only requires the refinement model we train to decide how to refine, making the task significantly easier.

D Misc. Objectives for Reranking

In Lightman et al. (2023) the PRM is used for reranking by estimating $P(\mathsf{Good}|S_i)$ for each step S_i and taking the product. Inspired by this, we experimented with a number of different weightings for SORM intermediate step estimates when doing final answer reranking. For a solution of the form $S = (S_1, ..., S_L)$ these heuristics included:

1. Final: $ORM(S_L)$
2. Mean: $\frac{1}{L}\sum_{i=1}^{L} ORM(S_i)$
3. Weighted mean: $\sum_{i=1}^{L} \frac{1}{L-i-1} ORM(S_i)$
4. Min: $min_{i\in[L]} ORM(S_i)$
5. Product: $\prod_{i=1}^{L} ORM(S_i)$
6. Penultimate mean: $\frac{ORM(S_{L-1})-ORM(S_L)}{2}$

The results are plotted in Figure 7. Overall using only the final ORM estimates gives the best reranking accuracy with the penultimate mean coming in at a close second. The weighted mean significantly underperforms all other strategies, even taking the minimum ORM estimate.

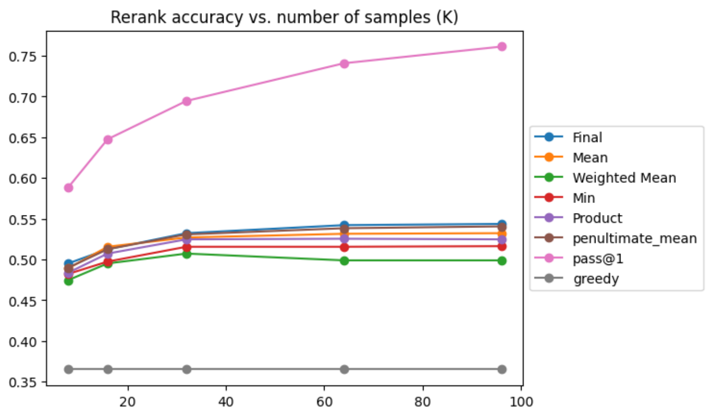


Figure 7 Comparison of heuristics for determining final rerank score.

	EI Test Accuracy	SFT Test Accuracy
ORM_{EI}	0.64	0.51
ORM_{SFT}	0.58	0.56

Table 7 Evaluation ORM_{EI} and ORM_{SFT} cross-generalization.

E ORM and SORM extra-model generalization

Both ORMs and SORMs exhibit signs of overfit to the data generating student π. When evaluated on GSM8K train set a 7B ORM model incorrectly classifies 42% of correct solutions as incorrect. To examine this more closely we take two base student models, EI (trained with expert iteration) and SFT (supervised fine-tuning), and use both to generate training data for ORM_{EI} and ORM_{SFT} respectively. we then evaluate both ORMs on test sets generated by each model. Results are reported in Table 7. We find both ORMs underperform on the test dataset generated by the opposite student model.

F Contrastive vs. Classifier RMs

Both the ORM and SORM are trained as classifiers to predict the probability of a `good` label at each intermediate step S_i. However, in RLHF there are only preference comparisons over solutions. So the RLHF reward model is often trained via a contrastive loss $-log(\sigma(RM(y_{\texttt{good}}) - RM(y_{\texttt{bad}}))$. We explore the use of a contrastive reward model in the reasoning setting, comparing reranking performance with the ORM. Training data is sampled from a 7B SFT student model with K = 96 rollouts per training prompt at a temperature T = 1.0. We assign ORM step labels in the usual way, setting $l_i = 1$ at a step S_i if $l_L = 1$ and otherwise $l_i = 0$. To construct preference pairs we select the maximal equal number of positive and negative solutions for the same prompt and form pairs $(S_{\texttt{good}}, S_{\texttt{bad}})$ with no solution repeated. Only the contrastive loss on the final token is backpropagated. We then rerank solutions on the test set using scores assigned by both the classifier ORM and contrastive ORM. We find the classifier ORM gets 0.56 rerank accuracy whereas the contrastive ORM gets 0.47, **suggesting the classifier to be a strictly better reranker**.

G Accuracy of the SORM data generating method

The SORM data generation process will suffer from both false positives and false negatives. False positives may occur when the student model solves the problem incorrectly but gets the right final answer. False negatives will occur when the rejection sampled student is simply unable to solve the problem from a prefix P_i desipte the prefix being logically valid. In order to verify the correctness of the SORM step-level verification process we hand-label several model generated solutions on GSM8K and compute how well the our ground truth labels align with the generated labels. Over $n = 64$ and a total of 257 steps we find the SORM data labels agree with our ground truth 94% of the time.

H Mixing other sources of PRM data

We additionally experiment with mixing in PRM data on the MATH dataset from Lightman et al. (2023). We train Llama-2 7B to predict `negative`, `neutral` and `good` labels for each step, with "bad" steps being incorrect, "neutral" steps neither making forward nor backward progress, and "good" steps being correct and useful to solving the problem. The resulting PRM gets 0.91 accuracy on the MATH PRM test set. However, we find the PRM transfers poorly as a final answer correctness predictor, getting only 0.58 accuracy on an EI generated test set.

I Self-supervised learning with the SORM

It is likely the SORM dataset generation process is fairly noisy. Low quality samples will directly impact the performance of the downstream SORM, making it critical to improve dataset quality. In an attempt to remove noisy outliers we filtered version of the SORM dataset via SORM self-supervsion. For each training pair (Q, P_i, l_i), where Q is the question, $P_i = (S_1, ..., S_i)$ is a solution prefix with i steps, and l_i is the correctness label, we apply $SORM((Q, P_i))$. This generates a self-supervised label $l_i' = \mathbf{1}_{SORM((Q,P_i))>0.5}$. We then filter out all training samples with $l_i' \neq l_i$.

We filter the SORM dataset with a SORM checkpoint trained for 1 epoch and another trained for 2 epochs. The first model, denoted as SORM_1, has 75% accuracy on the SORM test set but 91% on the SORM train set. SORM_2 gets 78% test but 95% on the train set. It could be SORM_2 is overfit to the train set, so we train new SORM models on both filtered datasets. SORM_1', trained on SORM data filtered with SORM_1, gets 79% accuracy. SORM_2', trained on SORM data filtered with SORM_2, gets the same.

J Value refinement

The local refinement strategy employed in the main body of this work uses critiques attempting to locate steps with logical errors. This can be interpreted as a type of *process-based refinement* which which gives feedback agnostic to the abilities of the refinement model. More sophisticated forms of feedback might take the underlying capabilities of the model into account, maximizing the chances of this particular student succeeding.

One alternative refinement strategy which gives student specific feedback is *value-based refinement*. A value-based refinement model receives feedback in the form of a "[BAD]" token at the step in a solution with the highest value for the model. Recall the value of a step S_i is the probability the model gets the correct answer from S_i. Note this is not the same thing as process-based feedback as the step S_{i+1} after the highest value step S_i may not necessarily contain an error. Instead, S_{i+1} may attempt to solve the problem is a difficult way, for example using division with a model which struggles dividing correctly.

Training a value-based refinement model Further recall the ORM directly estimates the value function V^π of its data generating policy π such that $ORM(P_i) \approx V^\pi(S_i)$ for a prefix with $P_i = (S_1, ..., S_i)$. Given a student model π on a reasoning task τ we generate an ORM training set $\mathcal{D}_{ORM}$ by sampling each prompt in τ_{train} $K = 96$ times. We train the ORM as a classifier, setting an intermediate step label $l_i = l_L$ where $l_L = \texttt{is\_correct(S)}$.

To construct the value based refinement dataset $\mathcal{D}_{\text{vrefine}}$ we start by reusing the SORM dataset $\mathcal{D}_{\text{SORM}}$ generated as above using policy π and rejection sampling. For a sample $S = (S_1, ..., S_L)$ we identify the highest value step S_i by choosing the step with the most correct verifying rollouts v_i^j. We then select one of the verifying rollouts whose first step differs from S_{i+1} as the improving refinement R. This forms a value-refinement training pair (Q, S, R, C) where C is a "[BAD]" token inserted before step S_{i+1} in S. We then train a value-based local refinement model by minimizing $p(R|Q, S, C)$ with the standard cross-entropy loss.

In practice we generate all downstream models and datasets using LLama-2 7B EI on GSM8K as π.

Results: To evaluate we use the 7B EI model trained on GSM8K from our best SFT checkpoint. We greedily sample solution drafts on the GSM8K test set, forming a (Q, D) test set for the value-based local refinement model. We then label the highest value step S_i of each draft using the ORM, placing the "[BAD]" token as a prefix to S_{i+1}.

We evaluate only on incorrect drafts, comparing directly the performance of process-based SORM refinement. Value-based refinement fixes 14% of incorrect drafts whereas the SORM baseline fixes 21%. Surprisingly, even global refinement outperforms value-based refinement by 6%. This again take this to suggest intermediate ORM estimates are fairly noisy on test data.