arXiv:2602.19069v2 [cs.AI] 15 May 2026

Asking the Right Questions: Improving Reasoning with Generated Stepping Stones

Hengyuan Hu[1,2,*,†], **Tingchen Fu**[1,3,*], **Minqi Jiang**[1,†], **Alexander H Miller**[1], **Yoram Bachrach**[1], **Jakob Nicolaus Foerster**[1,3]

[1]FAIR at Meta, [2]Stanford University, [3]University of Oxford
[*]Joint first author, [†]Work done while at Meta

Recent years have witnessed tremendous progress in enabling LLMs to solve complex reasoning tasks such as math and coding. As we start to apply LLMs to harder tasks that they may not be able to solve in one shot, it is worth paying attention to their ability to construct intermediate stepping stones that prepare them to better solve the tasks. Examples of stepping stones include simplifications, alternative framings, or subproblems. We study properties and benefits of stepping stones in the context of modern reasoning LLMs via ARQ (**A**sking the **R**ight **Q**uestions), a simple framework that introduces a question generator to the default reasoning pipeline. We first show that good stepping stone questions exist and are transferrable, meaning that good questions can be generated, and they substantially help LLMs of various capabilities in solving the target tasks. We next frame stepping stone generation as a post-training task and show that we can fine-tune LLMs to generate more useful stepping stones by SFT and RL on synthetic data.

Date: May 18, 2026
Correspondence: Hengyuan Hu at hengyuan@stanford.edu

Meta

1 Introduction

There has been rapid progress in training large language models (LLMs) to solve increasingly complex reasoning tasks such as mathematics (OpenAI et al., 2024; Lightman et al., 2023; Shao et al., 2024), coding (Chen et al., 2021; Gehring et al., 2025), and decision-making in long-horizon agentic environments (Chan et al., 2025; Jimenez et al., 2024).

A common recipe for building reasoning models combines post-training with test-time scaffolding. The first (optional) step in post-training is to curate datasets that explicitly exhibit the desired reasoning behaviors and use them for supervised finetuning (SFT). For instance, problem-solving datasets annotated with high-quality chain-of-thought (CoT) rationales (Wei et al., 2022) can encourage pretrained LLMs to reliably express these behaviors after SFT (Ye et al., 2025; Muennighoff et al., 2025). Then reinforcement learning (RL) can be used to reinforce effective strategies that achieve high reward (DeepSeek-AI et al., 2025). After post-training, models can further be augmented with test-time scaffolds that provide explicit procedures for verification (Zheng et al.,

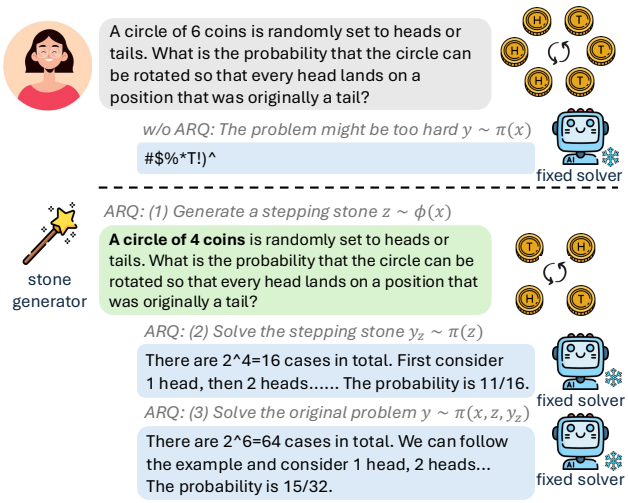


Figure 1 Illustration of ARQ. Instead of directly solving a task that may be too hard for a given solver (top), ARQ (bottom) adds a question-asking step that prompts or trains LLMs to generate *stepping stone questions* which, once solved, provide guidance or inspiration for the original problem. The LLMs used to generate stepping stone questions are referred to as *stepping stone generators*. In this example, the stepping stone (generated by an LLM) focuses on a special case of the original problem.

2023; Huang and Yang, 2025), feedback (Lee et al., 2025), or iterative refinement (Madaan et al., 2023; Shinn et al., 2023). These scaffolds further improve performance when the underlying model is capable of leveraging such behaviors (Kim et al., 2025; Shen et al., 2025; Qin et al., 2025).

However, most post-training and inference-time approaches have focused on reasoning behaviors that help models solve the given problem or verify proposed solutions. In contrast, the ability to ask questions — especially to pose relevant, insight-generating questions that expand one's knowledge and thus enable progress on problems that are otherwise too hard — has received less attention. We believe this is due to the paradigm shift to encapsulate all reasoning behaviors into a single, long thinking trace. Yet, asking the right question is a crucial component of human intelligence (Newell et al., 1972). For example, when tackling challenging tasks such as scientific research, we often construct related but more approachable *stepping stone problems* to build intuition (Ho et al., 2022). Crucially, stepping stone problems are often not only easier to solve, but also cheaper and easier to verify, enabling faster iteration. For example, in deep learning research, practitioners routinely prototype on small datasets or simplified toy settings to validate the feasibility and correctness of ideas, assumptions, or implementations before scaling up to the target, large-scale domain (Kaplan et al., 2020; Henighan et al., 2020).

As LLMs are applied to longer-term reasoning scenarios (METR, 2025), it is important to understand whether they, like humans, can also generate a useful test-time curriculum by asking the right stepping stone questions. In this work, we propose a straightforward framework, ARQ (**A**sking the **R**ight **Q**uestions), to study this question from both inference and post-training perspectives. As shown in Figure 1, ARQ consists of two LLM-based modules with distinct roles: a *question generator* ϕ and a *problem solver* π. Given a target problem x, instead of directly producing a solution $y \sim \pi(x)$, ARQ first generates a stepping stone problem $z \sim \phi(x)$. It then samples a solution $y_z \sim \pi(z)$ to the stepping stone and finally solves the target problem with (z, y_z) in the context, i.e., $y \sim \pi(x; z, y_z)$. For simplicity, we do not use more advanced techniques for the LLM-based solver π, such as majority voting, since they are complementarity to our method. We instead focus only on the question generator ϕ to study whether asking the right questions *in isolation* can improve a given solver's success rate on challenging tasks.

We first investigate ARQ as an inference-time scaffold on top of existing reasoning LLMs on three challenging math benchmarks, showing that good stepping stones exist and that the *best stones* greatly improve the solver's success rate by 13% on average (Section 4.1). We then demonstrate that these improvements are *transferable*: good stepping stones are broadly helpful across multiple solvers with various levels of performance, and the gains are not due to overfitting to a specific solver (Section 4.2). With this evidence, we establish stepping stone generation as a meaningful post-training task. Here, we score a given stepping stone using the expected reward of the solver when solving the target problem conditioned on this stone. We rate generations from existing LLMs with this score function, and curate a dataset for post-training. We show that additional post-training on this dataset substantially improves the ability of LLMs to ask effective stepping stone questions (Section 4.3). Finally, we extend ARQ to generate multiple stepping stones sequentially or recursively using both off-the-shelf and post-trained models (Section 4.4), where we find that additional stepping stones can further improve performance. We thus believe that ARQ offers an encouraging step into an underexplored research direction: what questions are worth asking, and what principled methods can train LLMs to generate those questions that are most beneficial for a given downstream task?

2 Related Work

Post-training LLMs for Reasoning. The standard paradigm for enhancing the reasoning capabilities of LLMs during post-training typically consists of two stages: (1) Supervised Fine-Tuning (SFT) on curated, high-quality reasoning data to elicit CoT, and (2) Reinforcement Learning (RL) to further refine reasoning behaviors via human or verifiable feedback (Li et al., 2025c; Kumar et al., 2025). In the SFT stage, a primary challenge is the acquisition of high-quality rationales. Current literature addresses this either through self-bootstrapping techniques (Zelikman et al., 2022; Zhang et al., 2024) or by curating compact, high-signal datasets that prioritize diversity and quality (Muennighoff et al., 2025; Ye et al., 2025; Li et al., 2025a; Zhao et al., 2025). Subsequently, the RL stage aligns these models with human preferences or objective reward functions. This is achieved through on-policy methods using learned reward models (RLHF) (Ouyang et al., 2022), or via Direct Preference Optimization (DPO) (Rafailov et al., 2023), which bypasses the complexities of on-policy RL while

enhancing reasoning effectiveness. Apart from human feedback, verifiable reward RL (Lambert et al., 2025) utilizes automated, deterministic objectives such as exact-match correctness in mathematics (DeepSeek-AI et al., 2025; Team et al., 2025) or unit-test pass rates in programming (Xue et al., 2025)—to provide reliable and scalable reward signals for large-scale optimization.

While prior research has largely focused on developing end-to-end solvers, we investigate a less-explored dimension of reasoning model: the ability to formulate "stepping stone" questions. By generating these intermediate queries, our approach incrementally builds the knowledge and skills necessary to tackle problems that exceed the capabilities of current LLMs.

Inference-time Scaffolds for Reasoning. Recent research has explored integrating LLMs into scaffolds (lightweight procedures or programs that wrap an LLM as a component.) (Welleck et al., 2024; Wang et al., 2025), aiming to improve LLM performance by adding a layer of search or planning algorithms at inference time without modifying the underlying weights. Notable examples include parallel sampling techniques such as majority voting and Best-of-N aggregation (Wang et al., 2023b; Brown et al., 2024), iterative refinement loops (Madaan et al., 2023; Snell et al., 2024), and tree-based search strategies (Yao et al., 2023; Besta et al., 2024). ARQ introduces a novel question-asking step that can be integrated as an additional operator within iterative or tree-based search. Existing techniques, like majority voting, can also be leveraged to further enhance the solver of ARQ.

The inference part of our approach is closely related to prompting techniques (Press et al., 2023; Yasunaga et al., 2024; Zhou et al., 2023) that utilize in-context learning to induce models to generate auxiliary questions before addressing the primary task. For instance, Self-Ask and similar works (Press et al., 2023; Liu et al., 2022; Zheng et al., 2024; Zhou et al., 2024) prompt LLMs to retrieve intermediate facts or knowledge (e.g., a person's age) before resolving relevant queries. Similarly, Analogical Reasoners (Yasunaga et al., 2024) instructs models to generate and solve a related sub-problem as a precursor to the target math problem. This can be viewed as a simpler version of the ARQ inference procedure, condensed into a single step rather than a multi-stage process. Furthermore, Least-to-Most (Zhou et al., 2023) and Decomposed Prompting (Khot et al., 2023) employ a similar philosophy by decomposing tasks into a linear sequence of sub-problems.

These existing methods are purely inference-time approaches that aim to improve performance by eliciting extended CoT traces through prompting alone. However, later in Section 3.2, we observe that these prompt-based approaches are not beneficial anymore when applied to the latest reasoning LLMs that already produce long CoTs. ARQ can be seen as a modernization of these approaches with multi-step reasoning instead of purely prompting. Our analysis also shows the limitation of existing LLMs in terms of their question asking capability.

Synthetic Task & Data Generation for LLMs. Synthetic data and task has emerged as a scalable and cost-effective paradigm for teaching LLMs specialized skills without relying on scarce expert data. A common method involves using a frontier model to synthesize both novel tasks and their solutions on the basis of existing seed tasks (Xu et al., 2024; Luo et al., 2023; Tian et al., 2024; Li et al., 2025b), followed by a rigorous curation process to select high-quality training pairs for fine-tuning (Wang et al., 2023c; Chen et al., 2025). Within the reasoning domain specifically, recent work has expanded mathematical corpora through multi-perspective rewriting (Yu et al., 2024) and back-translation of questions from augmented reasoning traces (Lu et al., 2024).

In this work, we formalize stepping stone generation as a novel reasoning task: the objective is to generate the intermediate query that maximizes the probability of the solver reaching the solution for a given target question. The reward can be derived from Monte Carlo rollouts of the solver conditioned on the stepping stone question. Our fine-tuning results indicate that such a task enables models to generate more strategic stepping stone questions.

3 Method

When humans tackle long-term reasoning tasks such as proving novel theorems or conducting scientific research, we commonly approach the challenging problem by asking less complicated questions to decompose the task, to reduce the cost of experimentation, or to build our intuitions. These intermediate problems serve

Solver only	Analogical	Least-to-Most	Plan-and-Solve	ARQ	MajVote@3	Self-Refine
69.8%	58.6%	64.2%	60.7%	72.8%	70.0%	73.2%

Table 1 Performance of running various test-time scaffold on BeyondAIME (100 questions) with GPT-OSS-120B (high reasoning effort). We run method 20 times on each problem. **Solver only** directly uses the LLM to solve the target question. The methods in the middle, including ARQ, share the similar idea of generating intermediate steps, either as analogical questions (**Analogical**), decomposition (**Least-to-Most**), or plan (**Plan-and-Solve**). On the right, we have two methods that directly improve the *solver*. We run **MajVote** with 3 parallel rollouts to match the number of rollouts of ARQ and also run **Self-Refine**, a well-established paradigm where the LLM can iteratively refine its previous solution. ARQ outperforms existing techniques focusing on introducing intermediate steps but slightly underperforms the mature Self-Refine method on existing LLMs.

as *stepping stones* toward solving the ultimate problem. This ability to generate diverse yet useful stepping stones is also important for algorithms such as search and planning (Yao et al., 2023). In this work, we propose ARQ (**A**sking the **R**ight **Q**uestions), a framework that includes both an inference procedure (Section 3.1) and a synthetic data curation pipeline (Section 3.3) for post-training LLMs to generate more useful stepping stones.

3.1 The Inference Procedure of ARQ

We formalize LLM-based problem solving with the following components. Let x denote a problem stated in natural language, and π be an LLM-based stochastic solver from which we sample solutions, $y \sim \pi(x)$. The final solution quality is evaluated by an external reward function $R(y)$.

Figure 1 illustrates the core idea of ARQ. We introduce a new LLM-based stochastic generator ϕ that produces stepping stone problems ("stones") conditioned on the target problem x, denoted as $z \sim \phi(x)$. In the generator prompt, we define stepping stones as problems that are relevant to x but easier to solve, such as special cases of x, or closely related problems that share the same underlying solution strategy and can be used to check whether a proposed approach is likely to work. Then ARQ uses the solver π to sample a solution to this stepping stones $y_z \sim \pi(z)$. Finally, we prepend the generated stepping stone and its solution to the target problem in the prompt, and samples a solution from the solver, $y \sim \pi(x; z, y_z)$. The prompts for generator and solver are provided in Figures 11 to 13 of Section F.

Note that both the generator ϕ and the solver π are both reasoning LLMs that generate CoT before producing final outputs. We remove the thinking tokens before passing the outputs to the next module to simplify the context, as we find it leads to better performance. Since we do not have a ground-truth reward function for the generated problems, we treat the sampled solutions as if they were always correct. It is possible to apply additional steps, such as iterative refinement (Madaan et al., 2023) or LLM-as-a-judge (Zheng et al., 2023), to improve the correctness of the sampled solutions. However, for simplicity, we focus on the question-generation component and keep the solver procedure simple.

We can further apply this procedure multiple times to generate a collection of stepping stones and build a test-time curriculum. In this paper, we will consider two strategies: generating stepping stones sequentially, or generating them recursively. The number of stepping stones is a predetermined hyperparameter. In the sequential version, every time we generate a new stepping stone, the LLM takes as input both the existing ones and the target problem, $z_i \sim \phi(x; z_1, \ldots, z_{i-1})$, and comes up with a new problem that aims to bridge any gap between them. The stepping stones are solved by the same solver, $y_{z_i} \sim \pi(z_i)$, and the full sequence of stones and their solutions forms the context for solving the target problem, $y \sim \pi(x; z_1, y_{z_1}, \ldots, z_i, y_{z_i})$. In the recursive version, we treat the previously generated stepping stone as the new target problem, $z_i \sim \phi(z_{i-1})$ and $z_0 = x$. The stepping stones are solved in *reverse* order, with the immediately subsequent stone and its solution in the context, $y_{z_{i-1}} \sim \pi(z_{i-1}; z_i, y_{z_i})$. The motivation behind the recursive version is to keep constructing stepping stones that help solve previously generated stepping stones, with the target problem at the root.

3.2 Limitation of Running ARQ on Existing LLMs

The ARQ procedure described above would ideally be an inference only approach applicable to any existing LLMs. However, due to the novelty nature of this question generation step, we find that existing LLMs are not

guaranteed to benefit from such an inference time procedure off-the-shelf as they generate a mixture of good and bad stepping stones. To illustrate this point, we run a pilot experiment in Table 1 using GPT-OSS-120B (high reasoning effort), an open-weight LLM that is more likely to ask useful questions due to its strong reasoning performance. We use BeyondAIME (Seed et al., 2025) as the test dataset because it is difficult enough that the LLM alone (**Solver only**) only achieves a moderate 69.8% success rate. We compare ARQ against works that share a similar spirit of introducing intermediate components to facilitate reasoning, such as Analogical (Yasunaga et al., 2024), Least-to-Most (Zhou et al., 2023), and Plan-and-Solve (Wang et al., 2023a), as well as methods that directly operate on the solver, such as Majority voting (Wang et al., 2023b) and Self-Refine (Madaan et al., 2023). Surprisingly, the prior prompting methods proposed before the reasoning LLMs era hurt the performance when the LLM itself is strong and emit long CoT out of the box, while ARQ is able to gain improvements with additional reasoning steps. However, ARQ slightly underperforms Self-Refine, which iteratively apply the LLM to solve and refine its previous solution. Our hypothesis is that the repeated-solving paradigm in the Self-Refine style methods benefits more directly from the existing RLVR (DeepSeek-AI et al., 2025) pipeline than our relatively novel paradigm of asking useful questions. We also want to emphasize the methods on the right (MajVote and Self-Refine) are complementary to our approach as they can be applied to the solver within ARQ to get better performance. We demonstrate this compatibility in Section B. The result is encouraging, while also highlighting the limited improvement when applying ARQ on existing LLMs that may have not been trained to ask the right questions. Therefore, we describe a recipe for curating a dataset of good stepping stone questions and post-training in the next section.

3.3 Data Curation and Post-training for ARQ

As we demonstrate in the previous section as well as in Section 4.1, existing LLMs generate a mixture of advantageous and detrimental stepping stones that lead to only moderate improvement on average. Therefore, a natural next step is to explore whether we can fine-tune LLMs to generate better stepping stones to achieve better performance under the ARQ framework. One option would be to collect annotated (*Question*, *CoT*, *Stepping Stones*) tuples from expert humans. However, this approach is expensive and limited by the human ability to generate good stepping stone problems for LLMs. Instead, we study the effect of fine-tuning LLMs to ask the right questions via a synthetic data generation pipeline.

A key step in generating synthetic datasets is to accurately evaluate the quality of the generated data. We define the score of a stone, z, as the expected reward on the target problem, x, averaged over Monte Carlo rollouts of solutions to the generated stone, y_z and the target problem, y; that is,

$$S(z, x) = \mathbb{E}_{y_z \sim \pi(z), y \sim \pi(z, y_z, x)} R(x, y). \tag{1}$$

Compared to the inference section where we roll out the solver π twice to get solutions to the stepping stone and the target question, here we roll out the solver $2n$ times in total to estimate the score in Equation (1) where n is the sampling budget.

To generate synthetic data for fine-tuning, we start from a dataset of problems and use an existing LLM to generate a pool of candidate stepping-stone problems for each target problem. We evaluate each stone with k rollouts to estimate its score $S(z, x)$, using an off-the-shelf LLM as the solver π. We then fine-tune the stone generator ϕ using a standard two-stage approach. First, we collect the highest-scoring stones for each target problem in the original dataset and use them for SFT. Next, we construct a preference dataset of paired stepping stones using the top- and bottom-performing ones, and use it to further fine-tune the LLM with DPO (Rafailov et al., 2023). Using these synthetically generated datasets, we substantially improve the performance of base LLMs as stone generators within the ARQ framework.

4 Experiments

The experiment section is divided into the following parts. First recall that in Section 3.2, we demonstrated that running ARQ on existing LLMs bring limited improvement compared to methods like Self-Refine when averaged over 20 individual runs where each run has its own stepping stone. In Section 4.1, we examine the score of individual stones generated with different LLMs as the stone generator. We show that existing LLMs generate a mixture of good and bad stepping stones, and that the good stepping stones are more

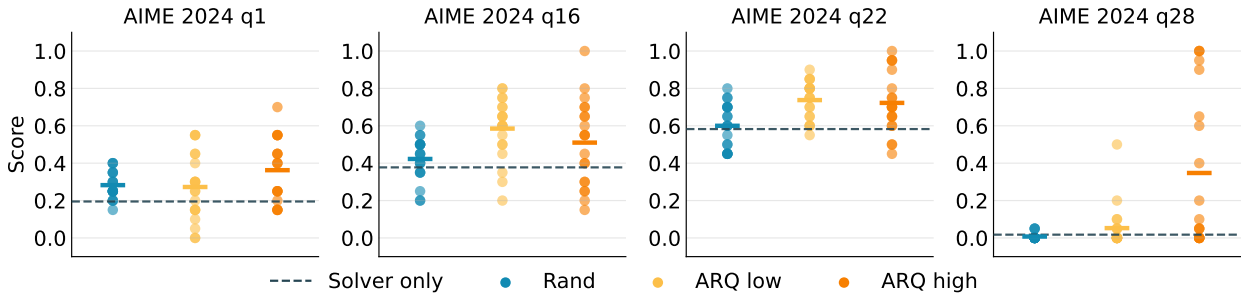


Figure 2 Scores of individual stones on selected questions from AIME 2024. Each point represents the score of the target problem conditioned on a generated stone, and the short horizontal bar indicates the average score over all stepping stones. The horizontal dashed line is the average performance of the *Solver only* baseline. ARQ generates stepping stones of varying quality. The best ones are highly beneficial, but some are detrimental and reduce the average improvement. ARQ with the more capable reasoning LLM is able to generate better stones, both in terms of the best and the average.

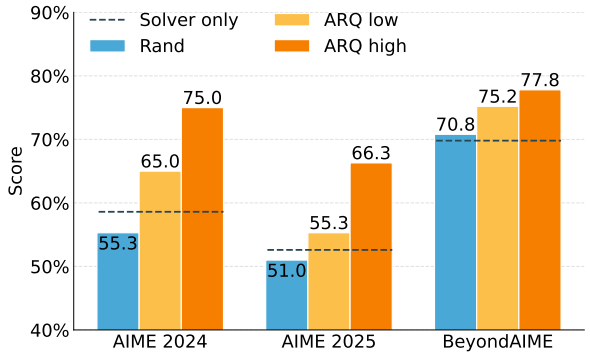


Figure 3 The performance of ARQ variants conditioned on the *best* stepping stone. Existing LLMs can generate highly useful stepping stones that increase the success rate on the target problems, and LLMs with better reasoning capabilities generate better stepping stones. In contrast, the best stone from *Rand* does not improve over the *Solver only* baseline (horizontal dashed lines). To avoid selection bias, we use half of the rollouts to select the best stones while we report the score using the other half.

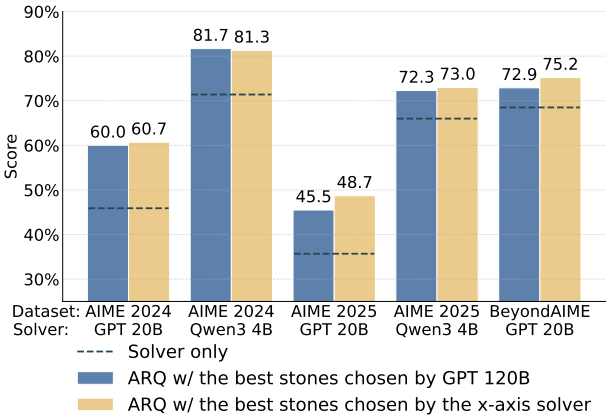


Figure 4 Transferability results of good stepping stones. Performance remains roughly the same regardless of whether the best stones are selected by the new solvers themselves or by a reference solver (GPT 120B) that has a different size and reasoning capacity. This indicates that good stepping stones are universally helpful across solvers. The *Solver only* scores of the new solvers are shown as dashed lines.

helpful towards solving the target problem. Then, in Section 4.2, we show that under our scoring system, the best stepping stones are not merely overfitting to a particular solver. Their benefit transfers to solvers of varying strength. The strong benefit of good stepping stones and their transferability lay the foundation for the post-training experiments in Section 4.3, where we fine-tune LLMs to generate better stepping stones. Finally, we extend ARQ to multiple stepping stones and evaluate two instantiations of this idea using both off-the-shelf LLMs and our fine-tuned LLMs.

4.1 Delving into Individual Stepping Stones

Setup: We use AIME 2024, AIME 2025, and BeyondAIME as our test benchmarks. The AIME 2024 and AIME 2025 datasets are widely used reasoning datasets. Each of them contains 30 challenging questions from the prestigious math competition for high school students. BeyondAIME (Seed et al., 2025) is a dataset of 100 questions that are much more challenging. We use MathVerify (Kydlíček) to compare generated answers against the ground truth. For the solver LLMs, we use GPT-OSS-120B with low reasoning effort in AIME 2024 and AIME 2025 and the same model with high reasoning effort in BeyondAIME due to their different difficulty levels. Using these two levels of reasoning effort also helps us understand the performance of different

methods when combined with LLMs of different capabilities.

For stepping stone generation, we consider a *Rand* baseline and two *ARQ variants*:

- **Rand**: a baseline that uses randomly generated stepping stones; the stone generator does not take the target problem as input and is asked to generate a random AIME-style question;
- **ARQ low**: ARQ using GPT-120B with reasoning effort *low* as stone generator ϕ, simulating a naive generator with superficial thinking before asking questions;
- **ARQ high**: ARQ using GPT-120B with reasoning effort *high* as stone generator ϕ, simulating the effect of having stepping stones from a more capable "oracle".

We first run the stone generator to generate 20 stepping stones independently. Then, to compute scores for each stone following Equation (1), we sample 20 rollouts per stone, i.e., 20 pairs of sampled solutions to the stepping stone and the target problem.

Results: We first look at the scores of individual stones Figure 2 on selected AIME 2024 problems. The full plots for all AIME 2024 and AIME 2025 problems are in Figures 7 and 8 of Section C. Figure 2 reveals that there is significant variance in the scores across generations. Many stones score much lower than the solver only baseline, even when using the more powerful LLM as the stone generator. Two possible factors contribute to the failure cases. One is that the generated stones are irrelevant or misleading, biasing the solver in the wrong direction. The other is that the generated questions are unsolvable for the solver and the wrong or invalid solutions in the context cause contamination when addressing the target problem. On the positive side, we notice that the best stepping stones are able to lead to significantly higher scores on the target problem, indicating that we may fine-tune LLMs to ask better questions with this score as a reward signal.

Next, we focus on the performance the best generated stepping stones averaged over the three datasets to get a holistic view. To report the scores of the best stepping stones, we use the average score of the first 10 solutions per stone to select the best stones and report their scores using the remaining 10 solutions to avoid selection bias. In Figure 3, we first notice that the best stepping stones substantially improve the performance in the ARQ framework. The best stones from *ARQ low* improve the performance by 5% over the *solver only* baseline, while the ones from *ARQ high* improve it by 13%. This improvement is not merely an effect of maximization over 20 runs or having more context, as it brings no improvement when maximizing over the 20 uninformative stepping stones in *Rand*. Second, between the two ARQ versions, we see generating useful stepping stones could be an emergent behavior for off-the-shelf LLMs, with the more capable reasoning models generating much better stepping stones. For a more intuitive understanding of the best stepping stones and the behavior difference between *ARQ high* and *ARQ low*, please refer to Section A for a qualitative analysis.

Obviously, the best score is not attainable at test time, as it requires access to the reward function in the math domain. However, this analysis is crucial to understand the strengths and weaknesses of ARQ on existing LLMs and paves the way for the post-training experiment in Section 4.3.

4.2 Transferability of Good Stepping Stones

After identifying the good stepping stones, the next question is whether the benefit from the good stones transfers and generalizes across different solvers.

Setup: The core idea for evaluating the transferability of good stepping stones is to take a set of *best stones* selected by one solver (*reference solver*) and evaluate them using a new solver (*target solver*). Then we compare the resulting score against ARQ using the best stones selected by the *target solver* itself. Specifically, we use the same datasets as in the previous section and use the stepping stones generated by *ARQ high* as the candidate pool. The reference solvers are the same ones used in the previous experiments, i.e., GPT-120B low for AIME 2024 and AIME 2025, and GPT-120B high for BeyondAIME. For the new target solvers, we use GPT-20B, a smaller model, with low and high reasoning efforts for AIME and BeyondAIME respectively. We also consider a Qwen3-4B (thinking) solver on AIME to evaluate transferability between solvers of different families. We use 20 rollouts per stone to score each stepping stone. When evaluating the stones selected by the target solvers themselves, we also perform the previously mentioned procedure to avoid bias, i.e., we use separate sets of rollouts for stone selection and evaluation.

Results: Figure 4 shows the results. First, it is worth noting that the new solvers have noticeably different strengths from the reference solver in the previous experiment. For example, on the AIME 2024 and AIME 2025 datasets, the new *GPT-20B low* solver is 21% worse than the reference solver on average, while the *GPT-20B medium* solver is 27% better. However, running ARQ with the best stepping stones improves the performance of all, reinforcing the observation that good stepping stones are helpful. Regarding the transferability of good stepping stones, the performance of ARQ using the stones selected by the reference solvers is on par with that using the stones selected by the target solvers shown on the x-axis. This shows that good stepping stones are generally useful across solvers of different reasoning and problem-solving capacity. They improve performance not because they overfit to some specific biases of a particular solver but rather because they are genuinely useful questions to ask and solve first.

4.3 Fine-tuning LLMs to Generate Stepping Stones

Next, we show that we can improve LLMs' ability to ask better questions by fine-tuning them on the ARQ task following the process described in Section 3.3.

Setup: We fine-tune Qwen3-8B (a thinking model) (Yang et al., 2025) and Qwen2.5-32B-Instruct (Qwen et al., 2025) to cover two representative starting points for additional post-training. Qwen3-8B is a capable reasoning model that has been extensively post-trained on various reasoning tasks. This model performs decently on math benchmarks, but we find that it is not not good at generating useful stepping stones out of the box. Qwen2.5-32B-Instruct is a base model that has not been post-trained for reasoning tasks. This model potentially has higher capacity, but it struggles to generate any stepping stones as it fails to follow our prompt and formatting instructions before fine-tuning.

To collect the dataset for fine-tuning, we run ARQ with GPT-120B high as the stone generator and GPT-120B low as the solver on all 918 pre-2024 AIME questions. We generate 20 stepping stones per question and use our scoring function to assign a reward to each stepping stone. We first run SFT on the best stepping stones, followed by DPO on paired stones constructed from the pool of stones. Additional details of the data curation and SFT + DPO hyper-parameters are in Section D. To verify the importance of the scoring function and differentiate our procedure from pure distillation, we also consider a **baseline** where we repeat the exact fine-tuning procedure on the 8B model but the stones have randomly assigned scores.

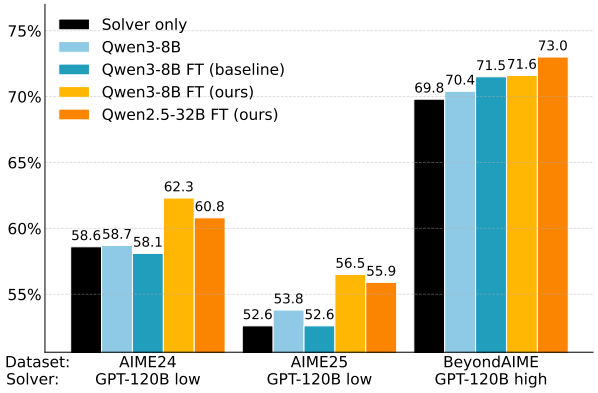


Figure 5 Performance of ARQ with Qwen3-8B and Qwen2.5-32B before and after fine-tuning (FT). Our fine-tuning procedure noticeably improves their performance as stone generators for different solvers.

We evaluate both LLMs before and after our fine-tuning procedure following the same practice as in Section 4.1. We run the ARQ inference pipeline 20 times per problem and report the *average* performance as an indicator of whether the models have improved their question-asking capability in general. As before, we use GPT-120B low as the solver in AIME 2024 and AIME 2025, and GPT-120B high as the solver in BeyondAIME.

Results: Figure 5 summarizes the results. We omit the score of Qwen2.5-32B as the stone generator before fine-tuning as it fails to follow our prompt. The Qwen3-8B reasoning model is able to generate well-formatted stepping stones, but they only improve performance by 0.6% on average compared to the solver-only baseline. After fine-tuning, both models greatly improve their question-asking capability, leading to improvements of 3.1% for Qwen3-8B and 2.9% for Qwen2.5-32B averaged over three datasets. The models now generate questions that are helpful both for a weak solver (as in AIME 2024 and AIME 2025) and for a solver that is significantly stronger than themselves in terms of model size and reasoning capability (as in BeyondAIME). The 8B model fine-tuned with the baseline approach, however, has mixed results across the three datasets. It performs noticeably worse than 8B (ours) on the two AIME datasets but on par on the hard BeyondAIME dataset, potentially because the strong solver used in BeyondAIME becomes the more dominant factor for

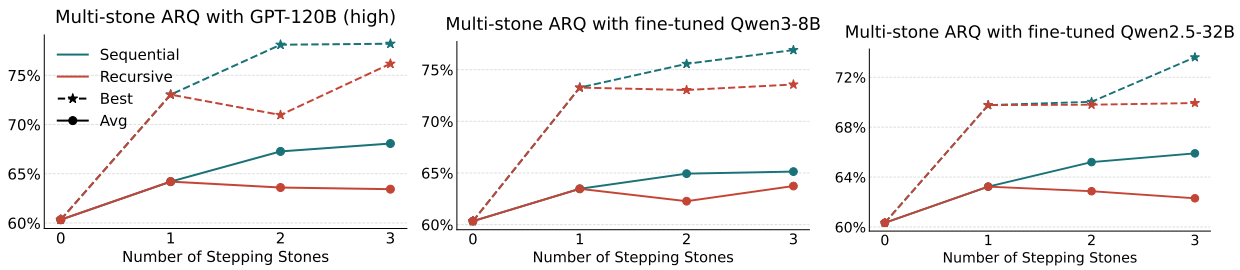


Figure 6 Performance of ARQ with multiple stones. We run ARQ with GPT-120B high (left), fine-tuned Qwen3-8B (middle), and fine-tuned Qwen2.5-32B (right) as stone generators. Results are averaged over the AIME 2024, AIME 2025, and BeyondAIME benchmarks. Each stone generator is asked to produce 20 sets of either 2 or 3 stepping stones, generated either *sequentially* or *recursively*. **Best** denotes performance conditioned on the best-performing set of stones, while **Avg** denotes performance averaged over the 20 sets. *Sequentially* generated stepping stones continuously improve both best-case and average-case performance. Results with 0 stepping stones correspond to the *solver only* in the previous sections.

performance. This result is particularly encouraging given the relatively small amount of data used for fine-tuning. It further validates the potential of stepping stone generation as a promising and interesting post-training task.

4.4 Extending ARQ to Multiple Stones

So far, we have focused on ARQ with a single stepping stone. Next, we evaluate whether increasing the number of stepping stones brings further improvement.

Setup: We consider two strategies for extending ARQ to multiple stepping stones discussed in Section 3.1: generating stepping stones *sequentially* or *recursively*. In the sequential case, the goal is to gradually build a curriculum, where each new question aims to bridge the gap between the existing stepping stones and the target problem. In the recursive case, each new stepping stone is generated to help solve the previously generated stepping stone, treating it as the new target problem. The prompts for sequential and recursive stone generation are listed in Section F.

We evaluate both strategies on the same three math benchmarks used in previous experiments, using the same solvers as in Section 4.1 for each benchmark. For the stepping stone generators, we consider an off-the-shelf model (GPT-120B with high reasoning effort) as well as the two fine-tuned LLMs from Section 4.3, Qwen3-8B FT (ours) and Qwen2.5-32B FT (ours). Note that sequential stone generation is a generalization test for our fine-tuned models, because they have not seen prompts in which they are given both the target problem and one or more existing stepping stones during our fine-tuning stage.

We run both multi-stone generation strategies to produce 20 sets of either 2 or 3 stepping stones for each benchmark problem. For each set, we use the solver to solve each stone and then solve the target problem, conditioning on the available stones and their solutions as context. We repeat this procedure 20 times to estimate the average score for each set of stones.

Results: Figure 6 summarizes the results for each model averaged over all three benchmarks. We also provide the separate results for each model on each dataset in Section E. The results show that sequential ARQ leads to substantial gains in both average and best performance across all three LLM backbones, while the recursive approach overall exhibits fluctuation or degradation. For the sequential version in particular, we observe a 5.2% absolute improvement in the best score and 3.9% in the average score when scaling from one stepping stone to three stepping stones using GPT-120B high as the stone generator. Notably, this performance gain extends to models that have only been fine-tuned on single stepping stone generation, demonstrating additional generalization benefits from the ARQ post-training procedure. The fine-tuned Qwen3-8B and the fine-tuned Qwen2.5-32B achieve absolute improvements of 1.7% and 2.7% in the average score, respectively, and their best scores improve by 3.7% and 3.8% respectively.

5 Conclusion

In this paper, we study LLMs' ability to address hard problems by asking the right stepping-stone questions. We propose the ARQ framework, which includes both an inference-time procedure to generate and utilize stepping-stone problems as well as a data-curation recipe and post-training procedure to fine-tune LLMs to generate better stepping stones. We show that good stepping stones improve the chance of solving the target problem, and we demonstrate that their helpfulness generalizes to solvers of varying capabilities. Finally, we curate a stepping-stone dataset and show that we can post-train multiple LLMs to become better stepping stone generators. The primary limitation of this work is the relatively small scale of the post-training experiment given constraints in computation and limited sources of hard target problems. The concept of stepping stones could be more useful in extremely challenging domains such as IMO level math and scientific research. Directions for future research include scaling up the study to larger datasets and other domains such as coding, as well as exploration of post-training LLMs to ask better questions via online RL.

References

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Michał Podstawski, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of Thoughts: Solving Elaborate Problems with Large Language Models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16): 17682–17690, Mar 2024. doi: 10.1609/aaai.v38i16.29720. https://ojs.aaai.org/index.php/AAAI/article/view/29720.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling, 2024. https://arxiv.org/abs/2407.21787.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Mądry. Mle-bench: Evaluating machine learning agents on machine learning engineering, 2025. https://arxiv.org/abs/2410.07095.

Lili Chen, Mihir Prabhudesai, Katerina Fragkiadaki, Hao Liu, and Deepak Pathak. Self-questioning language models. *arXiv preprint arXiv:2508.03682*, 2025.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. https://arxiv.org/abs/2501.12948.

Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Quentin Carbonneaux, Taco Cohen, and Gabriel Synnaeve.

Rlef: Grounding code llms in execution feedback with reinforcement learning, 2025. https://arxiv.org/abs/2410.02089.

Tom Henighan, Jared Kaplan, Mor Katz, Mark Chen, Christopher Hesse, Jacob Jackson, Heewoo Jun, Tom B Brown, Prafulla Dhariwal, Scott Gray, et al. Scaling laws for autoregressive generative modeling. *arXiv preprint arXiv:2010.14701*, 2020.

Mark K Ho, David Abel, Carlos G Correa, Michael L Littman, Jonathan D Cohen, and Thomas L Griffiths. People construct simplified mental representations to plan. *Nature*, 606(7912):129–136, 2022.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. https://openreview.net/forum?id=nZeVKeeFYf9.

Yichen Huang and Lin F. Yang. Winning gold at imo 2025 with a model-agnostic verification-and-refinement pipeline, 2025. https://arxiv.org/abs/2507.15855.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2024. https://arxiv.org/abs/2310.06770.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. In *The Eleventh International Conference on Learning Representations*, 2023. https://openreview.net/forum?id=_nGgzQjzaRy.

Joongwon Kim, Anirudh Goyal, Liang Tan, Hannaneh Hajishirzi, Srinivasan Iyer, and Tianlu Wang. Astro: Teaching language models to reason by reflecting and backtracking in-context. *arXiv preprint arXiv:2507.00417*, 2025.

Komal Kumar, Tajamul Ashraf, Omkar Thawakar, Rao Muhammad Anwer, Hisham Cholakkal, Mubarak Shah, Ming-Hsuan Yang, Phillip HS Torr, Fahad Shahbaz Khan, and Salman Khan. Llm post-training: A deep dive into reasoning large language models. *arXiv preprint arXiv:2502.21321*, 2025.

Hynek Kydlíček. Math-Verify: Math Verification Library. https://github.com/huggingface/math-verify.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James Validad Miranda, Alisa Liu, Nouha Dziri, Xinxi Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Christopher Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training. In *Second Conference on Language Modeling*, 2025. https://openreview.net/forum?id=i1uGbfHHpH.

Yoonho Lee, Joseph Boen, and Chelsea Finn. Feedback descent: Open-ended text optimization via pairwise comparison, 2025. https://arxiv.org/abs/2511.07919.

Dacheng Li, Shiyi Cao, Tyler Griggs, Shu Liu, Xiangxi Mo, Eric Tang, Sumanth Hegde, Kourosh Hakhamaneshi, Shishir G Patil, Matei Zaharia, et al. Llms can easily learn to reason from demonstrations structure, not content, is what matters! *arXiv preprint arXiv:2502.07374*, 2025a.

Jiazheng Li, Hongzhou Lin, Hong Lu, Kaiyue Wen, Zaiwen Yang, Jiaxuan Gao, Yi Wu, and Jingzhao Zhang. Questa: Expanding reasoning capacity in llms via question augmentation. *arXiv preprint arXiv:2507.13266*, 2025b.

Zhong-Zhi Li, Duzhen Zhang, Ming-Liang Zhang, Jiaxin Zhang, Zengyan Liu, Yuxuan Yao, Haotian Xu, Junhao Zheng, Pei-Jie Wang, Xiuyi Chen, et al. From system 1 to system 2: A survey of reasoning large language models. *arXiv preprint arXiv:2502.17419*, 2025c.

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*, 2023.

Jiacheng Liu, Alisa Liu, Ximing Lu, Sean Welleck, Peter West, Ronan Le Bras, Yejin Choi, and Hannaneh Hajishirzi. Generated knowledge prompting for commonsense reasoning. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3154–3169, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.225. https://aclanthology.org/2022.acl-long.225/.

Zimu Lu, Aojun Zhou, Houxing Ren, Ke Wang, Weikang Shi, Junting Pan, Mingjie Zhan, and Hongsheng Li. Mathgenie: Generating synthetic data with question back-translation for enhancing mathematical reasoning of llms, 2024. https://arxiv.org/abs/2402.16352.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct. *arXiv preprint arXiv:2306.08568*, 2023.

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. https://arxiv.org/abs/2303.17651.

METR. Measuring ai ability to complete long tasks. https://metr.org/blog/2025-03-19-measuring-ai-ability-to-complete-long-tasks/, 03 2025.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling, 2025. https://arxiv.org/abs/2501.19393.

Allen Newell, Herbert Alexander Simon, et al. *Human problem solving*, volume 104. Prentice-hall Englewood Cliffs, NJ, 1972.

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, Ally Bennett, Ananya Kumar, Andre Saraiva, Andrea Vallone, Andrew Duberstein, Andrew Kondrich, Andrey Mishchenko, Andy Applebaum, Angela Jiang, Ashvin Nair, Barret Zoph, Behrooz Ghorbani, Ben Rossen, Benjamin Sokolowsky, Boaz Barak, Bob McGrew, Borys Minaiev, Botao Hao, Bowen Baker, Brandon Houghton, Brandon McKinzie, Brydon Eastman, Camillo Lugaresi, Cary Bassin, Cary Hudson, Chak Ming Li, Charles de Bourcy, Chelsea Voss, Chen Shen, Chong Zhang, Chris Koch, Chris Orsinger, Christopher Hesse, Claudia Fischer, Clive Chan, Dan Roberts, Daniel Kappler, Daniel Levy, Daniel Selsam, David Dohan, David Farhi, David Mely, David Robinson, Dimitris Tsipras, Doug Li, Dragos Oprica, Eben Freeman, Eddie Zhang, Edmund Wong, Elizabeth Proehl, Enoch Cheung, Eric Mitchell, Eric Wallace, Erik Ritter, Evan Mays, Fan Wang, Felipe Petroski Such, Filippo Raso, Florencia Leoni, Foivos Tsimpourlas, Francis Song, Fred von Lohmann, Freddie Sulit, Geoff Salmon, Giambattista Parascandolo, Gildas Chabot, Grace Zhao, Greg Brockman, Guillaume Leclerc, Hadi Salman, Haiming Bao, Hao Sheng, Hart Andrin, Hessam Bagherinezhad, Hongyu Ren, Hunter Lightman, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian Osband, Ignasi Clavera Gilaberte, Ilge Akkaya, Ilya Kostrikov, Ilya Sutskever, Irina Kofman, Jakub Pachocki, James Lennon, Jason Wei, Jean Harb, Jerry Twore, Jiacheng Feng, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joaquin Quiñonero Candela, Joe Palermo, Joel Parish, Johannes Heidecke, John Hallman, John Rizzo, Jonathan Gordon, Jonathan Uesato, Jonathan Ward, Joost Huizinga, Julie Wang, Kai Chen, Kai Xiao, Karan Singhal, Karina Nguyen, Karl Cobbe, Katy Shi, Kayla Wood, Kendra Rimbach, Keren Gu-Lemberg, Kevin Liu, Kevin Lu, Kevin Stone, Kevin Yu, Lama Ahmad, Lauren Yang, Leo Liu, Leon Maksin, Leyton Ho, Liam Fedus, Lilian Weng, Linden Li, Lindsay McCallum, Lindsey Held, Lorenz Kuhn, Lukas Kondraciuk, Lukasz Kaiser, Luke Metz, Madelaine Boyd, Maja Trebacz, Manas Joglekar, Mark Chen, Marko Tintor, Mason Meyer, Matt Jones, Matt Kaufer, Max Schwarzer, Meghan Shah, Mehmet Yatbaz, Melody Y. Guan, Mengyuan Xu, Mengyuan Yan, Mia Glaese, Mianna Chen, Michael Lampe, Michael Malek, Michele Wang, Michelle Fradin, Mike McClay, Mikhail Pavlov, Miles Wang, Mingxuan Wang, Mira Murati, Mo Bavarian, Mostafa Rohaninejad, Nat McAleese, Neil Chowdhury, Neil Chowdhury, Nick Ryder, Nikolas Tezak, Noam Brown, Ofir Nachum, Oleg Boiko, Oleg Murk, Olivia Watkins, Patrick Chao, Paul Ashbourne, Pavel Izmailov, Peter Zhokhov, Rachel Dias, Rahul Arora, Randall Lin, Rapha Gontijo Lopes, Raz Gaon, Reah Miyara, Reimar Leike, Renny Hwang, Rhythm Garg, Robin Brown, Roshan James, Rui Shu, Ryan Cheu, Ryan Greene, Saachi Jain, Sam Altman, Sam Toizer, Sam Toyer, Samuel Miserendino, Sandhini Agarwal, Santiago Hernandez, Sasha Baker, Scott McKinney, Scottie Yan, Shengjia Zhao, Shengli Hu, Shibani Santurkar, Shraman Ray Chaudhuri, Shuyuan Zhang, Siyuan Fu, Spencer Papay, Steph Lin, Suchir Balaji, Suvansh Sanjeev, Szymon Sidor, Tal Broda, Aidan Clark, Tao Wang, Taylor Gordon, Ted Sanders, Tejal Patwardhan, Thibault Sottiaux, Thomas Degry, Thomas Dimson, Tianhao Zheng, Timur Garipov, Tom Stasi, Trapit Bansal, Trevor Creech, Troy Peterson, Tyna Eloundou, Valerie Qi, Vineet Kosaraju, Vinnie Monaco, Vitchyr Pong, Vlad Fomenko, Weiyi Zheng, Wenda Zhou, Wes McCabe, Wojciech Zaremba, Yann Dubois, Yinghai Lu, Yining Chen, Young Cha, Yu Bai, Yuchen He, Yuchen Zhang, Yunyun Wang, Zheng Shao, and Zhuohan Li. Openai o1 system card, 2024. https://arxiv.org/abs/2412.16720.

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 Miller, Maddie Simens,

Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022. https://arxiv.org/abs/2203.02155.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models, 2023. https://arxiv.org/abs/2210.03350.

Tian Qin, David Alvarez-Melis, Samy Jelassi, and Eran Malach. To backtrack or not to backtrack: When sequential search limits model reasoning. *arXiv preprint arXiv:2504.07052*, 2025.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. https://arxiv.org/abs/2412.15115.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. https://openreview.net/forum?id=HPuSIXJaa9.

John Schulman and Thinking Machines Lab. Lora without regret. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20250929. https://thinkingmachines.ai/blog/lora/.

ByteDance Seed, :, Jiaze Chen, Tiantian Fan, Xin Liu, Lingjun Liu, Zhiqi Lin, Mingxuan Wang, Chengyi Wang, Xiangpeng Wei, Wenyuan Xu, Yufeng Yuan, Yu Yue, Lin Yan, Qiying Yu, Xiaochen Zuo, Chi Zhang, Ruofei Zhu, Zhecheng An, Zhihao Bai, Yu Bao, Xingyan Bin, Jiangjie Chen, Feng Chen, Hongmin Chen, Riwei Chen, Liangqiang Chen, Zixin Chen, Jinsong Chen, Siyan Chen, Kaiyuan Chen, Zhi Chen, Jin Chen, Jiecao Chen, Jinxin Chi, Weinan Dai, Ning Dai, Jiahui Dai, Shihan Dou, Yantao Du, Zhengyin Du, Jianhui Duan, Chen Dun, Ting-Han Fan, Jiazhan Feng, Junda Feng, Ziyuan Feng, Yuwei Fu, Wenqi Fu, Hanjie Fu, Hao Ge, Hongyi Guo, Mingji Han, Li Han, Wenhao Hao, Xintong Hao, Qianyu He, Jerry He, Feng He, Wen Heng, Zehua Hong, Qi Hou, Liang Hu, Shengding Hu, Nan Hu, Kai Hua, Qi Huang, Ziyue Huang, Hongzhi Huang, Zihao Huang, Ting Huang, Wenhao Huang, Wei Jia, Bin Jia, Xiaoying Jia, Yuhua Jiang, Haobin Jiang, Ziheng Jiang, Kaihua Jiang, Chengquan Jiang, Jianpeng Jiao, Xiaoran Jin, Xing Jin, Xunhao Lai, Zheng Li, Xiang Li, Liyi Li, Hongkai Li, Zheng Li, Shengxian Wan, Ya Wang, Yunshui Li, Chenggang Li, Niuniu Li, Siyu Li, Xi Li, Xiao Li, Aoyan Li, Yuntao Li, Nianning Liang, Xinnian Liang, Haibin Lin, Weijian Lin, Ye Lin, Zhicheng Liu, Guanlin Liu, Guanlin Liu, Chenxiao Liu, Yan Liu, Gaohong Liu, Juncai Liu, Chundian Liu, Deyi Liu, Kaibo Liu, Siyao Liu, Qi Liu, Yongfei Liu, Kang Liu, Gan Liu, Boyi Liu, Rui Long, Weiqiang Lou, Chenwei Lou, Xiang Luo, Yao Luo, Caiping Lv, Heyang Lv, Bole Ma, Qianli Ma, Hongzhi Ma, Yiyuan Ma, Jin Ma, Wenchang Ma, Tingting Ma, Chen Mao, Qiyang Min, Zhe Nan, Guanghan Ning, Jinxiang Ou, Haojie Pan, Renming Pang, Yanghua Peng, Tao Peng, Lihua Qian, Lihua Qian, Mu Qiao, Meng Qu, Cheng Ren, Hongbin Ren, Yong Shan, Wei Shen, Ke Shen, Kai Shen, Guangming Sheng, Jinlong Shi, Wenlei Shi, Guang Shi, Shuai Shuai Cao, Yuxin Song, Zuquan Song, Jing Su, Yifan Sun, Tao Sun, Zewei Sun, Borui Wan, Zihan Wang, Xiaohui Wang, Xi Wang, Shuguang Wang, Jun Wang, Qinlong Wang, Chenyuan Wang, Shuai Wang, Zihan Wang, Changbao Wang, Jiaqiang Wang, Shihang Wang, Xuwu Wang, Zaiyuan Wang, Yuxuan Wang, Wenqi Wang, Taiqing Wang, Chengzhi Wei, Houmin Wei, Ziyun Wei, Shufa Wei, Zheng Wu, Yonghui Wu, Yangjun Wu, Bohong Wu, Shuang Wu, Jingqiao Wu, Ning Wu, Shuangzhi Wu, Jianmin Wu, Chenguang Xi, Fan Xia, Yuqiao Xian, Liang Xiang, Boren Xiang, Bowen Xiao, Zhen Xiao, Xia Xiao, Yongsheng Xiao, Chao Xin, Shulin Xin, Yuwen Xiong, Jingjing Xu, Ziwen Xu, Chenyin Xu, Jiayi Xu, Yifan Xu, Wei Xu, Yufei Xu, Shikun Xu, Shipeng Yan, Shen Yan, Qingping Yang, Xi Yang, Tianhao Yang, Yuehang Yang, Yuan Yang, Ximing Yang, Zeyu Yang, Guang Yang, Yifan Yang, Xuesong Yao, Bairen Yi, Fan Yin, Jianian Yin, Ziqiang Ying, Xiangyu Yu, Hongli Yu, Song Yu, Menghan Yu, Huan Yu, Siyu Yuan, Jun Yuan, Yutao Zeng, Tianyang Zhan, Zheng Zhang, Yun Zhang, Mofan Zhang, Wang Zhang, Ru Zhang, Zhi Zhang, Tianqi Zhang, Xinyi Zhang, Zhexi Zhang, Sijun Zhang, Wenqiang Zhang, Xiangxiang Zhang, Yongtao Zhang, Yuyu Zhang, Ge Zhang, He Zhang, Yue Zhang, Renjie Zheng, Ningxin Zheng, Zhuolin Zheng, Yaowei Zheng, Chen Zheng, Xiaoyun Zhi, Wanjun Zhong, Cheng Zhong, Zheng Zhong, Baoquan Zhong, Xun Zhou, Na Zhou, Huan Zhou, Hang Zhu, Defa Zhu, Wenjia Zhu, and Lei Zuo. Seed1.5-thinking: Advancing superb reasoning models with reinforcement learning, 2025. https://arxiv.org/abs/2504.13914.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. https://arxiv.org/abs/2402.03300.

Maohao Shen, Guangtao Zeng, Zhenting Qi, Zhang-Wei Hong, Zhenfang Chen, Wei Lu, Gregory Wornell, Subhro Das, David Cox, and Chuang Gan. Satori: Reinforcement learning with chain-of-action-thought enhances llm reasoning via autoregressive search. *arXiv preprint arXiv:2502.02508*, 2025.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. https://arxiv.org/abs/2303.11366.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024. https://arxiv.org/abs/2408.03314.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025.

Ye Tian, Baolin Peng, Linfeng Song, Lifeng Jin, Dian Yu, Lei Han, Haitao Mi, and Dong Yu. Toward self-improvement of llms via imagination, searching, and criticizing. *Advances in Neural Information Processing Systems*, 37:52723–52748, 2024.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2609–2634, Toronto, Canada, July 2023a. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.147. https://aclanthology.org/2023.acl-long.147/.

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

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions, 2023c. https://arxiv.org/abs/2212.10560.

Ziqi Wang, Boye Niu, Zipeng Gao, Zhi Zheng, Tong Xu, Linghui Meng, Zhongli Li, Jing Liu, Yilong Chen, Chen Zhu, et al. A survey on parallel reasoning. *arXiv preprint arXiv:2510.12164*, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *CoRR*, abs/2201.11903, 2022. https://arxiv.org/abs/2201.11903.

Sean Welleck, Amanda Bertsch, Matthew Finlayson, Hailey Schoelkopf, Alex Xie, Graham Neubig, Ilia Kulikov, and Zaid Harchaoui. From decoding to meta-generation: Inference-time algorithms for large language models. *arXiv preprint arXiv:2406.16838*, 2024.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. WizardLM: Empowering large pre-trained language models to follow complex instructions. In *The Twelfth International Conference on Learning Representations*, 2024. https://openreview.net/forum?id=CfXh93NDgH.

Zhenghai Xue, Longtao Zheng, Qian Liu, Yingru Li, Xiaosen Zheng, Zejun Ma, and Bo An. Simpletir: End-to-end reinforcement learning for multi-turn tool-integrated reasoning. *arXiv preprint arXiv:2509.02479*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. https://arxiv.org/abs/2505.09388.

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

Michihiro Yasunaga, Xinyun Chen, Yujia Li, Panupong Pasupat, Jure Leskovec, Percy Liang, Ed H. Chi, and Denny Zhou. Large language models as analogical reasoners. In *The Twelfth International Conference on Learning Representations*, 2024. https://openreview.net/forum?id=AgDICX1h50.

Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. LIMO: Less is more for reasoning. In *Second Conference on Language Modeling*, 2025. https://openreview.net/forum?id=T2TZ0RY4Zk.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models, 2024. https://arxiv.org/abs/2309.12284.

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

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. Rest-mcts*: Llm self-training via process reward guided tree search. *Advances in Neural Information Processing Systems*, 37:64735–64772, 2024.

Han Zhao, Haotian Wang, Yiping Peng, Sitong Zhao, Xiaoyu Tian, Shuaiting Chen, Yunjie Ji, and Xiangang Li. 1.4 million open-source distilled reasoning dataset to empower large language model training. *arXiv preprint arXiv:2503.19633*, 2025.

Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H. Chi, Quoc V Le, and Denny Zhou. Take a step back: Evoking reasoning via abstraction in large language models. In *The Twelfth International Conference on Learning Representations*, 2024. https://openreview.net/forum?id=3bq3jsvcQ1.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. https://openreview.net/forum?id=uccHPGDlao.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, and Ed H. Chi. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*, 2023. https://openreview.net/forum?id=WZH7099tgfM.

Pei Zhou, Jay Pujara, Xiang Ren, Xinyun Chen, Heng-Tze Cheng, Quoc V. Le, Ed H., Denny Zhou, Swaroop Mishra, and Huaixiu Steven Zheng. Self-discover: Large language models self-compose reasoning structures. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 126032–126058. Curran Associates, Inc., 2024. doi: 10.52202/079017-4004. https://proceedings.neurips.cc/paper_files/paper/2024/file/e41efb03e20ca3c231940a3c6917ef6f-Paper-Conference.pdf.

Appendix

A Qualitative Analysis for Stepping Stone

Model	Scale Reduction	Constraint Simplification	Generalization	Intermediate Step	Method Practice
ARQ high	68	13	8	8	3
ARQ low	62	22	8	5	3

Table 2 Distribution of the best stepping stones for 100 target problems in BeyondAIME among five categories.

In Section 4, we perform a series of experiments to understand the properties of stepping stones and their effect on solving target problems. To have a more intuitive understanding of how the stepping stone assists the solver, we manually review the stepping stone problems that produce the best roll-out performance for all 100 problems in BeyondAIME benchmark. We find that the stepping stones can be classified into five categories based on their strategies.

- **Scale Reduction:** The stepping stone keeps the original question structure unchanged and only reduces the large parameters to small values that can be calculated manually;
- **Constraint Simplification:** The stepping stone freezes a free variable, reduces the dimension of the problem, or only considers a special case.
- **Generalization:** The stepping stone changes a large value to a generic parameter such as "m" or "n" and asks for a generic formula.
- **Intermediate Step:** The stepping stone figures out the prerequisite subtasks or critical lemmas that must be completed first.
- **Method Practice:** The stepping stone changes the question structure to practice the core math tricks involved in the question.

The distribution of the best-performing stepping stones for *ARQ high* and *ARQ low* among the five distributions is shown in Table 2. We could observe from the table that *ARQ high* and *ARQ low* exhibit a highly similar trend. Under both settings, most of the best-performing stepping stones fall into the category of scale reduction and constraint simplification, while only very few best-performing stepping stones work by teaching the solver the required math tricks.

B Compatibility with Inference Approach that Improves the Solver

In Section 3.2, we compare our proposed ARQ with other similar methods that generate intermediate steps and two representative test-time scaling methods that directly operate on the solver, namely majority voting and self-refine. Notably, ARQ improves the problem-solving ability by introducing stepping stones with question generation ϕ. Being agnostic to the problem solver π, ARQ is compatible with test-time scaling approaches that enhance the performance of solver π. Here we use majority voting as an example, and the experimental results are in Table 3. As shown in the table, both *ARQ high* and *ARQ low* exhibit substantial

	No Maj	Maj@20	Gain
Solver only	69.8	76.0	+6.2
ARQ high	72.8	79.0	+6.2
ARQ low	72.7	77.0	+4.3

Table 3 The performance of ARQ and *solver only* baseline when combined with major voting.

gains with Maj@20, suggesting that ARQ can be combined with other test-time scaling approaches to achieve better performance.

C Supplementary Figures for Section 4.1

In Figure 2 of Section 4.1, we plot the performance of ARQ conditioned on individual stepping stones, together with a *Rand* baseline that generates random stepping stones without conditioning on the target problems. In Figure 7 and Figure 8, we show the results for all problems in AIME 2024 and AIME 2025. This holistic view reaffirms the observation that running ARQ with existing LLMs generates a diverse set of stepping stones containing both beneficial ones and detrimental ones. The LLM with better reasoning capability tends to generate more beneficial stepping stones with larger improvements. Both benchmarks also contain a number of questions where the solver itself can reliably solve the problem, in which case running ARQ is technically not necessary.

D Details of Dataset and Implementation of Post-Training in Section 4.3

	Dataset Size	Mean Prompt Length	Mean CoT Length	Mean Stepping Stone Length
SFT	918	184.25	4034.45	66.24
DPO	1063	193.93	4911.18	74.54

Table 4 The statistics of the SFT dataset and the DPO dataset in our experiments.

In this section we elaborate on the data collection process and the implementation details for our our post-training experiments in Section 4.3.

D.1 Data Collection

We take the 918 pre-2024 AIME questions as our source of target problems for both datasets. We first generate 20 stepping stones for every target AIME problem using GPT-120B high as the stone generator. Then for each stepping stone we sample 20 rollouts following Equation (1) to evaluate its quality. The stepping stone with the best score is included in the SFT dataset.

Next, we take the 20 generated stepping stones per problem to construct the DPO dataset. We rank the stepping stones by their scores and only keep the top-3 stones z_1, z_2, z_3 as well as the bottom-3 stones z_{18}, z_{19}, z_{20}. With the 6 remaining stones, we construct three pairs of stones, namely $(z_1, z_{20}), (z_2, z_{19})$ and (z_3, z_{18}). We compute the score difference for all pairs of stones and keep the ones with a difference ≥ 0.25 in the DPO dataset. In principle, the rationale behind the data construction procedure is to 1) avoid having too many pairs from the same problem; 2) avoid having the same stone appear multiple times, which may cause overfitting; and 3) avoid using pairs where the score gap between the preferred and non-preferred is small. This procedure results in a dataset of 1063 preference pairs generated from 436 unique target problems. The average score difference over the selected pairs is 0.47.

The statistics of the SFT dataset and the DPO dataset, including the dataset size and the mean length of prompts, chains of thought, and generated stones, are displayed in Section D. Note that for the DPO dataset, the length is averaged over both positive examples and negative examples. Moreover, we plot the distribution of the CoT length in Figure 9. From the figure, we observe that there is a long tail in the CoT length distribution, especially for the DPO dataset, where the CoT can be longer than 40,000.

D.2 Implementation and Hyper-Parameters

We use Qwen3-8B (Yang et al., 2025) and Qwen2.5-32B-Instruct (Qwen et al., 2025) as the base models for post-training, but in principle the post-training approach is agnostic to specific language models. Our experiments are conducted on a cloud Linux server with Ubuntu 22.04.5 operating system. The codes are written in Python 3.12.8. We run our experiments on Nvidia H200 GPU. The hyperparameters for SFT

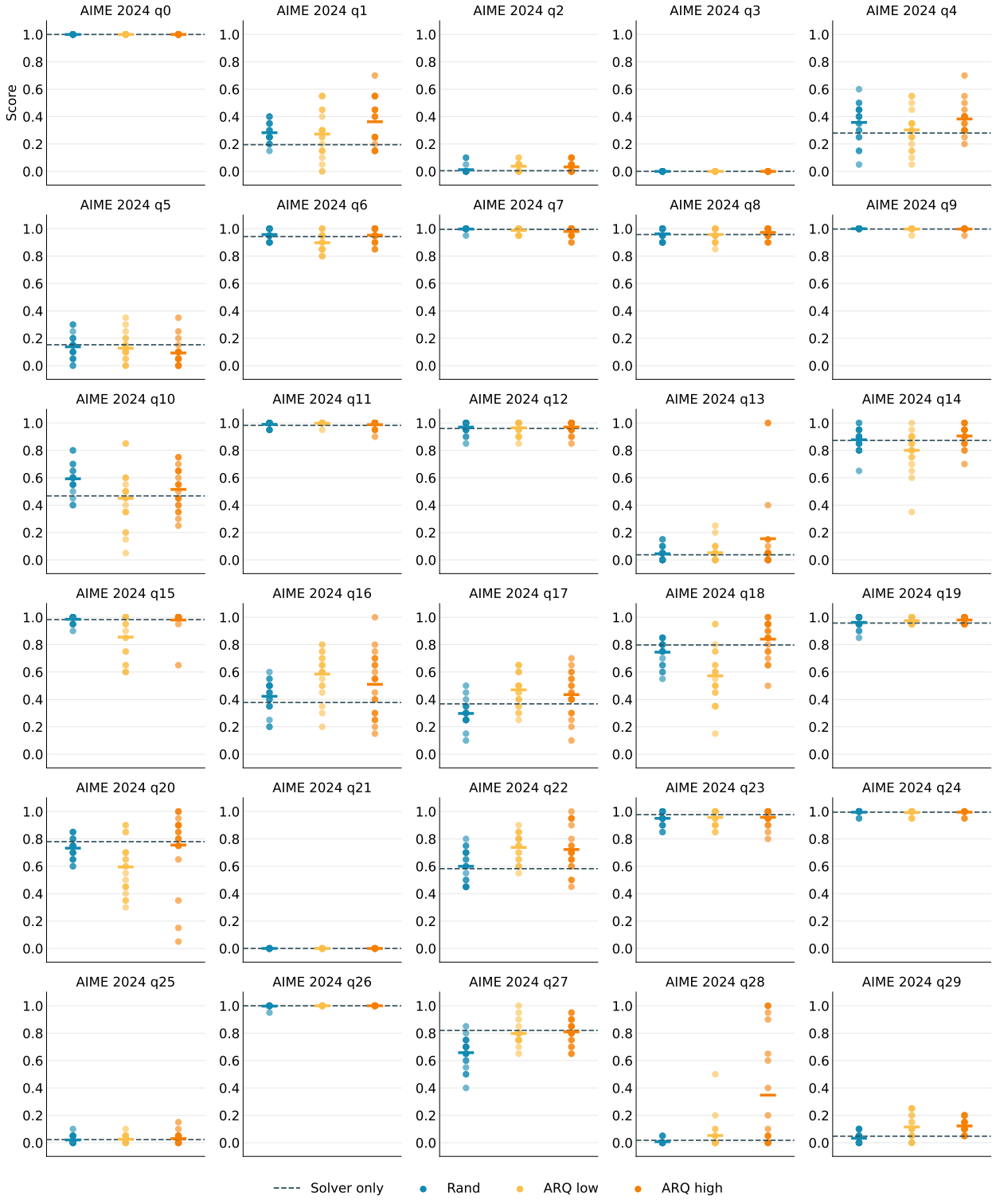


Figure 7 Score of individual stones on all questions from AIME 2024, a complete version of Figure 2. The solver is GPT-120B with low reasoning effort for all methods.

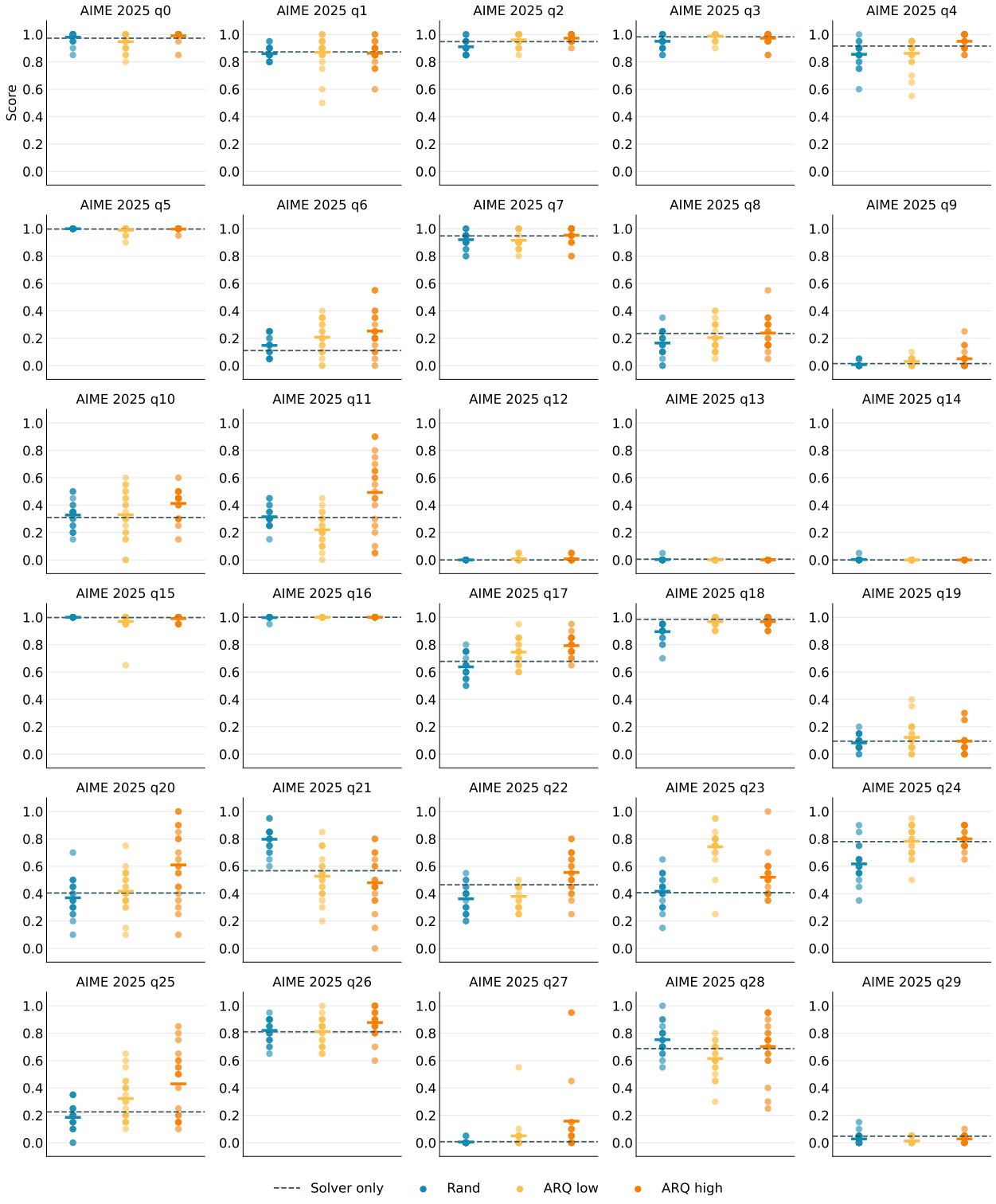


Figure 8 Score of individual stones on all questions from AIME 2025. The solver is GPT-120B with low reasoning effort for all methods.

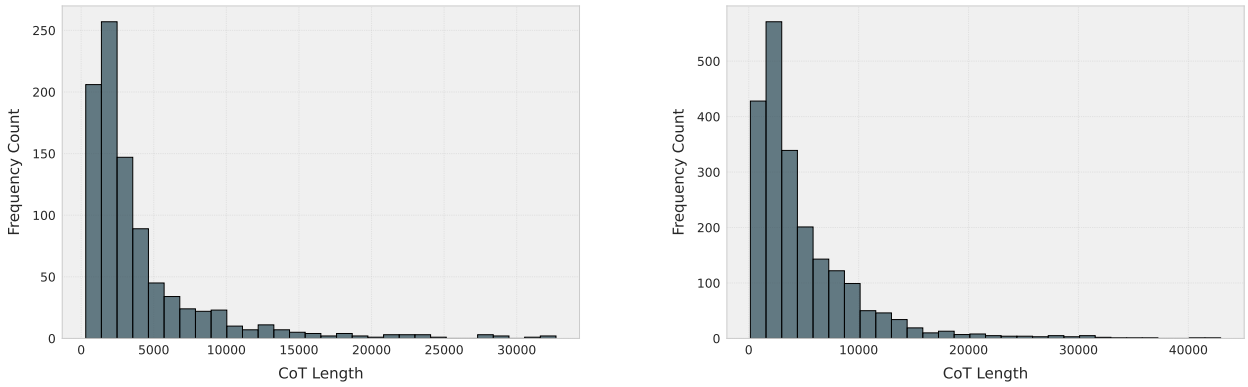


Figure 9 The distribution of the CoT length in our constructed SFT dataset (left) and DPO dataset (right).

Hyper-parameter	Value
max_length	16384
batch_size	16
lr	1e-5
adam_betas	(0.9, 0.95)
grad_clip	1.0
weight_decay	0.0001
warmup_ratio	0.05
schedule	cosine
epoch	5

Hyper-parameter	Value
max_length	14336
batch_size	16
lr	1e-6
adam_betas	(0.9, 0.95)
grad_clip	1.0
weight_decay	0.01
schedule	constant
epoch	1
dpo_beta	0.1

Table 5 The value of the hyper-parameters in our reasoning-oriented training experiment (Section 5.2) for SFT (left) and RL (right).

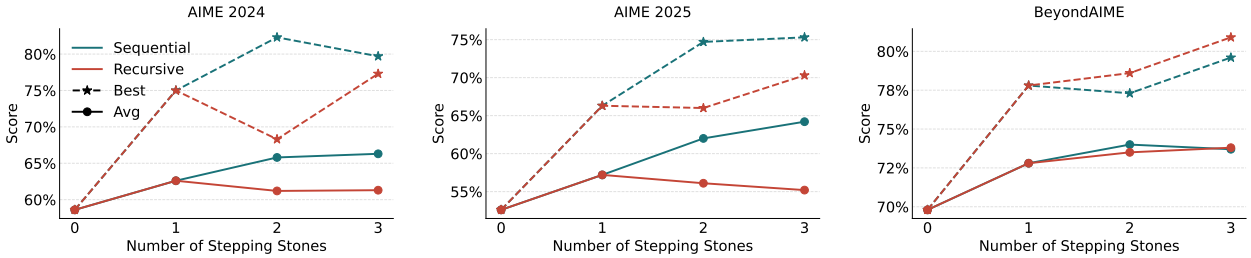
and DPO are shown in Section D.2. Notably, we set the maximum sequence length in both experiments to be 16,384, which is much longer than the average CoT length to include a small amount of extremely long sequences shown in Figure 9. Such a large sequence length makes it difficult to perform full-parameter fine-tuning on DPO, since DPO requires an additional reference model to compute KL-divergence, and it needs to process both positive and negative examples. Therefore, we use LoRA (Hu et al., 2022) for parameter-efficient fine-tuning with hyper-parameter $r = 256$ and $\alpha = 32$, following the setting of Schulman and Lab (2025).

E Supplementary Figures for Section 4.4

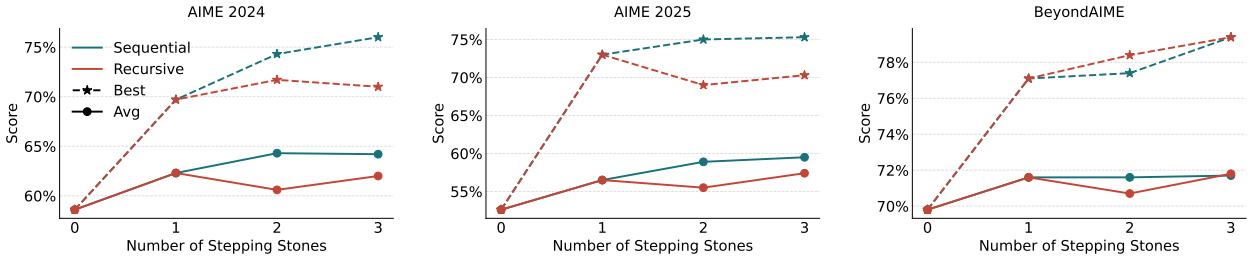
For the multiple stone experiments in Section 4.4, we propose two possible paradigms of extending the number of stepping stones, namely sequential stepping stones and recursive stepping stones. We compare these two paradigms using three stone generators and report the average performance over three datasets (AIME 2024, AIME 2025, and BeyondAIME) in Figure 6. To compare and analyze the performance on each individual dataset, we plot the performance of GPT-120B, Qwen3-8B post-trained, and Qwen3-32B post-trained in Figure 10a, Figure 10b and Figure 10c, respectively.

F The Complete List of Prompts

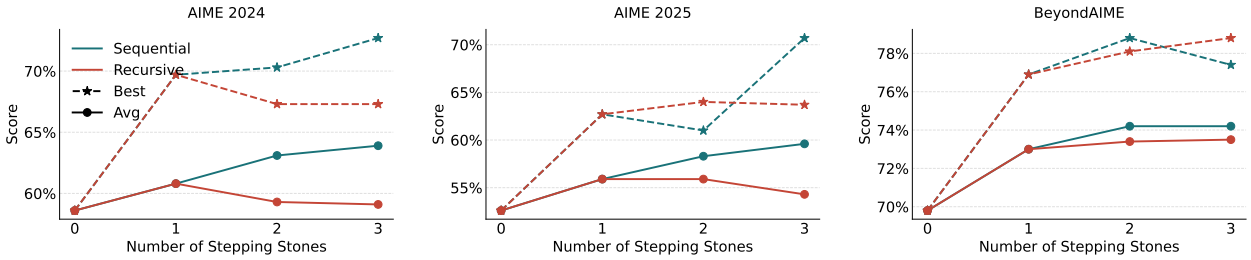
In this section we provide the prompts used for our experiments in Section 4.1 and Section 4.4. The prompt to generate a stepping stone in ARQ is shown in Figure 11. The prompts to solve the stepping stone and the target problem in ARQ are displayed in Figure 12 and Figure 13 respectively. The prompt to generate



(a) Multi-stone ARQ with GPT-120B (high) as stone generator.



(b) Multi-stone ARQ with Qwen3-8B post-trained as a stone generator.



(c) Multi-stone ARQ with Qwen2.5-32B post-trained as a stone generator.

Figure 10 Performance of sequential multiple stones and recursive multiple stones on AIME 2024 (left), AIME 2025 (middle), and BeyondAIME (right). **Best** is conditioned on the best generated stones for the target questions, and **Avg** is averaged over 20 stones.

a random question for the *Rand* baseline is shown in Figure 14. For the experiments involving multiple stones, we use the prompt in Figure 15 to generate stones sequentially and reuse the prompt in Figure 11 to generate stones recursively by treating the previously generated stone as the new target problem. The prompts in Figure 12 and Figure 13 are used to solve the stepping stones and final problems in the multi-stone experiments. For all prompts that require us to parse out the generated content, such as in stone generation, we instruct the LLMs to wrap the final output in YAML for easy parsing.

G The Use of Large Language Model

Large language model is used in our study as a general-purpose assist tools and we use it for checking grammar mistakes and fixing Latex compile errors.

H Dataset Credits and Licenses

We use the following datasets as test benchmarks. We acknowledge the original dataset creators and list the corresponding sources and licenses below:

AIME pre-2024: https://huggingface.co/datasets/gneubig/aime-1983-2024.

Task:
We are trying to write some subproblems that may be helpful or inspirational for solving the final problem. The subproblems should be stepping stones towards solving the final problem.

A subproblem can take one or a few of the following roles:
• a simpler version of the final problem;
• a special case of the final problem;
• a practice for some methods that will be useful for solving the final problem.

Subproblem:
<Placeholder>
Final problem:
{problem}

Instructions:
1. Be clear and **self-contained**, someone seeing only the subproblem should understand what's being asked.
2. You do not need to solve the proposed subproblem.
3. Write the subproblem in the **same tone and format** as the final problem.

Output format:
```yaml
problem: |
    ...
```

Figure 11 Prompt used to generate stepping stone problems in ARQ. The {problem} is replaced by the target problem.

Question:

{question}

Please reason step by step to solve the question above.
For the final solution, include critical details and put your final answer within \boxed{}.
If the final answer is a fraction, please reduce to the simplest form.
Don't use commas when writing out large numbers.

Figure 12 Prompt used to solve stepping stone problems in ARQ. The {question} is replaced by the stepping stone problem.

Study the following example problems and their solutions.

Example:

{stone problem}

Solution to Example:

{stone solution}

Task: Reason step by step to solve the following final problem, and put your final answer within \boxed{}.
If the final answer is a fraction, please reduce to the simplest form.
Don't use commas when writing out large numbers.

Final Problem:

{question}

Figure 13 Prompt used to solve target problems in ARQ. The {stone problem}, {stone solution}, and {question} are replaced by the stepping stone problem, the solution to the stepping stone problem, and the target problem.

Licensed under CC0: Public Domain.

AIME 2024: https://huggingface.co/datasets/gneubig/aime-1983-2024.
Licensed under CC0: Public Domain.

AIME 2025: https://huggingface.co/datasets/opencompass/AIME2025.
Licensed under the MIT License.

BeyondAIME: https://huggingface.co/datasets/ByteDance-Seed/BeyondAIME.
Licensed under the MIT License.

Task:
Please randomly generate an AMC or AIME style math problem to help student practice.

---

1. Be clear and **self-contained**.
2. You do not need to solve the proposed problem.

Output format:

```yaml
problem: |
    ...
```

Figure 14 Prompt used to generate a random stepping stone in the *Rand* baseline.

Task:
We are trying to write some subproblems that may be helpful or inspirational for solving the final problem. The subproblems should be stepping stones towards solving the final problem.

Each subproblem can take one of the following roles:
• a simpler version of the final problem;
• a special case of the final problem;
• a practice for some methods that will be useful for solving the final problem.

We have proposed a few subproblems. Please propose a new subproblem that takes a step closer towards solving the problem.

Subproblem {1}:
{subproblem 1}

Subproblem {2}:
{subproblem 2}

...

Subproblem {k}:
`<Placeholder>`

Final problem:
{problem}

Instructions:

1. Be clear and **self-contained**, someone seeing only the subproblem should understand what's being asked.
2. You do not need to solve the proposed subproblem.
3. Write the subproblem in the **same tone and format** as the final problem.

Output format:

‘‘‘ yaml
subproblem: |
...
‘‘‘

Figure 15 Prompt used to generate the k-th sequential problem in ARQ.