
ASK-E: An Environment for Calibrated Question Generation

Sarah Pratt^{1*} Jae Sung Park² Scott Geng¹ Ali Farhadi¹

¹University of Washington ²Allen Institute for AI

Abstract

Today, we improve models by training and evaluating them on problems at the frontier of their abilities. Creating such problems is itself a demanding task, requiring the ability to probe model limits and generalize beyond existing question distributions. It also means placing problems at a precise difficulty level, which requires understanding what it takes to solve them. In short, generating problems calibrated to a model’s current frontier demands capability beyond it, an increasingly burdensome constraint as models improve. Our key insight is that we can leverage this constraint to our advantage: a model that can generate problems consistently calibrated to a given frontier must possess capability beyond it. Accordingly, we present Ask-E, an environment that benchmarks and trains models on their ability to write questions at a given skill level, rather than answer them. Concretely, we define target skill levels as ranges bounded by the capabilities of two existing language models. A generated question is successfully calibrated if exactly one of the two models can solve it, placing it precisely within the target range and differentiating the capabilities of these models. Ask-E serves both as a benchmark and a training environment, where models generate problems calibrated to a variety of skill levels. We find that even frontier models achieve below 50% calibration on the benchmark, leaving significant headroom to measure future progress. We also show that training on this environment leads to improvements across a number of downstream math benchmarks even with no new math data, no interaction with stronger models, and no correctness-based reward.

“Computers are useless. They can only give you answers.”

— Pablo Picasso

1 Introduction

The dominant force behind language model improvement in domains like mathematics has been measuring their ability to solve existing problems. The resulting performance is then used either as a training signal or as a benchmark to measure progress. However, this approach depends on a supply of problems at the right difficulty level. Problems that are too easy for today’s frontier, such as grade school math, do nothing to differentiate between state-of-the-art models. Problems that are too hard, such as solving the Riemann hypothesis or correctly guessing a number between one and a million, are equally uninformative, as all models will fail regardless of their relative skill.

*Correspondence to spratt3@uw.edu

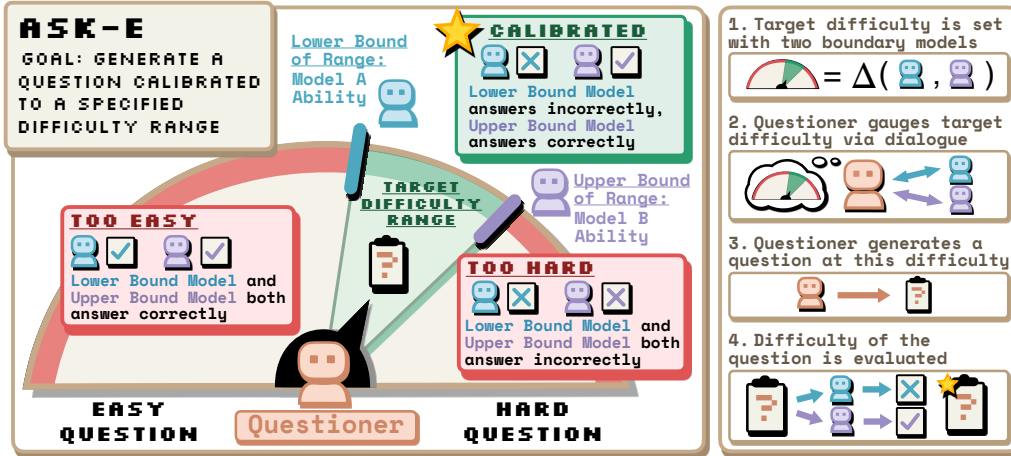


Figure 1: **Overview of the Ask-E environment.** The questioner model must generate a question calibrated to a target difficulty range defined by an upper and lower bound, where each bound corresponds to the ability level of a specific LM. A question is successfully calibrated if the lower bound model answers incorrectly while the upper bound model answers correctly. The questioner achieves this through a multi-turn dialogue with the boundary models (right) in which it probes their abilities before generating its final question.

Obtaining questions at the right difficulty level is not only crucial, it is very difficult. The best models currently perform on par with world-class high school students, meaning that researchers must find math competition champions, graduate students, and professors to continue to challenge them. Problem authors must be able to effectively probe models to understand their abilities, requiring knowledge of what is useful to ask and how to interpret the responses. They must have mathematical understanding beyond that of the models to identify their current shortcomings. They must possess creativity to write questions sufficiently different from existing distributions to identify true generalizable skills. Finally, problem authors must understand the skills and knowledge needed at each step from the problem statement to the solution in order to effectively target the capabilities they hope to measure or elicit.

In this work, we posit that the challenging nature of producing well-calibrated questions may itself provide a path forward. As previously noted, writing a question that discriminates between model abilities requires all the skills needed to answer it, and more. Additionally, a question author’s success is judged not by a more capable supervisor, but by the evaluated models themselves. If the evaluated models differ in their ability to solve the question, the author has succeeded. If all models answer correctly or all answer incorrectly, the author has failed. The question writer, performing the harder task, is indirectly graded by the solvers, performing the easier one.

Motivated by these observations, we present Ask-E, an environment where the goal is to do exactly what researchers have themselves been doing for decades: generate questions that will differentiate the capabilities of existing models. In Ask-E, a questioner-model interacts with various pairs of solver-models sampled across a broad range of capabilities. The questioner first probes the model in the pair to identify their capability difference, and then produces a final question that targets this span between the two models’ skill levels. The questioner model is successful in an interaction with a given pair if it produces a question that one model can answer correctly and one cannot.

Ask-E offers three advantages over the standard question-answering approach. First, it provides a more demanding evaluation paradigm. Instead of giving models increasingly difficult questions to answer, we ask them to do something harder than answering questions. Second, it naturally scales with the field. While improved model capabilities saturate most benchmarks, in Ask-E better models just represent more advanced skill levels to target meaning the task naturally gets harder as models get better. Finally, it offers a path forward

as models surpass human ability. Standard approaches require a supervisor more capable than the evaluated model, a constraint that becomes unsustainable as models improve. Ask-E inverts this dynamic: the supervisory signal can come from models of equal or lesser ability, whose answer variance reveals whether the evaluated model has successfully calibrated its question.

Our contributions are as follows:

- **The Ask-E environment.** We present an environment for measuring and developing a model’s ability to generate calibrated questions. Concretely, we define target difficulty ranges bounded by the capabilities of two existing LMs. A questioner model must interact with these boundary models to identify a gap in their abilities and then calibrate a question to that gap. The questioner succeeds if one model is able to solve the question and the other is not, differentiating their skill level.
- **Benchmarking on Ask-E.** We evaluate a range of models on calibrated question generation and find that performance correlates with general model ability. State-of-the-art models achieve only 45.1% success rate, demonstrating substantial room for improvement. Furthermore, the benchmark is designed to evolve: as models saturate the current boundary set, stronger boundary models can be introduced to raise the difficulty ceiling.
- **RL training on Ask-E.** We show that training on this environment produces transferable skills. After RL training with no new math data, no interaction with stronger models, and no correctness-based reward, the trained model shows improvements on challenging downstream math benchmarks including AIME, HMMT, and IMO.

2 Related Work

LMs for Question Generation. Prior work has explored training models to generate questions for a variety of practical applications. In the area of education, for example, a number of works [53, 49, 3, 14, 61] have explored the ability of LMs to generate educational questions to aid in student learning. Effective question generation has also been studied as a tool to resolve ambiguity in human preferences and instructions in human-LM conversations [38, 52, 31, 43]. These works point to the potential of LMs to not only provide information, but to learn to ask the useful questions. However, in this work, we take this one step further. We view question asking not only as a means to an end, but as a skill which can measure complex reasoning abilities within models as well as a training task to reinforce these abilities.

LMs for Automated Benchmarks. LMs have also been increasingly used to generate evaluation benchmarks for other LMs. As models grow more capable, hand-crafted benchmarks require greater expertise to construct [16, 37] and saturate more quickly [26]. One response to this problem has been to develop algorithmic methods that systematically generate harder questions through templates and composition [60, 39, 59, 50, 47]. These approaches can be more robust than static benchmarks, but remain limited in the scope of what they can evaluate. Other works use LMs themselves to generate challenging benchmarks. AutoBench [32] iteratively proposes and refines questions to satisfy properties like difficulty or novelty. MATH² [45] generates novel questions by combining pairs of skills extracted from existing benchmarks. CHASE [42] composes hard problems from simpler, verifiable components rather than generating them directly. In these frameworks, question generation is a means of constructing the benchmark, and a model is evaluated by whether its answers match the generating LM’s answers. We instead treat question generation as the capability being evaluated, assessing whether a model’s generated questions match a target LM’s ability level. This avoids two drawbacks of standard automated benchmarks. First, in standard settings, the supervising LM must be more capable than the evaluated model, as it must generate the question and answers while the evaluated LM just generates the answer. This means automated benchmarks can only evaluate models below the capability of the LM that generated them, limiting the usefulness for benchmarking frontier models. In Ask-E, this relationship is inverted. The evaluated model performs the harder task of question generation, while the supervising models perform the easier task of answering. This allows the benchmark to be fully automated without necessarily being solvable by an existing LM.

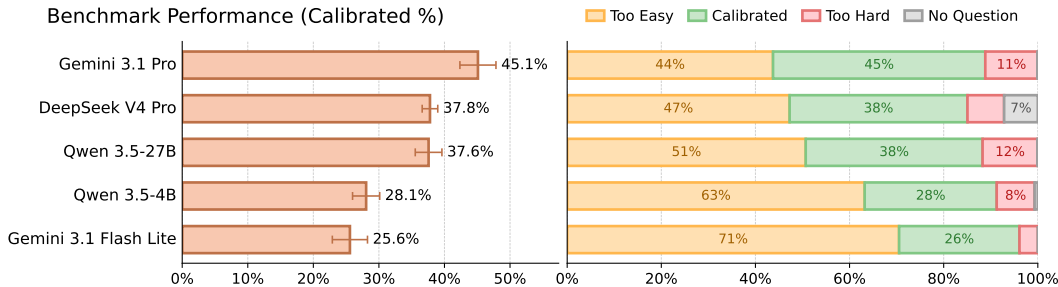


Figure 2: **Benchmark performance of five models on Ask-E.** (Left) Calibration rate across all sessions. Error bars correspond to 95% confidence intervals computed across 10 independent runs of all 190 boundary pairs. (Right) Breakdown of generated question outcomes into too easy, calibrated, too hard, and missing categories.

Second, standard automated benchmarks rely on LM-generated answers as ground truth, which can be noisy or incorrect [15]. In Ask-E, ground truth is not a generated answer but the boundary models’ own performance, which is an observable property rather than a generated artifact. As models continue to improve, automated evaluation will become increasingly necessary, and we see our approach as a natural step in this direction.

LMs for Answer Evaluation. Even with human-authored benchmarks, it is increasingly common for responses to be graded by LMs [10, 37], and this has been explored during training as well [57, 5]. In these settings, the grading LM is a tool for evaluation, not itself being evaluated. However, several recent works explicitly evaluate an LM’s ability to perform tasks beyond answering questions. IMO-GradingBench [37] evaluates whether LMs can accurately score mathematical proofs, and RewardBench [30] evaluates reward models on their ability to rank responses. These works highlight that tasks beyond answering questions also require genuine skill, and a model’s ability to perform them can be just as revealing of its underlying capabilities. The Delta Learning Hypothesis [17] goes further, showing that training a model to identify the better of two weak model responses improves the model’s own capabilities in the underlying domain, demonstrating that performing evaluation can itself be a source of learning. We view our work as an extension of this direction, where rather than ranking model capabilities, our task requires targeting them precisely enough to generate a question that falls within a specific ability gap.

LMs for Play via Dialogue. A recent related line of research explores learning through competitive interaction between models. DTE [48] trains a model on multi-agent debate traces. SPIN [9] uses a generator-discriminator framework where one copy attempts to produce text indistinguishable from human writing while another tries to tell them apart. CodeClash [55] has agents iteratively edit code based on how it performs against a competitor. SPIRAL [34] has agents compete in zero-sum games like poker and tic-tac-toe, finding that this improves general reasoning abilities.

Most closely related to our work is a series of recent papers on self-play for calibrated question generation [35, 56, 8, 27, 58, 24]. In these frameworks, a model acts as both a challenger, generating questions, and a solver, answering them. The challenger is rewarded for generating questions at the edge of the solver’s ability, and the solver is rewarded for answering correctly. These works differ from each other primarily in their chosen domain and in how they obtain ground truth for verification: grounding question-answer pairs in an external document corpus [56, 35], generating and executing code [58], using majority voting as a proxy for correctness [24, 8], or relying on an external reward model [27]. All demonstrate that models can set meaningful challenges for each other and improve by solving them, without relying on human-authored questions.

However, in all of these works, question generation is a means to an end, not a measured capability. The challenger adapts to a single solver as it improves, but there is no metric

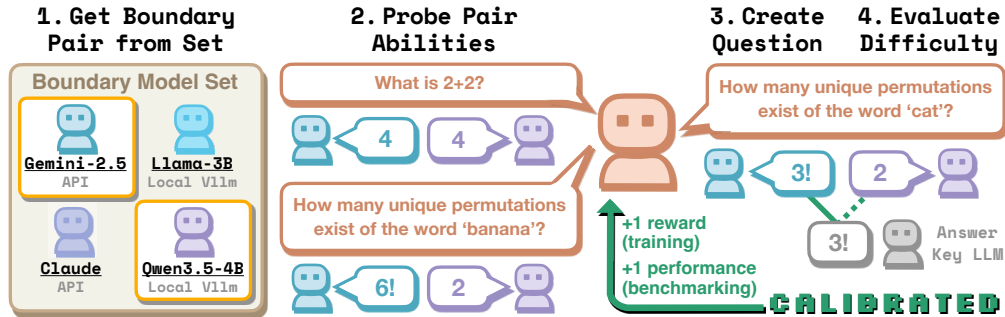


Figure 3: **A detailed walkthrough of the Ask-E environment for a single session.** (1) A boundary pair is selected from the boundary model set (2) The questioner probes the pair with questions and observes their responses to gauge the gap in their abilities. (3) The questioner generates a final question designed to fall within this gap. (4) Both boundary models and an independent answer key model answer the question; the session is scored as calibrated if exactly one boundary model matches the answer key.

of its question-asking ability. We frame calibrated question generation as the skill itself, evaluating it across a fixed set of boundary model pairs spanning a wide range of abilities. This allows Ask-E to serve not only as a training environment but also as a benchmark, tracking improvement at calibrated question generation within a training run and across model releases. As the works discussed above suggest, posing well-calibrated questions may require skills that include and exceed those needed to answer them, and scalable model-driven evaluation will only become more important as models improve. Our framework bridges these two directions, providing both a training environment and a benchmark for a capability that requires complex generalizable skills and scales without human supervision

3 The Ask-E Environment

Ask-E is an environment where models are presented with a target difficulty level, defined by the difference in ability levels of two existing LMs. A model succeeds by accurately gauging this target and generating a question at the specified difficulty. The environment consists of a questioner model, a set of boundary models, an answer key model, and a defined conversation pipeline. Each session consists of four steps, visualized in Figure 3, and results in a binary outcome: successful calibration or failure.

In the first step, two boundary models are selected from the boundary set. Because each boundary model is a distinct LM, the two models will generally differ in capability. The gap between their ability levels defines the target question difficulty for the session. The boundary set consists of n LMs, so the environment supports $\binom{n}{2}$ distinct difficulties ranges across sessions.

In the second step, the questioner must assess the target difficulty level by probing the capabilities of the two boundary models. It does this through k probing rounds, in which the questioner poses a question to both boundary models and receives their responses along with a short summary of each model’s approach. Throughout the probing phase, the questioner retains the full session history and knowledge of the overall task, while the boundary models are stateless and simply answer each question in isolation.

In the third step, the questioner uses the full probing context to generate a final question calibrated to discriminate between the two boundary models. The questioner’s role ends here; it does not observe the boundary models’ responses to the final question.

In the final step, the question is sent to both boundary models and the answer key model, each of which answers it independently with no additional task context. The session is scored as calibrated if exactly one boundary model matches the answer key, verified using standard equivalence-checking libraries. It does not matter which boundary model is correct, so long as the question targets the gap between them. This scoring mechanism has a useful

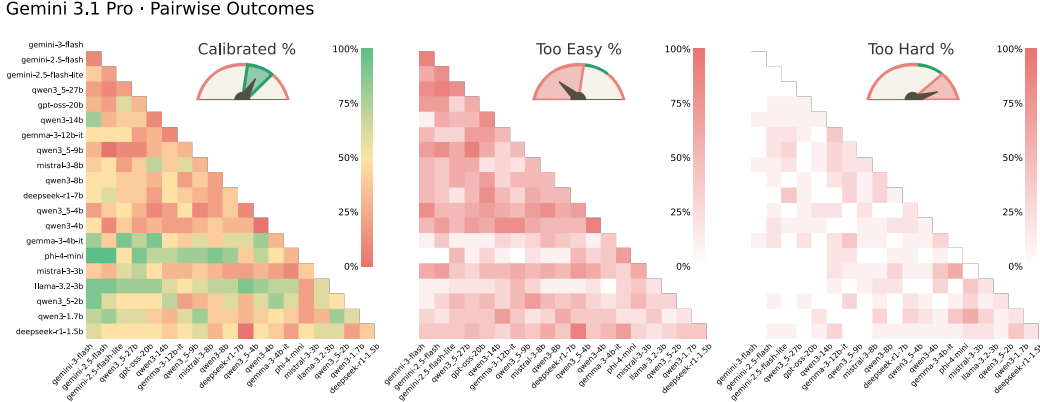


Figure 4: **Pairwise calibration outcomes for Gemini 3.1 Pro across all 190 boundary model pairs.** Boundary models are ordered by size from top to bottom and left to right. The left heatmap shows calibration rate, which is highest when the pair consists of one small and one large model (bottom-left region), representing wide target ranges. Performance is moderate for pairs of small models (bottom-right region), which define narrow but easier targets, and lowest for pairs of large models (upper region), where the target is both narrow and requires generating highly challenging questions.

implication for the answer key model. In standard automated benchmarks, the answer key must be at least as capable as the model being evaluated, since both are performing the same task. In Ask-E, the questioner and the answer key perform fundamentally different tasks: the questioner generates a calibrated question, while the answer key only solves it. The answer key therefore only needs to solve questions at the boundary models’ level. It may even be the same model type as the questioner, so long as it is a separate instance, ensuring the evaluated model cannot influence its own scoring.

We implement Ask-E using the Verifiers library [6], which supports efficient multi-turn environments for LMs. We limit the scope of this work to math questions, though the environment could readily extend to other domains by changing the model prompts and answer equivalence libraries. Full prompts for the questioner, boundary, and answer key models can be found in Appendix B.

4 Benchmarking on Ask-E

A primary use case of this environment is for benchmarking models. Calibration of ability level and generation of novel problems at a targeted level is challenging task requiring skill including and beyond those required for standard question answering.

4.1 Benchmark Implementation

Boundary-model sets. Our benchmark uses 20 boundary models: 17 open-weight checkpoints ranging from 1.5B to 27B parameters, plus 3 closed-API Gemini Flash models (Table 2). This yields $\binom{20}{2} = 190$ unordered pairs. Each questioner model interacts with every pair 10 times, for a total of $190 \times 10 = 1900$ sessions per evaluated questioner. Boundary models are queried with `temperature = 0.7`; all other sampling parameters are left at the backend’s default (vLLM defaults for open-weight models, API defaults for the Gemini models). Where applicable we set thinking effort to `low` and disable thinking entirely on models that expose a switch (`enable_thinking=false`) to encourage brevity. Open-weight models are served locally via vLLM [28] on 5 H200 GPUs (141 GB each); closed Gemini Flash models are queried through Google’s OpenAI-compatible endpoint.

Dialogue specifications. Each session consists of 4 probing rounds followed by 1 final round. The questioner is instructed to structure each reply into three sections, marked

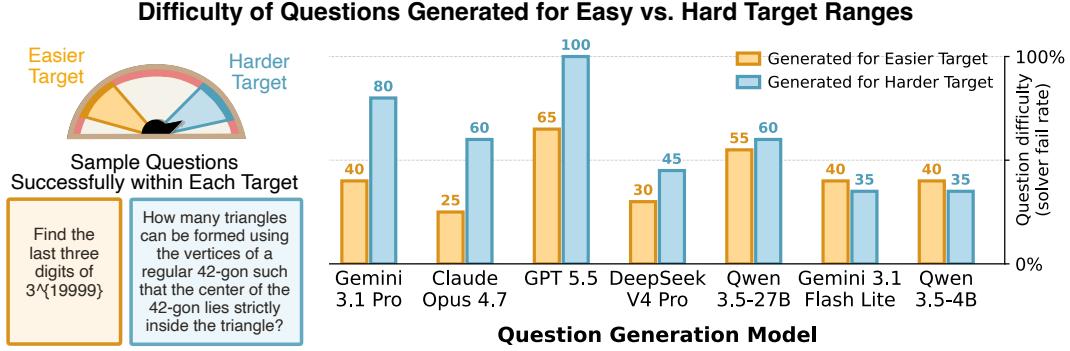


Figure 5: **Difficulty of questions generated for easy versus hard target ranges.** Each questioner model generates questions targeting an easier range (Qwen3-1.7B to Gemma-3-4b-it) and a harder range (Gemini-3-Flash to Gemini-2.5-Flash). An independent solver model (Qwen3.5-2B) attempts each set; its fail rate serves as a proxy for question difficulty. Sample questions successfully calibrated within each target are shown on the left.

by `#Reasoning#`, `#Draft#`, and `#Question#` tags, where it thinks through its next steps, drafts candidate questions and solutions, and outputs a final question respectively. Only the question is forwarded to the boundary models. Boundary models respond with a solution and boxed answer, of which only a short summary is forwarded back to the questioner. Full dialogue specifications including token budgets and parsing details are in Appendix F.

Scoring. We use Gemini 3.1 Pro [20] as the answer-key model. Final answers are extracted from boxed expressions and compared using the `math-verify` library [29] with string-equality fallbacks. A session is labeled *calibrated* when exactly one boundary model matches the answer key, *too_easy* when both match, and *too_hard* when neither matches. Full scoring details are in Appendix F.

4.2 Benchmark Results

Model Performance. We evaluate 5 models on our proposed benchmark that span a wide range of abilities: Gemini 3.1 Pro [20], Gemini 3.1 Flash Lite [19], Deepseek v4 Pro [12], Qwen3.5-27B [44] and Qwen3.5-4B[44]. Their performance is given in figure 2. Of the examined models, Gemini 3.1 Pro has the strongest performance, generating a calibrated question in 45.1% of sessions. Deepseek v4 Pro achieves 37.8%. Qwen3.5 27B and 4B achieve 37.6% and 28.1% respectively. Finally Gemini 3.1 flash light achieves the worst performance and 25.6% calibration rate. Figure 2 also breaks down outcomes into questions that are too easy, too hard, missing, or correctly calibrated. As calibration performance worsens, the proportion of questions that are too easy increases consistently, suggesting that the primary failure mode of weaker models is generating insufficiently difficult questions. These results highlight two takeaways. First, more capable models perform better, suggesting that calibration correlates with broader model quality. Second, even state-of-the-art models fall well below perfect performance, demonstrating that Ask-E has substantial headroom to measure future improvements. Generation configurations given in Appendix F.

Per Target Calibration. The 20 possible upper and lower bounds allow for 190 unique target difficulty ranges. Each questioner model is evaluated 10 times on each range. Figure 4 shows the calibration rate of Gemini 3.1 Pro for each pair, with boundary models ordered by size from top to bottom and left to right. Unsurprisingly, calibration is highest in the lower-left region of the heatmap, where one boundary model is small and the other large, creating wide target ranges that are easier to hit. Performance is moderate in the lower-right region, where both models are small. These ranges are narrow but correspond to easier questions that may not require as much skill to generate. Calibration is lowest in the upper region, where both boundary models are highly capable. These pairs define narrow

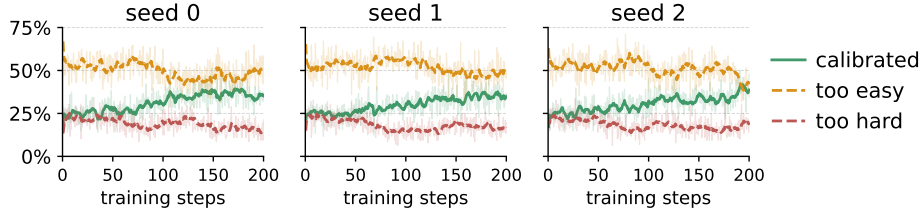


Figure 6: **Training curves across three seeds.** Calibration rate, too easy rate, and too hard rate over 200 training steps. All show improvement in calibration rate.

gaps that also demand generating very challenging questions. The too-easy and too-hard heatmaps mirror this pattern. The questioner is more likely to undershoot (too easy) when both boundary models are large, and to overshoot (too hard) when both are small.

Measuring Question Difficulty. Beyond calibration rate, we verify that models actually adjust question difficulty to the target rather than relying on a degenerate strategy, such as generating questions at a fixed difficulty that happens to fall within many ranges. To test this, we select two target ranges: an easy one (Qwen3-1.7B [54] vs. Gemma-3-4b-it [51]) and a hard one (Gemini-3-Flash [18] vs. [11]). Each questioner generates questions for both ranges, and an independent solver (Qwen3.5-2B) attempts each set. Its failure rate serves as a proxy for question difficulty. As shown in Figure 5, frontier questioner models produce substantially harder questions for the harder target, indicating genuine calibration, while less capable models show little or no difference between the two sets.

Extending the Benchmark We do not view the current benchmark formulation as fixed. As stronger models become available, the boundary set can be updated with more capable models, raising the difficulty ceiling without requiring any changes to the environment itself. Similarly, practical constraints on conversation length can be relaxed as technology advances, allowing for richer interactions and more nuanced calibration. We implement Ask-E so that boundary sets and dialogue specifications can be easily updated through configuration files, facilitating the evolution of the environment alongside the models it evaluates.

5 RL Training on Ask-E

In addition to a benchmark, Ask-E can be used as a training environment. Because the task is to generate calibrated questions, models can learn from the act of asking itself. As we show in Section 5.2, the skills developed through this training transfer to downstream tasks.

5.1 Training Implementation

Model and stack. We train Qwen3.5-4B (Instruct, no-thinking) as the questioner policy with the `prime-r1` [25] training library, which is natively compatible with the `verifiers` [6] environment used to define our question-generation rollout. A single training run uses 11 H200 GPUs, partitioned across three roles: 4 GPUs serve the questioner’s rollout copy (inference for in-flight rollouts), 4 GPUs run the trainer copy of the questioner together with optimizer state, and 3 GPUs host the small boundary-model inference endpoints.

Boundary set: weak only. To ensure that improvements come from the task itself rather than from learning from more capable models, the training boundary set contains only models that perform at or below the Qwen3.5-4B questioner on math benchmarks (the 12-model Small set in Appendix C, 1.5B–8B open-weight checkpoints). The questioner therefore never sees a stronger model’s reasoning, ruling out implicit distillation as the source of any gains. Additionally, all tokens from the boundary models are masked during training; only the questioner’s own assistant-role tokens contribute to the loss, so no boundary-model logits ever propagate back into the policy. Also note that the boundary model weights remain frozen through training.

Self-judge answer key. Whereas the benchmark uses a frontier model (Gemini 3.1 Pro) as the answer key, during training the questioner’s own current weights serve as the answer key. This ensures no information from a stronger model can leak into the policy through the reward channel. We use self-judging as a precaution, even though the questioner never sees the answer key’s output and the answer key only grades the boundary models.

Reward shape. Per-rollout reward is the sum of three components. The primary signal is a calibration reward on the final question’s outcome: +1.0 if exactly one boundary model is correct (calibrated), +0.2 if both are correct (too easy), and −0.2 if neither is (too hard). We find rewarding too easy questions more than too hard questions is helpful, possibly because unclear questions tend to result in a too hard label. A format penalty of −0.05 is applied for each turn that does not contain the required tags. Finally, an embedding-diversity bonus encourages varied questions, detailed in Appendix E.

Additional Training Configurations. We optimize the policy with CISPO [7], using a batch size of 256 rollouts per optimizer step with group size $G = 16$, covering 16 distinct boundary-pair examples per step. Training rollouts are shorter than the benchmark configuration for efficiency: 3 probing rounds plus 1 final question, with reduced token budgets. Training runs for 200 optimizer steps across 3 seeds. We report results aggregated across seeds. Full hyperparameters and optimizer details are in Appendix E.

5.2 Training Results

Over 200 steps, calibration rate improves consistently across all three seeds (Figure 6). More importantly, we find that the skills learned during training transfer to downstream tasks. To measure this, we evaluate the trained Ask-E model on four challenging math benchmarks, comparing performance to the Qwen3.5-4B model before training. Both models use the recommended Qwen3.5-4B sampling parameters for reasoning tasks in instruct mode (additional details in Appendix G). As shown in Table 1, Ask-E training improves pass@8 performance on all benchmarks with one tie, and avg@8 performance on 3 out of 4 benchmarks. Notably, this improvement occurs without any new math data, any interaction with stronger models, or any feedback on the correctness of the trained model’s answers.

Dataset	Metric	n	Qwen3.5-4B	with Ask-E
AIME 2022–2024 [2]	pass@8	90	87.78%	90.00%
AIME 2022–2024	avg@8	90	68.06%	67.36%
AIME 2025 [41]	pass@8	30	83.33%	83.33%
AIME 2025	avg@8	30	54.17%	57.22%
HMMT Feb 2025 [13]	pass@8	30	70.00%	72.22%
HMMT Feb 2025	avg@8	30	45.00%	45.97%
IMO-AnswerBench [37]	pass@8	400	70.25%	71.25%
IMO-AnswerBench	avg@8	400	43.91%	44.90%

Table 1: Math benchmark performance. Qwen3.5-4B Instruct accuracy before Ask-E training vs. accuracy after Ask-E RL training. Ask-E values are averaged across the 3 training seeds (full breakdown in Appendix H). **bold** marks improvements over the baseline. n indicated benchmark size. IMO AnswerBench is graded with the LLM judge.

6 Conclusion

We have introduced Ask-E, an environment that benchmarks and trains models on calibrated question generation rather than question answering. Our results demonstrate that this task correlates with general model ability, remains far from saturated, and produces transferable skills through training. As models continue to improve, matching and surpassing human ability, we believe the field will need to move beyond standard supervised evaluation and training regimes, and we hope Ask-E represents a step in that direction.

References

- [1] Abdelrahman Abouelenin, Atabak Ashfaq, Adam Atkinson, Hany Awadalla, Nguyen Bach, Jianmin Bao, Alon Benhaim, Martin Cai, Vishrav Chaudhary, Congcong Chen, et al. Phi-4-mini technical report: Compact yet powerful multimodal language models via mixture-of-loras. *arXiv preprint arXiv:2503.01743*, 2025.
- [2] AI-MO. Aimo validation aime. <https://huggingface.co/datasets/AI-MO/aimo-validation-aime>, 2024. Licensed under Apache-2.0.
- [3] Said Al Faraby, Ade Romadhony, et al. Analysis of llms for educational question classification and generation. *Computers and Education: Artificial Intelligence*, 7:100298, 2024.
- [4] Anthropic. System card: Claude opus 4.7. Technical report, April 2026. URL <https://cdn.sanity.io/files/4zrzovbb/website/037f06850df7f871e206dad004c3db5fd50340.pdf>.
- [5] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.
- [6] William Brown. Verifiers: Environments for llm reinforcement learning. <https://github.com/PrimeIntellect-ai/verifiers>, 2025. Commit abcdefg • accessed DD Mon YYYY.
- [7] Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao Wang, Cheng Zhu, et al. Minimax-m1: Scaling test-time compute efficiently with lightning attention. *arXiv preprint arXiv:2506.13585*, 2025.
- [8] Lili Chen, Mihir Prabhudesai, Katerina Fragkiadaki, Hao Liu, and Deepak Pathak. Self-questioning language models. *arXiv preprint arXiv:2508.03682*, 2025.
- [9] Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. Self-play fine-tuning converts weak language models to strong language models. *arXiv preprint arXiv:2401.01335*, 2024.
- [10] Konstantin Chernyshev, Vitaliy Polshkov, Ekaterina Artemova, Alex Myasnikov, Vlad Stepanov, Alexei Miasnikov, and Sergei Tilga. U-math: A university-level benchmark for evaluating mathematical skills in llms. *arXiv preprint arXiv:2412.03205*, 2024.
- [11] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.
- [12] DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026.
- [13] Jasper Dekoninck, Nikola Jovanović, Tim Gehringer, Kári Rögvalddson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: Matharena as an evaluation platform for mathematics with llms. 2026. URL <https://arxiv.org/abs/2605.00674>.
- [14] Sabina Elkins, Ekaterina Kochmar, Iulian Serban, and Jackie CK Cheung. How useful are educational questions generated by large language models? In *International Conference on Artificial Intelligence in Education*, pages 536–542. Springer, 2023.
- [15] Benjamin Feuer, Chiung-Yi Tseng, Astitwa Sarthak Lathe, Oussama Elachqar, and John P Dickerson. When judgment becomes noise: How design failures in llm judge benchmarks silently undermine validity. *arXiv preprint arXiv:2509.20293*, 2025.
- [16] Suhaas Garre, Erik Knutsen, Sushant Mehta, and Edwin Chen. Riemann-bench: A benchmark for moonshot mathematics. *arXiv preprint arXiv:2604.06802*, 2026.

- [17] Scott Geng, Hamish Ivison, Chun-Liang Li, Maarten Sap, Jerry Li, Ranjay Krishna, and Pang Wei Koh. The delta learning hypothesis: Preference tuning on weak data can yield strong gains. *arXiv preprint arXiv:2507.06187*, 2025.
- [18] Google DeepMind. Gemini 3 flash model card. Technical report, December 2025. URL <https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Flash-Model-Card.pdf>.
- [19] Google DeepMind. Gemini 3.1 flash-lite model card. Technical report, 2026. URL <https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-1-Flash-Lite-Model-Card.pdf>.
- [20] Google DeepMind. Gemini 3.1 pro model card. Technical report, February 2026. URL <https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-1-Pro-Model-Card.pdf>.
- [21] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.
- [22] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- [23] Zhang-Wei Hong, Idan Shenfeld, Tsun-Hsuan Wang, Yung-Sung Chuang, Aldo Pareja, James Glass, Akash Srivastava, and Pulkit Agrawal. Curiosity-driven red-teaming for large language models. *arXiv preprint arXiv:2402.19464*, 2024.
- [24] Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-zero: Self-evolving reasoning llm from zero data. *arXiv preprint arXiv:2508.05004*, 2025.
- [25] Prime Intellect. Prime-rl, 2025. URL <https://github.com/PrimeIntellect-ai/prime-rl>.
- [26] Douwe Kiela, Max Bartolo, Yixin Nie, Divyansh Kaushik, Atticus Geiger, Zhengxuan Wu, Bertie Vidgen, Grusha Prasad, Amanpreet Singh, Pratik Ringshia, et al. Dynabench: Rethinking benchmarking in nlp. In *Proceedings of the 2021 conference of the North American chapter of the Association for Computational Linguistics: human language technologies*, pages 4110–4124, 2021.
- [27] Jakub Grudzien Kuba, Mengting Gu, Qi Ma, Yuandong Tian, Vijai Mohan, and Jason Chen. Language self-play for data-free training. *arXiv preprint arXiv:2509.07414*, 2025.
- [28] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.
- [29] Hynek Kydliček. Math-verify: Math verification library, 2025. URL <https://github.com/huggingface/math-verify>.
- [30] Nathan Lambert, Valentina Pyatkin, Jacob Morrison, Lester James Validad Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, et al. Rewardbench: Evaluating reward models for language modeling. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 1755–1797, 2025.
- [31] Belinda Z Li, Been Kim, and Zi Wang. Questbench: Can llms ask the right question to acquire information in reasoning tasks? *arXiv preprint arXiv:2503.22674*, 2025.
- [32] Xiang Lisa Li, Farzaan Kaiyom, Evan Zheran Liu, Yifan Mai, Percy Liang, and Tatsunori Hashimoto. Autobench: Towards declarative benchmark construction. *arXiv preprint arXiv:2407.08351*, 2024.

- [33] Alexander H Liu, Kartik Khandelwal, Sandeep Subramanian, Victor Jouault, Abhinav Rastogi, Adrien Sadé, Alan Jeffares, Albert Jiang, Alexandre Cahill, Alexandre Gavaudan, et al. Ministral 3. *arXiv preprint arXiv:2601.08584*, 2026.
- [34] Bo Liu, Leon Guertler, Simon Yu, Zichen Liu, Penghui Qi, Daniel Balcells, Mickel Liu, Cheston Tan, Weiyan Shi, Min Lin, et al. Spiral: Self-play on zero-sum games incentivizes reasoning via multi-agent multi-turn reinforcement learning. *arXiv preprint arXiv:2506.24119*, 2025.
- [35] Bo Liu, Chuanyang Jin, Seungone Kim, Weizhe Yuan, Wenting Zhao, Ilya Kulikov, Xian Li, Sainbayar Sukhbaatar, Jack Lanchantin, and Jason Weston. Spice: Self-play in corpus environments improves reasoning. *arXiv preprint arXiv:2510.24684*, 2025.
- [36] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.
- [37] Minh-Thang Luong, Dawsen Hwang, Hoang H Nguyen, Golnaz Ghiasi, Yuri Chervonyi, Insuk Seo, Junsu Kim, Garrett Bingham, Jonathan Lee, Swaroop Mishra, et al. Towards robust mathematical reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 35406–35430, 2025.
- [38] Davide Mazzaccara, Alberto Testoni, and Raffaella Bernardi. Learning to ask informative questions: Enhancing llms with preference optimization and expected information gain. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 5064–5074, 2024.
- [39] Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*, 2024.
- [40] OpenAI. gpt-oss-120b gpt-oss-20b model card, 2025. URL <https://arxiv.org/abs/2508.10925>.
- [41] OpenCompass. Aime 2025. <https://huggingface.co/datasets/opencompass/AIME2025>, 2025.
- [42] Arkil Patel, Siva Reddy, and Dzmitry Bahdanau. How to get your llm to generate challenging problems for evaluation. *arXiv preprint arXiv:2502.14678*, 2025.
- [43] Daniel M Pedrozo, Telma W de L Soares, and Bryan LM de Oliveira. Do reasoning models ask better questions? a formal information-theoretic analysis on multi-turn llm games. *arXiv preprint arXiv:2601.17716*, 2026.
- [44] Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL <https://qwen.ai/blog?id=qwen3.5>.
- [45] Vedant Shah, Dingli Yu, Kaifeng Lyu, Simon Park, Jiatong Yu, Yinghui He, Nan Rosemary Ke, Michael Mozer, Yoshua Bengio, Sanjeev Arora, et al. Ai-assisted generation of difficult math questions. *arXiv preprint arXiv:2407.21009*, 2024.
- [46] Aaditya Singh et al. OpenAI GPT-5 system card. 2025. URL <https://arxiv.org/abs/2601.03267>.
- [47] Gaurav Srivastava, Aafiya Shamshad Hussain, Zhenyu Bi, Swastik Roy, Priya Pitre, Meng Lu, Morteza Ziyadi, and Xuan Wang. Beyondbench: Contamination-resistant evaluation of reasoning in language models. In *The Fourteenth International Conference on Learning Representations*.
- [48] Gaurav Srivastava, Zhenyu Bi, Meng Lu, and Xuan Wang. Debate, train, evolve: Self-evolution of language model reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 32752–32798, 2025.

- [49] Megha Srivastava and Noah Goodman. Question generation for adaptive education. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 692–701, 2021.
- [50] Yiyu Sun, Shawn Hu, Georgia Zhou, Ken Zheng, Hannaneh Hajishirzi, Nouha Dziri, and Dawn Song. Omega: Can llms reason outside the box in math? evaluating exploratory, compositional, and transformative generalization. *arXiv preprint arXiv:2506.18880*, 2025.
- [51] Gemma Team. Gemma 3. 2025. URL <https://goo.gle/Gemma3Report>.
- [52] Wenxuan Wang, Shi Juluan, Zixuan Ling, Yuk-Kit Chan, Chaozheng Wang, Cheryl Lee, Youliang Yuan, Jen-tse Huang, Wenxiang Jiao, and Michael R Lyu. Learning to ask: When llm agents meet unclear instruction. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 21784–21795, 2025.
- [53] Zichao Wang, Andrew S Lan, Weili Nie, Andrew E Waters, Phillip J Grimaldi, and Richard G Baraniuk. Qg-net: a data-driven question generation model for educational content. In *Proceedings of the fifth annual ACM conference on learning at scale*, pages 1–10, 2018.
- [54] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.
- [55] John Yang, Kilian Lieret, Joyce Yang, Carlos E Jimenez, Ofir Press, Ludwig Schmidt, and Diyi Yang. Codeclash: Benchmarking goal-oriented software engineering. *arXiv preprint arXiv:2511.00839*, 2025.
- [56] Ziyi Yang, Weizhou Shen, Chenliang Li, Ruijun Chen, Fanqi Wan, Ming Yan, Xiaojun Quan, and Fei Huang. Spell: Self-play reinforcement learning for evolving long-context language models. *arXiv preprint arXiv:2509.23863*, 2025.
- [57] Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason E Weston. Self-rewarding language models. In *Forty-first International Conference on Machine Learning*, 2024.
- [58] Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Matthieu Lin, Shenzi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data. *arXiv preprint arXiv:2505.03335*, 2025.
- [59] Yang Zhou, Hongyi Liu, Zhuoming Chen, Yuandong Tian, and Beidi Chen. Gsm-infinite: How do your llms behave over infinitely increasing context length and reasoning complexity? *arXiv preprint arXiv:2502.05252*, 2025.
- [60] Kaijie Zhu, Jiaao Chen, Jindong Wang, Neil Zhenqiang Gong, Diyi Yang, and Xing Xie. Dyval: Dynamic evaluation of large language models for reasoning tasks. *arXiv preprint arXiv:2309.17167*, 2023.
- [61] Qingfeng Zhuge, Han Wang, and Xuyang Chen. Twinstar: A novel design for enhanced test question generation using dual-llm engine. *Applied Sciences*, 15(6):3055, 2025.

A Limitations and Broader Impact

Limitations. We identify two primary limitations of this work. First, there exist degenerate strategies that could satisfy the calibration criterion without requiring genuine mathematical skill. For example, a questioner could pose a question with a random or ambiguous answer, such as asking models to flip a coin, which would produce answer variance without targeting any meaningful capability gap. While we have not observed this behavior in practice, it remains a possibility and may require mitigations such as running multiple answer key models and requiring their agreement. Second, Ask-E is best suited for developing and measuring skills that a model already possesses to some degree. A model must be capable of generating reasonable questions to begin with in order to benefit from training in this environment, limiting its applicability for teaching entirely new skills from scratch.

Broader Impacts. Our work is primarily foundational research into model evaluation and training. On the positive side, Ask-E offers a scalable approach to benchmarking and improving models that reduces dependence on expensive human-generated datasets. On the negative side, models that improve at generating calibrated questions could in principle be used to generate targeted evaluation or training data in ways that amplify existing biases in the boundary models. However, we view this risk as minimal given the current scope of the work, which is limited to math question generation.

B Environment Prompts

The environment runs a multi-turn dialogue between three roles:

- **Questioner** — generates probing questions and a final differentiating question. Receives a system prompt once, then a structured user message every turn.
- **Two boundary models** — answer each generated question. They share a single system prompt, see only the current question (no memory across turns), and reply with a boxed answer plus a brief summary that is fed back to the questioner.
- **Answer-key model** — independently solves the final question to provide ground truth for the reward function. It uses a stripped-down “grade” system prompt: no `#Summary#` is needed because only the boxed answer is consumed.

Each rollout has N probing turns followed by one final turn. The questioner’s reply on every turn is parsed for three sections: `#Reasoning#` (private), `#Draft#` (private), and `#Question#` (forwarded verbatim to both boundary models).

Questioner prompts

System prompt (set once, at the start of every rollout). The questioner is told the task, the constraints on its questions, and the three-section response format it must emit every turn.

```
Questioner system prompt

TASK: Find a math question that exactly one of two models can solve correctly.

CONSTRAINTS:
- Questions must have a single unambiguous answer expressible in \boxed{} format
  (integer, fraction, decimal, or short expression).
- Answering models see ONLY the current question. They have no memory of prior
  rounds.
- 'Calibrated' means exactly one model is correct - both right or both wrong
  does not count.

STRATEGY:
You have {num_probing} probing rounds before your final question. Use them to
find the difficulty boundary where one model fails and the other succeeds.
```

RESPONSE FORMAT (every turn, probing and final):

#Reasoning#

Reason through the previous round (if any) and decide what to target next. This is your private scratchpad - NOT sent to the models. Each round, consider:

- What concepts or difficulty levels seem to separate the two models
- What kinds of questions could effectively target that gap

#Draft#

Propose a candidate question and work through its solution end-to-end. If the answer is ambiguous, has multiple valid values, or depends on unstated assumptions, draft another and try again. End with `\boxed{answer}`. This section is NOT sent to the answering models.

#Question#

Your final question, copied cleanly from the draft. Everything after the `#Question#` tag is sent verbatim to both models, so stop generating once you've finished writing the question itself - do not include solutions, reasoning, hints, or any other text after the question.

Output exactly one `#Reasoning#` section, one `#Draft#` section, and one `#Question#` section per turn, in that order.

First-turn user prompt. On round 1 the questioner has no prior boundary-model responses to react to, so it is just shown the structure once more and asked for a probing question.

First-turn user prompt

Use `#Reasoning#` to plan your approach: think about what concepts or difficulty levels might separate the two models and what kinds of questions could effectively target those gaps. Use `#Draft#` to propose a candidate question and work through its solution. Then commit to your final probing question under `#Question#`.

EXAMPLE STRUCTURE:

`#Reasoning#`

(your reasoning)

`#Draft#`

(your candidate question + worked-out solution)

`#Question#`

(your question)

Probing-round feedback (rounds 2...N). After each probing round the questioner is given both boundary models' post-`#Summary#` replies and asked for the next probing question.

Probing feedback (rounds 2..N)

Model 1 answered:

{boundary_a_answer}

Model 2 answered:

{boundary_b_answer}

Use `#Reasoning#` to process these results: assess who got it right, what that reveals about each model's capabilities, and consider what to target next. Use `#Draft#` to propose a candidate question and work through its solution. Then commit to probing question number `{next_question_num}` under `#Question#`.

EXAMPLE STRUCTURE:

```
#Reasoning#
(your reasoning)

#Draft#
(your candidate question + worked-out solution)

#Question#
(your question)
```

Final-question prompt. On the last turn the questioner is told to synthesize and emit its final differentiating question.

Final-question prompt

Model 1 answered:
{boundary_a_answer}

Model 2 answered:
{boundary_b_answer}

This was your last probing round. Use `#Reasoning#` to:

- Synthesize what you have learned about each model's strengths and weaknesses
- Identify the specific gap your final question should exploit
- Brainstorm several candidate final questions and work through their solutions
- Pick the candidate most likely to be solved by exactly one model

You may reuse a probing question if you believe it discriminates well. Use `#Draft#` to propose your final question and work through its solution. Then commit to your final question under `#Question#`.

EXAMPLE STRUCTURE:

```
#Reasoning#
(your reasoning)

#Draft#
(your candidate question + worked-out solution)

#Question#
(your question)
```

Recovery prompts. If the previous response did not contain a parseable `#Question#`, the questioner gets a one-turn recovery prompt that nudges it to produce one. There are two variants — for probing rounds and for the final round — each with two sub-cases (truncated vs. malformed).

No-question recovery (probing round)

[truncated case]
No question was found in your previous response - you ran out of tokens before producing a `#Question#` tag, and thus the models had no question to answer. Keep your reasoning shorter this time.

[malformed case]
No question was found in your previous response - you did not use the #Question# tag, and thus the models had no question to answer. You must output your question under the #Question# tag exactly as shown below.

Ask probing question number {next_question_num} of {num_probing} under #Question#.

EXAMPLE STRUCTURE:

```
#Reasoning#  
(your reasoning)  
  
#Draft#  
(your candidate question + worked-out solution)  
  
#Question#  
(your question)
```

No-question recovery (final round)

[truncated case]
No question was found in your previous response - you ran out of tokens before producing a #Question# tag. Keep your reasoning much shorter this time.

[malformed case]
No question was found in your previous response - you did not use the #Question# tag. You must output your question under the #Question# tag exactly as shown below.

This was your last probing round. Use #Reasoning# to synthesize what you have learned, output the worked-out answer to your final question under #Solution#, and present your final differentiating question under #Question#.

EXAMPLE STRUCTURE:

```
#Reasoning#  
(your reasoning)  
  
#Draft#  
(your candidate question + worked-out solution)  
  
#Question#  
(your question)
```

Boundary-model prompts

Probing system prompt. Used during the probing rounds. Boundary models output a brief #Summary# that is forwarded to the questioner so it can diagnose each model's reasoning without sending raw chains of thought back upstream.

Boundary system prompt (probing)

Answer the given question. Work through the problem, keeping your reasoning concise. When you are done, output #Summary# followed by your final answer in `\boxed{}` and a summary (3-10 sentences) of how you arrived at it.

Question suffix (probing). Appended to every `#Question#` forwarded to the boundary models during probing rounds, to reinforce the format.

Boundary question suffix (probing)

```
When you are done, output #Summary# followed by your final answer in \boxed{}  
and a summary (3-10 sentences) of your approach.
```

Answer-key model prompts

Grading system prompt. The answer-key model only needs to emit a boxed final answer for the reward function — no summary section. The same prompt is reused if a boundary model is invoked in pure “grade” mode (e.g. when self-judge is enabled).

Grading system prompt

```
Answer the given question. Think step-by-step but keep your reasoning concise.  
Output the final solution in \boxed{}
```

Grading question suffix

```
Output the final solution in \boxed{}
```

C Boundary Model Sets

We work with two boundary-model collections. The **Full set** is used at evaluation time to benchmark questioner models; it spans 20 models from 1.5B parameters up to a frontier closed-API class. The **Small set** is the subset used during RL training: 12 open-weight models in the 1.5B–8B range. Every Small-set model is also in the Full set.

Model	Family	Params	Full	Small
deepseek-r1-1.5b [22]	DeepSeek	1.5B	✓	✓
qwen3-1.7b [54]	Qwen	1.7B	✓	✓
qwen3.5-2b [44]	Qwen	2.0B	✓	✓
llama-3.2-3b [21]	Llama	3.0B	✓	✓
ministral-3-3b [33]	Mistral	3.0B	✓	✓
phi-4-mini [1]	Phi	3.84B	✓	✓
qwen3.5-4b [44]	Qwen	4.0B	✓	✓
qwen3-4b [54]	Qwen	4.0B	✓	✓
gemma-3-4b-it [51]	Gemma	4.0B	✓	✓
deepseek-r1-7b [22]	DeepSeek	7.0B	✓	✓
qwen3-8b [54]	Qwen	8.0B	✓	✓
ministral-3-8b [33]	Mistral	8.0B	✓	✓
qwen3.5-9b [44]	Qwen	9.0B	✓	
gemma-3-12b-it [51]	Gemma	12.0B	✓	
qwen3-14b [54]	Qwen	14.0B	✓	
gpt-oss-20b [40]	GPT-OSS	20.0B	✓	
qwen3.5-27b [44]	Qwen	27.0B	✓	
gemini-2.5-flash-lite [11]	Gemini	—	✓	
gemini-2.5-flash [11]	Gemini	—	✓	
gemini-3-flash [18]	Gemini	—	✓	

Table 2: Boundary-model sets. The **Full set** (20 models) is used for benchmarking; the **Small set** (12 models) is the training subset. Parameter counts are reported in billions; closed-API Gemini Flash models do not have published sizes. Rows are ordered by parameter count, with the Small set on top, the larger open-weight models in the middle, and closed-API models at the bottom.

Model	Family	License
deepseek-r1-1.5b	DeepSeek	MIT + Apache 2.0
qwen3-1.7b	Qwen	Apache 2.0
qwen3.5-2b	Qwen	Apache 2.0
llama-3.2-3b	Llama	Llama 3.2 Community
ministral-3-3b	Mistral	Apache 2.0
phi-4-mini	Phi	MIT
qwen3.5-4b	Qwen	Apache 2.0
qwen3-4b	Qwen	Apache 2.0
gemma-3-4b-it	Gemma	Gemma Terms of Use
deepseek-r1-7b	DeepSeek	MIT + Apache 2.0
qwen3-8b	Qwen	Apache 2.0
ministral-3-8b	Mistral	Apache 2.0
qwen3.5-9b	Qwen	Apache 2.0
gemma-3-12b-it	Gemma	Gemma Terms of Use
qwen3-14b	Qwen	Apache 2.0
gpt-oss-20b	GPT-OSS	Apache 2.0
qwen3.5-27b	Qwen	Apache 2.0
gemini-2.5-flash-lite	Gemini	Proprietary (API ToS)
gemini-2.5-flash	Gemini	Proprietary (API ToS)
gemini-3-flash	Gemini	Proprietary (API ToS)

Table 3: Licenses for each model in the boundary-model sets.

D Example Rollout

This section shows one full rollout from the Gemini 3.1 Pro benchmark on the boundary pair **mistral-3-3b** vs. **qwen3_5-27b** (sample 2 of 10). For each turn we show the questioner's complete output (its `#Reasoning#`, `#Draft#`, and `#Question#` sections), the question forwarded verbatim to both boundary models, and the *summaries* returned by each boundary model. On the final turn we additionally show the answer-key model's full response, which the reward function uses as ground truth.

A boundary summary is constructed from the boundary model's full response by taking everything after the last `#Summary#` (or `#Output#`) tag, truncated to 2000 characters; the `\boxed{}` final answer is always preserved even if it falls outside the truncation window or sits above the `#Summary#` tag. If the model emits no `#Summary#/#Output#` tag, only the `\boxed{}` answer is forwarded; if neither is present a placeholder string is forwarded instead. The full chains of thought are never sent back to the questioner.

Turn 1 (probing)

Questioner output

`#Reasoning#`

I'll start with a standard modular exponentiation problem to gauge their capabilities in number theory. The problem requires understanding of Euler's totient theorem or finding patterns in powers modulo 100. It's a classic competition-style problem, so both models might get it right, but if one fails, it gives me a clear idea of their upper bounds on number theory tasks.

`#Draft#`

Find the remainder when 2023^{2023} is divided by 100.

Solution:

We need to find $2023^{2023} \pmod{100}$. First, $2023 \equiv 23 \pmod{100}$. Since $\gcd(23, 100) = 1$, by Euler's totient theorem with $\phi(100) = 40$ we get $23^{40} \equiv 1 \pmod{100}$, so $23^{2023} \equiv 23^{23} \pmod{100}$. Computing successive squares: $23^2 \equiv 29$, $23^4 \equiv 41$, $23^{16} \equiv 61$. Then $23^{16} \cdot 23^4 \equiv 61 \cdot 41 = 2501 \equiv 1$, so $23^{23} \equiv 23^3 \equiv 67 \pmod{100}$.
`\boxed{67}`

`#Question#`

Find the remainder when 2023^{2023} is divided by 100.

Question forwarded to both boundary models

Find the remainder when 2023^{2023} is divided by 100.

mistral-3-3b summary returned to questioner

To find the remainder when 2023^{2023} is divided by 100, we first reduced the base modulo 100 to simplify the expression to $23^{2023} \pmod{100}$. Leveraging Euler's theorem, we simplified the exponent modulo $\phi(100)=40$, reducing the problem to computing $23^{23} \pmod{100}$. By using the Chinese Remainder Theorem to break down 100 into 4 and 25, we effectively solved the problem modulo these factors separately. Computing $23^{23} \pmod{4}$ and $23^{23} \pmod{25}$ gave us congruences $3 \pmod{4}$ and $17 \pmod{25}$, respectively. Combining these results using the CRT yielded the final solution, 67.

\boxed{67}

qwen3_5-27b summary returned to questioner

To find $2023^{2023} \pmod{100}$, we first reduced the base to $23 \pmod{100}$. Using Euler's Totient Theorem, we determined that the powers of 23 repeat with a period of $\phi(100)=40$. We reduced the exponent 2023 modulo 40 to get 23, simplifying the problem to $23^{23} \pmod{100}$. By calculating powers of 23 (specifically finding that $23^{20} \equiv 1$), we reduced the calculation to $23^3 \pmod{100}$. Finally, computing $23^3 = 12167$ gave a remainder of 67.

\boxed{67}

Turn 2 (probing)

Questioner output

#Reasoning#

Both models successfully solved the modular exponentiation problem, showing that they understand the Chinese Remainder Theorem, Euler's totient theorem, and basic modular arithmetic. I will now step up to a sequence problem that requires recognizing a pattern or finding a linear recurrence from a non-linear one. Nonlinear recurrences of the form $a_{n+1} = (a_n^2 + c)/a_{n-1}$ often reduce to a simple linear recurrence. Testing this will evaluate their algebraic manipulation and pattern recognition skills.

#Draft#

A sequence is defined by $a_1 = 1$, $a_2 = 2$, and $a_{n+1} = (a_n^2 + 1)/a_{n-1}$ for $n \geq 2$. Find a_{10} .

Calculate: $a_3 = 5$, $a_4 = 13$, $a_5 = 34$. The terms 1,2,5,13,34 are odd-indexed Fibonacci numbers. Equivalently, the sequence satisfies $a_{n+1} = 3a_n - a_{n-1}$. Iterating: $a_6=89$, $a_7=233$, $a_8=610$, $a_9=1597$, $a_{10}=4181$.

\boxed{4181}

#Question#

A sequence is defined by $a_1 = 1$, $a_2 = 2$, and $a_{n+1} = (a_n^2 + 1)/a_{n-1}$ for $n \geq 2$. Find the value of a_{10} .

Question forwarded to both boundary models

A sequence is defined by $a_1 = 1$, $a_2 = 2$, and $a_{n+1} = (a_n^2 + 1)/a_{n-1}$ for $n \geq 2$. Find the value of a_{10} .

mistral-3-3b summary returned to questioner

The sequence was computed step-by-step using the recurrence relation $a_{n+1} = (a_n^2 + 1)/a_{n-1}$ with initial terms $a_1 = 1$ and $a_2 = 2$. An earlier error was identified in a_8 , which was corrected to 610. Correcting back through earlier terms resolved inconsistencies. Finally, a_{10} was computed exactly as 4181 by careful division.

\boxed{4181}

qwen3_5-27b summary returned to questioner

First, I calculated the first few terms of the sequence using the given recurrence relation $a_{n+1} = (a_n^2 + 1)/a_{n-1}$ to find $a_3=5$, $a_4=13$, $a_5=34$, $a_6=89$. Next, I observed that the terms satisfy the linear recurrence relation $a_{n+1} = 3a_n - a_{n-1}$, which is much easier to iterate than the original non-linear formula. Using this linear pattern, I computed the subsequent terms: $a_7=233$, $a_8=610$, $a_9=1597$. Finally, I calculated $a_{10} = 3(1597) - 610 = 4181$. Alternatively, recognizing the sequence as every second Fibonacci number ($a_n = F_{2n-1}$), I identified that $a_{10} = F_{19}$, which confirms the result is 4181.

`\boxed{4181}`

Turn 3 (probing)

Questioner output

#Reasoning#

Both models handled the sequence problem correctly, recognizing the linear recurrence or carefully computing terms. To push them further, I'll test their combinatorics and dynamic programming capabilities with a string avoidance problem. I'll ask for the number of sequences of a certain length avoiding a specific substring. This can be modeled with a recurrence relation or a state machine. It is straightforward but error-prone if the recurrence is built incorrectly.

#Draft#

Find the number of sequences of length 8 formed by the letters A, B, and C such that the sequence does not contain the consecutive substring "AB".

Solution:

Let S_n be the total number of valid sequences of length n , and let a_n , b_n , c_n be the count ending in A, B, C respectively. Appending A or C is unrestricted; appending B is forbidden after A. This gives $S_{n+1} = 3S_n - S_{n-1}$ with $S_1=3$, $S_2=8$. Iterating: $S_3=21$, $S_4=55$, $S_5=144$, $S_6=377$, $S_7=987$, $S_8=2584$.

`\boxed{2584}`

#Question#

Find the number of sequences of length 8 formed by the letters A, B, and C such that the sequence does not contain the consecutive substring "AB".

Question forwarded to both boundary models

Find the number of sequences of length 8 formed by the letters A, B, and C such that the sequence does not contain the consecutive substring "AB".

mistral-3-3b summary returned to questioner

`\boxed{1393}`

This problem was solved using dynamic programming and recurrence relations to count valid sequences of length 8 formed by letters A, B, and C that avoid the forbidden substring "AB." The approach involved defining three separate counts for sequences ending in A, B, or C, then updating these counts iteratively based on the restrictions --- specifically, ensuring that a sequence cannot end with "AB." By breaking the problem into smaller subproblems (sequences of length 1 through 8), we accumulated the counts

step-by-step, leveraging the relationships between sequences ending in A, B, or C. The solution efficiently explores all valid configurations without repetition, ensuring correctness through a systematic recurrence. The totals were computed up to length 8, yielding the final answer of 1,393 valid sequences.

qwen3_5-27b summary returned to questioner

To solve this problem, we defined a recurrence relation based on the last character of the sequence. Let a_n be the total number of valid sequences of length n . We observed that a sequence ending in 'A' or 'C' can be formed by appending that letter to any valid sequence of length $n-1$, while a sequence ending in 'B' can only be formed by appending 'B' to a sequence not ending in 'A'. This led to the recurrence $a_n = 3a_{n-1} - a_{n-2}$. Starting with base cases $a_1 = 3$ and $a_2 = 8$, we iteratively calculated the values up to $n=8$. The sequence of counts is 3, 8, 21, 55, 144, 377, 987, 2584.

$\boxed{2584}$

Turn 4 (probing)

Questioner output

#Reasoning#

Model 2 perfectly recognized the recurrence relation and correctly computed the values, while Model 1 failed, likely due to a flawed DP state setup. This tells me Model 2 is quite strong at standard competition combinatorics and recursive logic, whereas Model 1 struggles when it has to correctly abstract a state machine. I will now test a probability problem involving expected value and Markov chains/state transitions. A classic problem is the expected number of rolls to get consecutive specific outcomes.

#Draft#

A fair standard 6-sided die is rolled repeatedly. What is the expected number of rolls needed to obtain two consecutive 6s?

Solution: Let E_0 be the expected rolls from the start and E_1 from after rolling a 6. Then $E_0 = 1 + (5/6)E_0 + (1/6)E_1$ and $E_1 = 1 + (5/6)E_0 + (1/6) \cdot 0$. Solving gives $E_0 = 42$.

$\boxed{42}$

#Question#

A fair standard 6-sided die is rolled repeatedly. What is the expected number of rolls needed to obtain two consecutive 6s?

Question forwarded to both boundary models

A fair standard 6-sided die is rolled repeatedly. What is the expected number of rolls needed to obtain two consecutive 6s?

mistral-3-3b summary returned to questioner

The problem involves calculating the expected number of die rolls needed to obtain two consecutive sixes. This was approached by modeling the process using Markov states to capture the history of consecutive sixes observed. We defined two states: one where no six or only one six has appeared (State

A), and another where a six has just appeared (State B). We set up equations for the expected number of rolls from each state, considering the probabilities of transitioning between states upon each roll. By solving the system of equations, we found the expected number of rolls needed to reach two consecutive sixes starting from the initial state to be 42.

\boxed{42}

qwen3_5-27b summary returned to questioner

To solve this problem, we defined two states: E representing the expected rolls needed from the start, and E_1 representing the expected rolls needed after rolling a single 6. We set up a system of linear equations based on the transition probabilities between these states. Specifically, rolling a non-6 keeps us in the current state (or resets us to the start), while rolling a 6 advances us toward the goal. Solving the system $E = 1 + (1/6)E_1 + (5/6)E$ and $E_1 = 1 + (5/6)E$ yielded $E = 42$.

\boxed{42}

Turn 5 (final)

Questioner output

#Reasoning#

Through the probing rounds, both models demonstrated strong capabilities in modular arithmetic, recurrence relations, and expected value with state machines. However, in Round 3, Model 1 failed on a combinatorics problem involving strings that avoid a specific consecutive substring. Model 1 seems to struggle with setting up the correct dynamic programming states or principle of inclusion-exclusion for counting sequences with forbidden substrings. Model 2, on the other hand, excels at recognizing the exact recurrence relation and calculating the answer cleanly.

To exploit this gap, I will give a similar combinatorics string-avoidance problem but with slightly different parameters to ensure it cannot guess from memory. A 4-letter alphabet sequence of length 6 avoiding "AB" allows for verification via Principle of Inclusion-Exclusion (PIE) or a recurrence relation. Model 2 will likely solve it using the recurrence $S_n = 4 S_{n-1} - S_{n-2}$, while Model 1 will likely set up the DP incorrectly again.

#Draft#

Find the number of sequences of length 6 formed by the letters A, B, C, and D such that the sequence does not contain the consecutive substring "AB".

Solution: With four letters and one forbidden adjacency the recurrence becomes $S_{n+1} = 4 S_n - S_{n-1}$, with $S_0=1$, $S_1=4$. Iterating: $S_2=15$, $S_3=56$, $S_4=209$, $S_5=780$, $S_6=2911$. Verified independently by inclusion-exclusion over the 5 possible "AB" positions: $4096 - 1280 + 96 - 1 = 2911$.

\boxed{2911}

#Question#

Find the number of sequences of length 6 formed by the letters A, B, C, and D such that the sequence does not contain the consecutive substring "AB".

Question forwarded to both boundary models

Find the number of sequences of length 6 formed by the letters A, B, C, and D such that the sequence does not contain the consecutive substring "AB".

mistral-3-3b summary returned to questioner

$\boxed{2397}$

qwen3_5-27b summary returned to questioner

$\boxed{2911}$

Answer-key model response (used as ground truth)

To find the number of valid sequences of length 6, we can set up a recurrence relation. Let S_n be the number of valid sequences of length n that do not contain the consecutive substring "AB".

A valid sequence of length n can be formed by appending a letter to a valid sequence of length $n-1$:

- We can append 'A', 'C', or 'D' to any valid sequence of length $n-1$. This gives $3 S_{n-1}$ valid sequences.
- We can append 'B' to any valid sequence of length $n-1$ *unless* it ends in 'A'. The number of valid sequences of length $n-1$ that end in 'A' is exactly the number of valid sequences of length $n-2$ (since the last letter is fixed as 'A'). Therefore, we can append 'B' in $S_{n-1} - S_{n-2}$ ways.

Adding these together, the recurrence relation for the total number of valid sequences is:

$$S_n = 3 S_{n-1} + (S_{n-1} - S_{n-2}) = 4 S_{n-1} - S_{n-2}$$

We can find the base cases easily:

- For $n=0$: $S_0 = 1$ (the empty sequence)
- For $n=1$: $S_1 = 4$ (A, B, C, D)

Now we can use the recurrence relation to find S_6 :

- $S_2 = 4(4) - 1 = 15$
- $S_3 = 4(15) - 4 = 60 - 4 = 56$
- $S_4 = 4(56) - 15 = 224 - 15 = 209$
- $S_5 = 4(209) - 56 = 836 - 56 = 780$
- $S_6 = 4(780) - 209 = 3120 - 209 = 2911$

$\boxed{2911}$

E Additional Training Details

Loss and clipping. We optimize the policy with the CISPO importance-clipped surrogate [?], using asymmetric clip bounds $\epsilon_{\text{low}} = 0.75$ and $\epsilon_{\text{high}} = 3.0$. The KL coefficient is fixed at zero. The loss is normalized per-token across the batch.

Optimizer. We use AdamW [36] with learning rate $1e-6$, weight decay 0.01, and gradient norm clipped at 1.0. The learning rate uses a linear warmup of 60 steps with no decay thereafter.

Mixed-precision setup. Optimizer state is held in float32 with CPU offloading; the LM head is kept un-tied and in float32 separately from the model weights. Attention uses FlashAttention-3, with activation checkpointing on every transformer block. AdamW betas are $(\beta_1, \beta_2) = (0.9, 0.999)$ with $\varepsilon = 10^{-8}$.

Asynchronous rollouts. The trainer and inference pool run asynchronously with at most one trainer update fired before any in-flight rollout’s log-probabilities are re-scored against the current policy. Sequence lengths are capped at 24,576 tokens on the trainer side and 32,768 tokens on the inference side.

Sampling. Rollout sampling uses temperature 1.0 with the Qwen3.5-4B Instruct chat template (thinking disabled). The boundary models are queried at temperature 0.7, and the self-judge answer-key model is queried at temperature 0.0 on the final turn.

Token budgets. Training rollouts use 4 turns (3 probing rounds + 1 final question). The questioner is capped at 4,000 tokens during probing and the boundary models at 3,000. All other sampling parameters are left at the inference backend’s defaults.

Self-judge answer key. The prime-rl inference pool that hosts the policy doubles as the answer-key endpoint. On the final turn, the question is sent (with a stripped grading prompt) to that same vLLM session at `temperature = 0.0`.

Embedding-diversity bonus. Following [23], each generated question is encoded with a small sentence embedder (`bge-small-en-v1.5`), and the bonus is $\alpha \cdot (1 - \overline{\text{cos-sim}}_{k\text{-NN}})$, where $\overline{\text{cos-sim}}_{k\text{-NN}}$ is the mean cosine similarity to the $k = 5$ nearest neighbors in a sliding global buffer of the most recent 5,000 questions and $\alpha = 1.0$. We gate the diversity bonus on the calibration reward: the diversity term is multiplied by the (non-negative-clipped) calibration reward, so diversity is only rewarded when the question is calibrated. This prevents the policy from gaming the bonus by emitting novel but uncalibrated nonsense.

Seeds. Across the three seeds we vary both the rollout-example ordering and the inference-time sampling seed, so each seed differs in the order in which boundary pairs are presented and in the specific sequences sampled at each rollout step.

F Additional Ask-E Benchmark Implementation Details

Questioner-model configurations. Unless stated otherwise, we run questioner models at the following thinking effort: `low` for all Gemini models, `medium` for Claude and GPT models, and `high` for DeepSeek. For all API-based questioners (Gemini, DeepSeek, Claude [4], GPT [46]) we leave every sampling parameter at the provider’s default and only override the thinking level. Qwen questioners are run in `instruct (no-thinking)` mode with the sampling hyperparameters Qwen recommends for reasoning tasks in `instruct` mode: `temperature = 1.0`, `top_p = 0.95`, `top_k = 20`, `min_p = 0.0`, `presence_penalty = 1.5`, and `repetition_penalty = 1.0`.

Token budgets. Per-turn token budgets are 6,000 tokens for the questioner during probing and 10,000 for its final response; boundary models receive 4,000 tokens during probing and 6,000 on the final question.

Tag-based parsing. The text forwarded between roles is constructed by tag-based parsing, not by sending the raw model outputs verbatim:

- **Questioner → Boundary models.** Only the content following the questioner’s `#Question#` tag is forwarded; everything in `#Reasoning#` and `#Draft#` stays private. The forwarded question is truncated to 200 words to handle cases where the questioner keeps generating text after the question itself, which would otherwise eat into the boundary models’ token budget.

- **Boundary models** \rightarrow **Questioner**. The boundary model’s full response is reduced to a short summary before being shown to the questioner. We take everything after the model’s last `#Summary#` (or `#Output#`) tag, truncated to 2,000 characters, with the `\boxed{}` answer always preserved (re-prepended if it would otherwise fall outside the truncation window or sits above the `#Summary#` tag). If the model emits no `#Summary#/#Output#` tag at all, only the `\boxed{}` answer is forwarded; if the response contains neither, a placeholder string is forwarded indicating the model failed to produce a structured answer.

If the questioner fails to emit a parseable `#Question#` on the final turn (e.g. it runs out of tokens before reaching the tag), no boundary call is made and the session is bucketed as *too_hard*.

Scoring details. We use Gemini 3.1 Pro (`gemini-3.1-pro-preview`, `thinking_level=low`, `temperature = 0.7`) as the answer-key model. On the final turn, the questioner’s question is sent independently to the answer key, whose output is treated as ground truth.

Final answers are extracted from both boundary responses and the answer-key response with the same procedure: scan for `\boxed{...}` occurrences and return the contents of the last one. If no `\boxed` appears, return the last non-punctuation word of the response. Empirically this means responses that hit the max-token limit are usually marked incorrect, but we find this acceptable: models that are uncertain tend to repeat flawed reasoning rather than commit to a wrong final answer, so frequently running out of tokens on a given question is itself a strong signal that the model could not solve it.

Equivalence is checked with the `math-verify` library [29], preceded by a normalization pass that strips `\boxed`, `\text/\mathrm` wrappers, math delimiters, `\left/\right` pairs, and LaTeX spacing commands. If symbolic verification fails we fall through to whitespace- and case-insensitive string equality, comma-separated set equality (for reordered solution sets), and text-only fuzzy matching for “no solution” synonyms. A session is labelled *calibrated* when exactly one of the two boundary models matches the answer key, *too_easy* when both match, and *too_hard* when neither matches.

G Additional Downstream Math Benchmark Implementation Details

We evaluate the trained questioner (Qwen3.5-4B Instruct, no-thinking) and the un-trained Qwen3.5-4B Instruct baseline on the same downstream math benchmark suite, using identical sampling configurations.

Datasets. We evaluate on four public math reasoning benchmarks. All datasets are loaded from their respective HuggingFace hubs:

Dataset	n	HuggingFace ID
AIME (2022-2024)	90	AI-MO/aimo-validation-aime
AIME 2025	30	opencompass/AIME2025 (parts I + II)
HMMT 2025	30	MathArena/hmmt_feb_2025
IMO AnswerBench	400	Hwilner/imo-answerbench

Table 4: Downstream math benchmarks used to evaluate the trained questioner against the Qwen3.5-4B Instruct baseline. Counts are number of problems; AIME 2022–2024 is the AIMO validation pool of 90 problems, AIME 2025 is the union of parts I and II, HMMT 2025 is the February problem set, and IMO AnswerBench is the full release.

Sampling. Both the baseline and trained models are evaluated with the same sampling configuration to keep the comparison apples-to-apples. Specifically, we use the Qwen recommended hyperparameters for reasoning tasks in instruct mode (matching the questioner sampling described in Section 4):

- `temperature = 1.0`
- `top_p = 0.95`
- `top_k = 20`
- `min_p = 0.0`
- `presence_penalty = 1.5`
- `repetition_penalty = 1.0`
- Thinking disabled (instruct chat template, no `<think>` block)
- `max_tokens = 32,000`
- `n = 8` samples per problem

We do not use any system prompt and we do not provide few-shot examples; the problem is sent as a single user message with a short instruction appended that tells the model to reason step-by-step and emit its final answer in `\boxed{}`. The exact instruction depends on whether the dataset has integer answers:

- **AIME and AIME 2025** (integer answers in $[0, 999]$): *“Please solve this problem step by step. The answer is an integer from 0 to 999. Put your final answer in `\boxed{}`.”*
- **HMMT 2025 and IMO AnswerBench** (free-form answers): *“Please solve this problem step by step and put your final answer in `\boxed{}`.”*

The instruction is appended to the raw problem text and the combined string is the user-role content sent to the model’s chat template (with thinking disabled).

Metrics. For each dataset we report two metrics computed over the 8 samples per problem:

- **pass@8:** 1 if at least one of the 8 samples for that problem produces the correct final answer, 0 otherwise. Averaged over problems.
- **avg@8:** per-problem fraction of correct samples, averaged over problems (equivalently, the per-sample accuracy averaged across the dataset).

Answer extraction. We extract the final answer from each model response with the same procedure used elsewhere in the paper: we scan the response for `\boxed{...}` occurrences and return the contents of the last one. If no `\boxed{}` is present, we fall back to the last non-punctuation word. Responses that hit the 32,000-token cap without producing a `\boxed` answer are scored as incorrect (in practice, models that have to be cut off rarely commit to a final answer at all).

Equivalence checking. For AIME, AIME 2025, and HMMT 2025, we use rule-based equivalence matching (the `math-verify` [29] library combined with our LaTeX normalization pass) since their answers are integers or short closed-form expressions. For IMO AnswerBench, the ground-truth answers are frequently LaTeX expressions whose surface form varies substantially across valid restatements (e.g. $1+\sqrt{5}/2$ vs. $\frac{1+\sqrt{5}}{2}$), so rule-based matching under-counts correct answers. We therefore **re-grade IMO AnswerBench with an LLM judge**: each (problem, ground-truth, candidate) triple is sent to `gemin-3.1-flash-lite-preview` with a fixed yes/no prompt asking whether the student answer is mathematically equivalent to the ground truth, and the judge’s verdict is taken as correctness. The IMO AnswerBench numbers reported in Table 5 are the LLM-judge accuracies.

H Full Training Results Table

Dataset	Metric	n	Baseline	Mean ₃	seed 0	seed 1	seed 2
AIME (2022–2024)	pass@8	90	87.78%	90.00%	93.33%	86.67%	90.00%
AIME (2022–2024)	avg@8	90	68.06%	67.36%	67.08%	68.19%	66.81%
AIME 2025	pass@8	30	83.33%	83.33%	83.33%	83.33%	83.33%
AIME 2025	avg@8	30	54.17%	57.22%	58.33%	54.58%	58.75%
HMMT 2025	pass@8	30	70.00%	72.22%	73.33%	73.33%	70.00%
HMMT 2025	avg@8	30	45.00%	45.97%	45.83%	45.83%	46.25%
IMO AnswerBench	pass@8	400	70.25%	71.25%	72.00%	71.50%	70.25%
IMO AnswerBench	avg@8	400	43.91%	44.90%	45.28%	45.47%	43.94%

Table 5: Math-benchmark accuracy of Qwen3.5-4B Instruct accuracy before Ask-E training vs. accuracy after Ask-E RL training. Mean₃ is the mean across the 3 training seeds; **bold** marks values that beat the baseline. IMO AnswerBench is graded with the LLM judge (numbers are pass@8 and avg@8 over 8 samples per problem).