Loong: Synthesize Long Chain-of-Thoughts at Scale through Verifiers

Xingyue Huang[*], **Rishabh**[*], **Gregor Franke**[*], **Ziyi Yang**[*],
Jiamu Bai[†], **Weijie Bai**, **Jinhe Bi**, **Zifeng Ding**, **Yiqun Duan**, **Chengyu Fan**, **Wendong Fan**,
Xin Gao, **Ruohao Guo**, **Yuan He**, **Zhuangzhuang He**, **Xianglong Hu**, **Neil Johnson**,
Bowen Li, **Fangru Lin**, **Siyu Lin**, **Tong Liu**, **Yunpu Ma**, **Hao Shen**, **Hao Sun**,
Beibei Wang, **Fangyijie Wang**, **Hao Wang**, **Haoran Wang**, **Yang Wang**, **Yifeng Wang**,
Zhaowei Wang, **Ziyang Wang**, **Yifan Wu**, **Zikai Xiao**, **Chengxing Xie**, **Fan Yang**,
Junxiao Yang, **Qianshuo Ye**, **Ziyu Ye**, **Guangtao Zeng**, **Yuwen Ebony Zhang**,
Zeyu Zhang, **Zihao Zhu**, **Bernard Ghanem**, **Philip Torr**, **Guohao Li**[‡]

CAMEL-AI.org

Abstract

Recent advances in Large Language Models (LLMs) have shown that their reasoning capabilities can be significantly improved through Reinforcement Learning with Verifiable Reward (RLVR), particularly in domains like mathematics and programming, where ground-truth correctness can be automatically evaluated. However, extending this success to other reasoning-intensive domains remains challenging due to the scarcity of high-quality, verifiable datasets and the high cost of human supervision. In this work, we introduce the Loong Project: an open-source framework for scalable synthetic data generation and verification across a diverse range of reasoning-intensive domains. The framework consists of two key components: (1) LOONGBENCH, a curated seed dataset containing 8,729 human-vetted examples across 12 domains (e.g., Advanced Mathematics, Chemistry, Logic), each paired with executable code and rich metadata; and (2) LOONGENV, a modular synthetic data generation environment that supports multiple prompting strategies to produce new question-answer-code triples. Together, these components form an agent-environment loop that enables reinforcement learning, where an LLM-based agent is rewarded for generating Chain-of-Thought (CoT) solutions that align with code-executed answers. Empirically, we benchmark LOONGBENCH on a broad suite of both open-source and proprietary LLMs to evaluate domain coverage and reveal performance bottlenecks. In addition, we conduct a comprehensive analysis of synthetic data generated by LOONGENV, examining correctness, difficulty, and diversity. Code and documentation are available at `https://github.com/camel-ai/loong`.

1 Introduction

Recent Large Reasoning Models such as DeepSeek-R1 [1] and o3 [2] have demonstrated that the general reasoning capabilities of LLMs greatly improve when base models undergo post-training with Reinforcement Learning (RL) with a verifiable reward [3–5]. Mathematics and programming [3, 6]

[*]Equal contribution
[†]Authors listed here are in alphabetical order
[‡]Corresponding author

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Workshop on Bridging Language, Agent, and World Models for Reasoning and Planning.

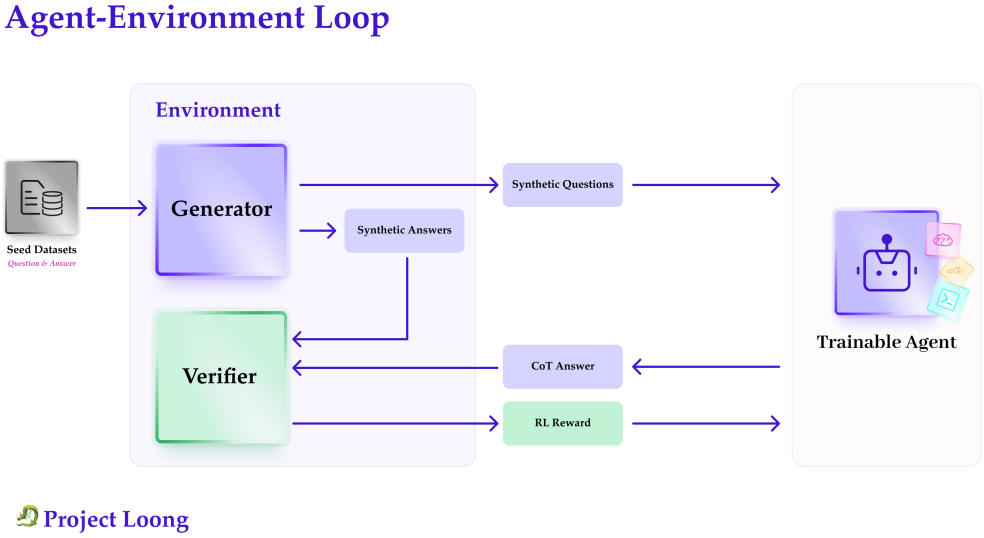


Figure 1: Agent-Environment Loop

have particularly benefited from this approach, as these domains can be verified quite easily, allowing accurate interpretation of LLM responses and effective comparison to the ground truth on a semantic level. This idea that the ease of verification is crucial to improving domain-specific capabilities has become widely accepted in the research community [7–9].

Another critical prerequisite which is often overlooked is the abundance of high-quality datasets, featuring questions paired with verified correct answers in the domains of Maths and Coding [10, 6]. These curated datasets provided the necessary signal for models to learn to construct coherent Chains-of-Thought (CoTs) [6], leading reliably to correct answers.

However, many other domains also require reliable reasoning, such as logic, graph theory, physics, and finance. These domains lack comparable datasets [11, 4, 12, 13], and human-supervised data production at scale is prohibitively expensive [5, 14, 13]. Without abundant correct answers to learn from, models cannot easily acquire domain-specific reasoning patterns. This raises a crucial question: *Can similar reasoning performance be achieved in domains beyond maths and programming?*

Approach. We present the Loong Project: an open framework for scaling synthetic data generation with verifiable supervision across a diverse set of reasoning-centric domains. The framework comprises two key components:

1. LOONGBENCH, a meticulously curated seed dataset comprising 8,729 examples across 12 reasoning-intensive domains, each accompanied by executable code and verified answers.
2. LOONGENV, a modular and versatile synthetic data generation environment, capable of generating diverse and semantically verifiable question-answer pairs using various automated generation strategies.

As illustrated in Figure 1, the overall agent-environment loop operates as follows: First, given a collection of seed datasets, our generator produces synthetic data points consisting of automatically generated questions and corresponding executable codes that answer these questions. Second, these codes are executed within the environment to yield synthetic answers. Third, a trainable agent is prompted to solve the synthetic questions by generating natural language CoT responses. Finally, a verifier compares the agent's CoT-derived answer to the code-generated answer. This setup will enable large-scale reinforcement learning with minimal human supervision while preserving semantic correctness through automated verification in the future.

Contributions. Our main contributions are:

- We introduce LOONGBENCH, a high-quality seed dataset of 8,729 examples spanning 12 reasoning-intensive domains, each paired with executable code and semantically verified answers.
- We develop LOONGENV, a synthetic data generation environment that supports multiple generation strategies to produce diverse and verifiable question-answer pairs.
- We benchmark LOONGBENCH across a diverse suite of large language models-including both open-source and proprietary, general-purpose and reasoning-specialized models-to establish baseline performance and identify domain-specific challenges.
- We conduct a detailed analysis of the synthetic data generated by LOONGENV, evaluating it in terms of semantic correctness, question difficulty, and diversity.

2 Loong Project

The Loong Project is focused on scaling up synthetic data generation with verification mechanisms across a broad spectrum of domains. We believe that generating synthetic data is essential not just to overcome the lack of datasets in under-represented fields, but also to strengthen reasoning abilities in areas like mathematics and programming by making more training examples available.

Our system relies on a multi-agent setup that starts with a seed dataset and expands it by generating new questions and corresponding answers. These synthetic questions are then passed to a model under training, which attempts to answer them. Verifiers are then employed to compare the model's answers with the pre-generated and pre-computed solutions, checking for semantic agreement.

At the heart of our approach is a simple idea: *an LLM equipped with a code interpreter is often far more reliable when solving complex tasks than one that relies solely on natural language reasoning*. This idea is backed by how many scientific disciplines operate in practice: whether it's physics, neurophysiology, economics, or computational biology, code-based solutions are a standard way to formulate and solve domain-specific problems.

To achieve our goals, two key components are required: a high-quality seed dataset that spans multiple domains, and a modular environment capable of generating synthetic questions and answers in a structured, verifiable manner.

LOONGBENCH provides the foundational data for our framework. It consists of 8,729 carefully curated examples covering 12 diverse, reasoning-intensive domains. Each example is annotated with executable code and paired with semantically verified answers. These seed examples ensure coverage of domain-specific patterns while maintaining correctness and diversity, offering a reliable basis for downstream synthetic data generation and benchmarking.

Complementing LOONGBENCH is LOONGENV, a flexible and extensible synthetic data generation environment. LOONGENV takes the seed examples from LOONGBENCH and uses various strategies, including few-shot prompting, self-instruction, and programmatic transformations, to generate new question-answer pairs. It also executes associated code to produce verifiable answers, which are used to supervise and evaluate model performance. This environment is domain-agnostic and modular, supporting plug-and-play verifiers and generation policies across reasoning domains.

2.1 LOONGBENCH: Human-Vetted Seed Datasets Across Multiple Domains

We begin by manually collecting domain-specific datasets consisting of questions and ground truth answers. Each question in the seed dataset is ensured to be solvable using code. If available, we also record the code that leads to the ground truth. The purpose of the seed dataset is not to be a large-scale dataset to use directly for training, but as a means to bootstrap the synthetic data generation process by seeding the generative process of the LLM.

To ensure broad coverage across diverse dataset types, we first collected 8,729 data points spanning 12 different domains to construct LOONGBENCH. Detailed statistics of the LOONGBENCH are provided in Table 1. In particular, every data point in the seed set contains:

- a *natural language question*,
- a *verified final answer*, and

Table 1: Statistics of the LOONGBENCH by domains.

Domain	Main Dependency	Size
Advanced Maths	`sympy`	1,611
Advanced Physics	`sympy, numpy`	429
Chemistry	`rdkit, numpy`	3,076
Computational Biology	-	51
Finance	`QuantLib`	235
Board Game	-	926
Graph & Discrete Maths	`networkx`	178
Logic	`python-constraint`	130
Mathematical Programming	`gurobipy, cvxpy, pyscipopt, statsmodel`	76
Medicine	`medcalc-bench`	916
Security & Safety	`cryptography, gmpy2, pycryptodome`	516
Programming	-	585

- an *accompanying rationale* in the form of executable python code.
- corresponding *metadata*, including license, source, domain, required dependencies, name, contributor, creation date, difficulty level, and relevant tags.

We introduce, in detail, how we collect the seed datapoints for each domain, and provide the construction details of LOONGBENCH in Appendix A.

2.2 LOONGENV: Modular Synthetic Generation with Verifiable Supervision

LOONGENV is a modular, black-box synthetic data generator seeded with a high-quality dataset, such as LOONGBENCH. Given this seed, it can generate an unbounded number of question-answer pairs that expand the training distribution in a controllable and verifiable manner. The generator is abstracted from downstream modules, enabling the flexible integration of various prompting strategies and agent behaviors.

We support both simple prompting methods and complex multi-agent generation workflows. This section details the generation process and experimental configurations.

Question Synthesis We explore three strategies for generating synthetic questions from seed examples:

- **Few-shot prompting** [15]: We provide a few seed QA pairs as demonstrations and prompt the model to generate new problems in a similar style. This serves as the simplest generation baseline.
- **Self-Instruct** [16]: An instruction-tuned model is recursively prompted to generate increasingly diverse and structured prompts.
- **Evol-Instruct** [17]: This approach evolves seed questions through mutation operations such as generalization, specification, and complexity scaling.

Answer Synthesis For every generated question, we generate a corresponding answer using a coder agent, which generates the corresponding code and tries to execute it to obtain the results. While we do not assume the correctness of these synthetic answers, code execution enables us to produce grounded numerical outputs in many domains. Notably, we do not use the raw synthetic answers directly for training.

Verifiers To ensure high-quality generated data, we incorporate a verification mechanism that filters out incorrect synthetic solutions generated by our pipeline. To do this effectively, we validate synthetic answers using two independent approaches:

Deriving one solution directly through the Generator's code execution. Independently generating another solution via natural-language Chain-of-Thought (CoT) reasoning. If these independent solutions agree, it's highly likely that the answer is correct. Although rare, there's still a possibility of

false positives (both approaches incorrectly agreeing). However, given the fundamentally different methods involved, we believe this will not occur often enough to be detrimental to model training.

Each environment also includes a verifier that semantically compares the LLM response with the synthetic answer, ensuring they are effectively equivalent. This verification step is crucial for accurately filtering semantic equivalences, significantly reducing false negatives (cases where semantically correct answers would otherwise be wrongly rejected).

Note that while our current setup relies on the *LLM-as-a-judge* paradigm, where large language models are used to assess the correctness of solutions, we ultimately aim to develop domain-specific verifiers (e.g., for mathematics or programming), which are both more efficient and more reliable.

Future direction In future work, we plan to use this verification framework to support reinforcement learning (RL). Specifically, the CoT-generating agent, the model we ultimately aim to train, can receive positive rewards only when its final answer is semantically verified to match the trusted synthetic answer. This setup enables a Reinforcement Learning from Verified Rewards (RLVR) paradigm, where the agent learns exclusively from high-confidence, semantically aligned supervision.

3 Experiments

We evaluate the capabilities of state-of-the-art language models on LOONGBENCH and synthetic datasets produced by LOONGENV. Specifically, we aim to answer the following questions:

- How well models perform across diverse reasoning domains on LOONGBENCH?
- How reliably can they generate executable and semantically valid solutions using LOONGENV?
- How do different prompting strategies affect the diversity and difficulty of the generated tasks?

3.1 Setup

Dataset and Models. We evaluate the reasoning and problem-solving capabilities of a range of state-of-the-art language models across 12 domains provided as LOONGBENCH: **Advanced Math**, **Advanced Physics**, **Chemistry**, **Computational Biology**, **Finance**, **Game**, **Graph & Discrete Math**, **Logic**, **Mathematical Programming**, **Medicine**, **Security & Safety**, and **Programming**. Each domain consists of curated problems sourced from high-quality benchmarks, academic competitions, and domain-specific datasets, as described in Section 2.1. We include both open- and closed-source language models in our evaluation. Proprietary models such as `GPT4.1-mini` [18], `o3-mini` [2], `Grok-3` [19], and `Claude-3.7-Sonnet` [20] provide strong baselines from leading commercial providers. In parallel, we incorporate high-performing open-source models like `DeepSeek-r1` [1] and `Qwen3-8B` [21], both of which demonstrate competitive performance on reasoning-intensive tasks.

Implementation. All models are evaluated using their publicly available APIs or checkpoints, with consistent temperature, top-k, and top-p settings where applicable. For fairness, we disable function/tool calling unless stated otherwise, and restrict outputs to a single response per prompt without retries. We set the max-token for all experiments to 4096[4]. We use a single NVIDIA H100 80GB GPU for inference with the open-sourced model. Our development is based on the CAMEL framework [22].

Evaluation. Accuracy is measured as the percentage of correctly answered problems per domain. We employ LLM-as-judge [23], using a `GPT4.1-mini` [18], to assess correctness due to the complex nature of answers, which may vary in format across different domains. The judge accounts for symbolic equivalence where applicable, ensuring that mathematically correct but differently expressed answers are not penalized.

[4]Some questions in **mathematical programming** thus go out of this window and the result would be incomplete.

Table 2: Benchmarking accuracy across domains and model categories. The best model is highlighted in **bold**, and the second best is underlined.

Domain	GPT4.1-mini	o3-mini	Grok-3	Claude-3.7	DeepSeek-r1	Qwen3-8B
Advanced Maths	91.4	**97.4**	92.3	79.3	96.7	79.2
Advance Physics	71.8	75.3	69.0	63.9	**77.4**	59.2
Chemistry	75.2	79.5	71.2	**80.7**	74.7	79.7
Computational Biology	90.2	88.2	**96.1**	90.2	88.2	86.2
Finance	23.8	**24.3**	19.1	22.0	**24.3**	12.8
Game	92.0	96.0	93.0	95.1	**97.3**	43.2
Graph	80.9	82.0	80.1	73.6	**83.7**	62.9
Logic	**65.4**	61.6	55.4	46.9	62.3	39.2
Math. Programming	11.8	9.2	6.4	**13.2**	10.5	10.0
Medicine	**59.6**	46.3	50.7	54.1	52.6	28.4
Security	25.6	11.2	22.3	4.7	**28.7**	7.9
Programming	98.6	**100.0**	91.5	97.4	98.8	81.7

3.2 Benchmarking LOONGBENCH

We report the benchmarking results in Table 2. The results reveal key trends regarding domain difficulty, model specialization, and the performance gap between open- and closed-source systems. We highlight three central findings below.

A well-calibrated spectrum of difficulty. The twelve domains in our benchmark exhibit a wide range of difficulty levels. For example, the *Mathematical Programming* domain yields accuracies as low as 10%, indicating substantial unresolved complexity, whereas the *Programming* domain is nearly saturated, with models like `o3-mini` achieving 100% accuracy. Other domains, such as *Logic*, *Graph & Discrete Math*, and *Chemistry*, distribute evenly across the middle range. This balance ensures the benchmark is broadly discriminative: it avoids ceiling effects for strong models while remaining accessible for lower-capacity systems, making it a robust testbed for both evaluation and ablation studies.

Reasoning-optimized models consistently outperform. We observe that models explicitly tuned or pretrained for reasoning, particularly `o3-mini` and `DeepSeek-r1`, achieve top scores across the majority of domains. Notably, `DeepSeek-r1` consistently reached top-2 in 8 out of 12 datasets, whereas `o3-mini` also reached 6 out of 12. These results suggest that the benchmark requires more than surface-level pattern matching or factual recall: it emphasizes structured, multi-step reasoning, which is better captured by models with strong CoT or planning capabilities.

Open-source models lag in reasoning-heavy domains. Although open models like `DeepSeek-r1` and `Qwen3-8B` perform competitively in certain areas, a clear performance gap remains in the most reasoning-intensive domains. For instance, in the *Game* and *Logic* domains, `Qwen3-8B` trails `o3-mini` by 50 and 22 percentage points, respectively. These discrepancies highlight two key challenges: (i) current open-source systems underperform on strategy-based and logic-heavy tasks, and (ii) this benchmark suite is well-positioned to reveal such fine-grained capability gaps, offering concrete targets for future model development and alignment in the open-source community.

3.3 Synthesizing Data with LOONGENV

We use LOONGENV to generate synthetic data under three instruction paradigms: **Few-shot prompting** [15], **Self-instruct** [16], and **Evol-instruct** [17]. For each setting, we initialize the generator with seed examples from LOONGBENCH and use a multi-agent workflow to synthesize new data:

1. A **question synthesis agent** generates a new natural language question based on seed data.
2. A separate **code generation agent** is then prompted to produce an executable program that answers the generated question.

We then evaluate the quality of the generated samples in two stages:

- **Executability Check:** We run each generated code snippet in a sandboxed Python environment and measure the fraction that executes without error. This provides a proxy for functional correctness and yields the *execution success rate*.
- **Verification via Judge Agent:** For each generated question-code pair, we prompt a **judge agent** to assess two criteria: (1) whether the question is well-formed and meaningful, and (2) whether the generated code correctly solves the posed problem. The fraction of samples that fail either criterion determines the *rejection rate*.

This two-stage evaluation process provides a systematic measure of both functional correctness and semantic fidelity of the generated question-code pairs.

Implementation. We use `GPT-4.1-mini` [18] as both the question synthesis agent and the code generation agent across all experiments. For each domain and each generation strategies, we randomly selected from the seed dataset and generated 100 synthetic questions. For verification, we employ `DeepSeek-R1` [1] as the judge agent. All other settings remain consistent with the experimental setup described in the previous section.

3.3.1 Execution and Verification Outcomes

Figure 2 summarizes the execution outcomes of synthetic data generated by LOONGENV under three prompting strategies-FewShot, Self-Instruct, and Evol-Instruct-across two domains: *Logic* and *Physics*. We categorize outcomes into three classes: *Pass* (code executed and the judge approved the answer), *Judge-Rejected* (code executed but result disagreed with the judge's answer), and *Not Executable* (code failed to run).

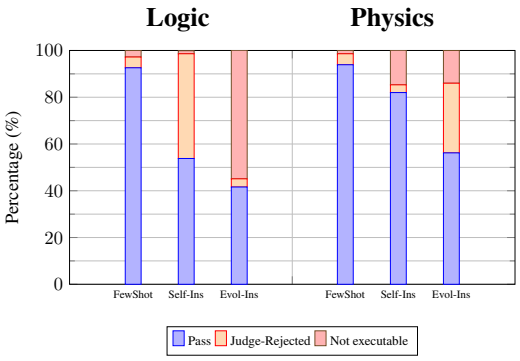


Figure 2: Execution outcome breakdown across synthetic data generation strategies for Logic and Physics domains.

We observe that in the *Logic* domain, FewShot prompting yields a high pass rate (92.6%) with very few failures, while Self-Instruct exhibits a much higher rejection rate (44.8%) and Evol-Instruct produces a large portion of non-executable code (55.0%). In contrast, for the *Physics* domain, both FewShot and Self-Instruct maintain high pass rates (93.9% and 82.0% respectively), but Evol-Instruct again suffers from significantly reduced executability and semantic agreement, with 29.8% judged as incorrect and 14.0% failing to execute.

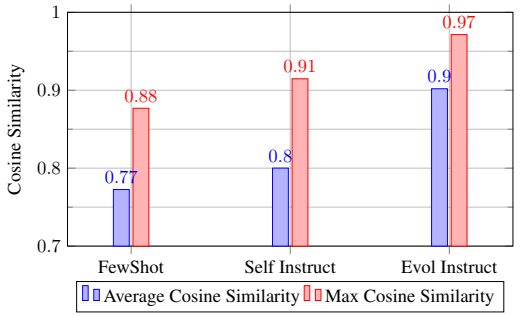
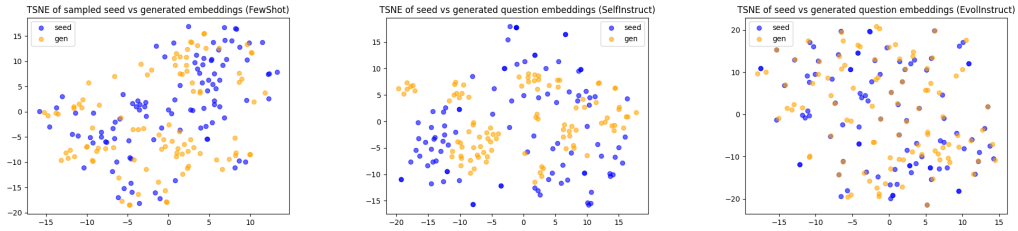


Figure 3: Seed-generated pairwise cosine similarity on the Advanced Physics domain across synthetic data generation strategies.

These results highlight a trade-off between prompting complexity and generation reliability: while FewShot prompting offers the most stable pipeline with the highest overall pass rates, Evol-Instruct, despite its higher rejection and execution failure rates, remains highly valuable from a training perspective. Its ability to synthesize more diverse and challenging reasoning tasks makes it especially well-suited for building robust models. As we demonstrate in the later sections, Evol-Instruct more effectively captures edge cases and reasoning depth that are essential for meaningful generalization.

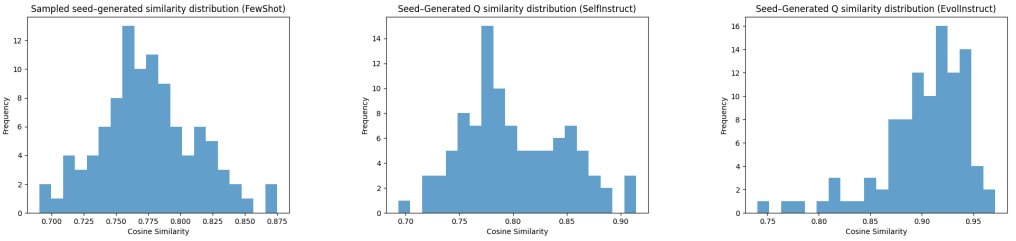
3.3.2 Diversity

We assess the semantic diversity of generated questions in the Advanced Physics domain by comparing their embeddings to those of the seed questions using cosine similarity and t-SNE visualization. Figure 3 reports the average and maximum cosine similarity between 100 seed-generated question



(a) Few-shot (b) Self-Instruct (c) Evol-Instruct

Figure 4: t-SNE projection of embedding space for seed vs. generated problems on the Advanced Physics across different generation strategies. Generated examples (orange) and seed samples (blue) cluster with varying degrees of overlap, indicating distributional proximity and diversity.



(a) Few-shot (b) Self-Instruct (c) Evol-Instruct

Figure 5: Cosine similarity distribution between seed and generated questions on the Advanced Physics domain for different generation strategies.

pairs under each strategy, while Figures 4 and 5 provide a qualitative breakdown of embedding proximity using t-SNE and distributional skew.

Overall, we observe that **Few-Shot** prompting tends to produce questions that are more lexically distinct from the seeds, as reflected by its lower average similarity (0.77) and more dispersed t-SNE distribution. However, these questions often remain close to the original in structure and complexity, representing surface-level variation.

In contrast, **Self-Instruct** generates questions that are both lexically and semantically more diverse, often drifting farther from the seed set in the t-SNE space. This suggests a bias toward novelty, albeit sometimes at the expense of coherence or executability (see correctness analysis).

Interestingly, **Evol-Instruct** generates questions that appear much closer to the seed questions semantically, as suggested by the high average (0.90+) and maximum cosine similarities, along with tight clustering in the t-SNE plots. This pattern may indicate that Evol-Instruct tends to apply transformations, such as generalization, specification, or rephrasing, that preserve the underlying semantics while increasing complexity.

These observations suggest a potential trade-off between surface-level lexical variation and semantic alignment: while Few-Shot prompting yields more lexically distinct outputs, Evol-Instruct may generate structurally richer and semantically coherent examples. We hypothesize that such properties could be beneficial for capturing deeper reasoning patterns and edge cases, which are valuable for robust model training.

3.3.3 Difficulty

Finally, we proceed to measure the difficulty level of the generated datasets. Here, we again focus on *Advanced Physics* domain and assess the difficulty of generated questions by measuring the accuracy of two models: `GPT4.1-mini` and `DeepSeek-r1`.

Table 3 presents model accuracies on questions generated via each strategy in the Advanced Physics domain. We observe that both `GPT4.1-mini` and `DeepSeek-r1` perform best on Few-Shot generated

Table 3: Average accuracy (%) on Advanced Physics domain across synthetic data generation strategies

Model	Few-shot	Self-Instruct	Evol-Instruct	Seed Dataset
`GPT4.1-mini`	92.0 ↑	83.0 ↑	62.0 ↓	71.8
`DeepSeek-r1`	93.2 ↑	87.4 ↑	70.3 ↓	77.4

data, achieving 92.0% and 93.2% accuracy respectively. Accuracy slightly drops on Self-Instruct data (83.0% and 87.4%), and significantly declines on Evol-Instruct questions (62.0% and 70.3%).

Interestingly, despite the Evol-Instruct examples being more semantically similar to the seed questions (Figure 3), their lower model accuracy suggests that they are substantially harder to solve. This supports our earlier hypothesis that Evol-Instruct tends to preserve core semantics while increasing reasoning complexity, possibly through abstract transformations or compound formulations.

4 Related Work

Post-training for LLM Early efforts in aligning large language models with human preferences focused on fine-tuning via human feedback. InstructGPT demonstrated that reinforcement learning from human feedback (RLHF) can make smaller models more helpful, honest, and harmless than much larger base models [24], building on earlier work that optimized from pairwise preferences [25] and fine-tuned models to match human judgments [26]. Recent approaches pursue more stable or efficient alignment techniques. Direct Preference Optimization (DPO) reframes reward modeling as a supervised objective [27], while Preference Ranking Optimization (PRO) models task-aware ranks [28] and RRHF bypasses RL entirely [29]. Other efforts include training helpful assistants via RLHF [24], improving summarization through feedback [30], and formalizing verifiable reward learning [31]. Co-evolutionary training with unit testers [32], single-example reward tuning [33], autoregressive search [3], cross-domain RLVR [34], and prolonged fine-tuning [35] further extend the design space.

Synthetic data generation In recent years, there has been a surge of interest in using language models to synthesize their own training data. Self-Instruct introduced a pipeline in which models generate and train on synthetic instructions to improve alignment [36]. WizardLM [17] and WizardCoder [37] extend this with Evol-Instruct, automatically evolving instruction complexity for general and code tasks, respectively. The Flan Collection [38] curates diverse instructional tasks and data augmentation strategies to boost zero-shot generalization, while LIMA [39] shows that only a small amount of high-quality prompts can yield strong performance. Super-NaturalInstructions [40] aggregates over 1600 NLP tasks to study generalization to unseen instructions. Beyond instruction generation, more systematic data generation frameworks have emerged. DataGen provides a unified pipeline with modules for controllability, diversity, and factuality [41]. A comprehensive survey reviews challenges and advances in LLM-based synthetic data generation for text and code [42].

Reinforcement Learning with Verifiable Reward (RLVR) RLVR combines reinforcement learning with automatic, verifiable reward signals, often programmatic correctness or agreement with auxiliary tools, to scale alignment and reasoning. GRPO formalizes reward dynamics [31], while one-shot RLVR shows that even a single example can substantially improve reasoning [33]. Satori proposes a Chain-of-Action-Thought framework [3], and ProRL shows that prolonged optimization enables emergent capabilities [35]. RLVR has also been applied to instruction following [34], co-evolutionary coding setups [32], and multi-domain generalization. These efforts enable scalable, automatic reward supervision in open-ended environments [35, 34, 32, 3, 33, 31].

5 Conclusion

We present Loong, a modular framework for aligning LLMs via synthetic data generation and verifiable reward supervision. Our approach is driven by two key insights: effective alignment requires diverse, domain-specific data and scalable, annotation-free reward mechanisms.

Our contributions are fourfold: (1) **LOONGBENCH**, a seed dataset of 8,729 examples across 12 reasoning-intensive domains with executable code and verified answers; (2) **LOONGENV**, a flexible environment enabling diverse synthetic data generation strategies; (3) comprehensive benchmarking of open-source and proprietary models to assess domain generalization; and (4) in-depth analysis of generated data quality in terms of correctness, diversity, and complexity.

Together, these components form a cohesive framework for studying alignment at scale. Our results demonstrate that structured synthetic generation and verification can yield diverse and challenging synthetic datasets. A key future direction is leveraging LOONGENV to support RLVR with synthetically generated questions. We also plan to extend LOONGENV with tool-augmented generation and formal abstraction, and scale LOONGBENCH to cover multilingual and multimodal tasks.

References

[1] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. URL `https://arxiv.org/abs/2501.12948`.

[2] OpenAI. o3-mini large language model. `https://openai.com/index/openai-o3-mini/`, 2025. Accessed: 2025-05-15.

[3] Maohao Shen, Guangtao Zeng, Zhenting Qi, et al. Satori: Reinforcement learning with chain-of-action-thought enhances llm reasoning via autoregressive search. *arXiv preprint arXiv:2502.02508*, 2025.

[4] Miao Peng, Nuo Chen, Zongrui Suo, and Jia Li. Rewarding graph reasoning process makes llms more generalized reasoners, 2025.

[5] Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. Reasoning gym: Reasoning environments for reinforcement learning with verifiable rewards, 2025.

[6] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2022. URL `https://arxiv.org/abs/2201.11903`. arXiv preprint.

[7] Sebastian Raschka. The state of reinforcement learning for llm reasoning. blog post, May 2025, 2025.

[8] Anonymous. Automatic post-training via reinforcement learning with verifiable rewards, 2025.

[9] Xueguang Ma, Qian Liu, Dongfu Jiang, Ge Zhang, Zejun Ma, and Wenhu Chen. General-reasoner: Advancing llm reasoning across all domains, 2025.

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

[11] Junteng Liu, Yuanxiang Fan, Zhuo Jiang, Han Ding, Yongyi Hu, Chi Zhang, Yiqi Shi, Shitong Weng, Aili Chen, Shiqi Chen, Yunan Huang, Mozhi Zhang, Pengyu Zhao, Junjie Yan, and Junxian He. Synlogic: Synthesizing verifiable reasoning data at scale for learning logical reasoning, 2025.

[12] Zijian Wu, Jinjie Ni, Xiangyan Liu, Zichen Liu, Hang Yan, and Michael Qizhe Shieh. Synthrl: Scaling visual reasoning with verifiable data synthesis, 2025.

[13] Zhuohan Xie, Dhruv Sahnan, Debopriyo Banerjee, Georgi Georgiev, Rushil Thareja, Hachem Madmoun, Jinyan Su, Aaryamonvikram Singh, Yuxia Wang, Rui Xing, Fajri Koto, Haonan Li, Ivan Koychev, Tanmoy Chakraborty, Salem Lahlou, Veselin Stoyanov, and Preslav Nakov. Finchain: A symbolic benchmark for verifiable chain-of-thought financial reasoning, 2025.

[14] Zhaowei Liu, Xin Guo, Fangqi Lou, Lingfeng Zeng, Jinyi Niu, Zixuan Wang, Jiajie Xu, Weige Cai, Ziwei Yang, Xueqian Zhao, Chao Li, Sheng Xu, Dezhi Chen, Yun Chen, Zuo Bai, and Liwen Zhang. Fin-r1: A large language model for financial reasoning through reinforcement learning, 2025.

[15] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901, 2020. URL https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[16] 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. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13484–13508, Toronto, Canada, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.754. URL https://aclanthology.org/2023.acl-long.754/.

[17] Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*, 2023. URL https://arxiv.org/abs/2304.12244.

[18] OpenAI. Introducing gpt-4.1 in the api. https://openai.com/index/gpt-4-1/, 2025. Accessed: 2025-05-22.

[19] xAI. Grok 3 beta — the age of reasoning agents, 2025. URL https://x.ai/news/grok-3. Accessed: 2025-05-22.

[20] Anthropic. Claude 3.7 sonnet: Hybrid reasoning model announcement, February 2025. URL https://www.anthropic.com/news/claude-3-7-sonnet. Accessed: 2025-05-22.

[21] 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. *arXiv preprint arXiv:2505.09388*, 2025.

[22] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large language model society. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[23] Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni, and Jian Guo. A survey on llm-as-a-judge, 2025. URL https://arxiv.org/abs/2411.15594.

[24] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:27730–27744, 2022.

[25] Paul F. Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 4299–4307, 2017.

[26] Daniel M. Ziegler, Nisan Stiennon, Jeff Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *CoRR*, abs/1909.08593, 2019.

[27] Rafael Rafailov et al. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*, 2023.

[28] Feifan Song, Bowen Yu, Minghao Li, Haiyang Yu, Fei Huang, Yongbin Li, and Houfeng Wang. Preference ranking optimization for human alignment. *arXiv preprint arXiv:2306.17492*, 2023.

[29] Zheng Yuan, Hongyi Yuan, Chuanqi Tan, Wei Wang, Songfang Huang, and Fei Huang. Rrhf: Rank responses to align language models with human feedback without tears. *arXiv preprint arXiv:2304.05302*, 2023.

[30] Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel Ziegler, Ryan Lowe, et al. Learning to summarize from human feedback. *Advances in Neural Information Processing Systems (NeurIPS)*, 33: 3008–3021, 2020.

[31] Youssef Mroueh. Reinforcement learning with verifiable rewards: Grpo's effective loss, dynamics, and success amplification. *arXiv preprint arXiv:2503.06639*, 2025.

[32] Yinjie Wang, Ling Yang, Ye Tian, Ke Shen, and Mengdi Wang. Co-evolving llm coder and unit tester via reinforcement learning. *arXiv preprint arXiv:2506.03136*, 2025.

[33] Yiping Wang, Qing Yang, Zhiyuan Zeng, et al. Reinforcement learning for reasoning in large language models with one training example. *arXiv preprint arXiv:2504.20571*, 2025.

[34] Yi Su, Dian Yu, Linfeng Song, et al. Expanding rl with verifiable rewards across diverse domains. *arXiv preprint arXiv:2503.23829*, 2025.

[35] Mingjie Liu, Shizhe Diao, Ximing Lu, et al. Prorl: Prolonged reinforcement learning expands reasoning boundaries in large language models. *arXiv preprint arXiv:2505.24864*, 2025.

[36] 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. *arXiv preprint arXiv:2212.10560*, 2022.

[37] 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.

[38] Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V. Le, Barret Zoph, Jason Wei, and Adam Roberts. The flan collection: Designing data and methods for effective instruction tuning. *arXiv preprint arXiv:2301.13688*, 2023.

[39] Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. Lima: Less is more for alignment. *arXiv preprint arXiv:2305.11206*, 2023.

[40] Yizhong Wang, Swaroop Mishra, et al. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. In *EMNLP*, 2022.

[41] Yue Huang, Siyuan Wu, Chujie Gao, et al. Datagen: Unified synthetic dataset generation via large language models. In *International Conference on Learning Representations (ICLR)*, 2025.

[42] Mihai Nadas, Laura Diosan, and Andreea Tomescu. Synthetic data generation using large language models: Advances in text and code. *arXiv preprint arXiv:2503.14023*, 2025.

[43] Xiaoxuan Wang, Ziniu Hu, Pan Lu, Yanqiao Zhu, Jieyu Zhang, Satyen Subramaniam, Arjun R. Loomba, Shichang Zhang, Yizhou Sun, and Wei Wang. SciBench: Evaluating College-Level Scientific Problem-Solving Abilities of Large Language Models. In *Proceedings of the Forty-First International Conference on Machine Learning*, 2024.

[44] Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems, 2024.

[45] OpenAI. Gpt-4o: Openai's new multimodal flagship model. https://openai.com/index/hello-gpt-4o/, 2024. Accessed: 2025-05-22.

[46] David Duarte. Quantlib-python object building documentation. https://quantlib-python-docs.readthedocs.io/, 2020. Accessed: 2025-05-22.

[47] Wenqi Zhang, Ke Tang, Hai Wu, Mengna Wang, Yongliang Shen, Guiyang Hou, Zeqi Tan, Peng Li, Yueting Zhuang, and Weiming Lu. Agent-pro: Learning to evolve via policy-level reflection and optimization. *arXiv preprint arXiv:2402.17574*, 2024.

[48] Daochen Zha, Kwei-Herng Lai, Songyi Huang, Yuanpu Cao, Keerthana Reddy, Juan Vargas, Alex Nguyen, Ruzhe Wei, Junyu Guo, and Xia Hu. Rlcard: A platform for reinforcement learning in card games. In *IJCAI*, 2020.

[49] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[50] Da Zha, Jun Zhang, Chao Zhou, Zhen Xu, and Xia Hu. Douzero: Mastering doudizhu with self-play deep reinforcement learning. In *Proceedings of the 38th International Conference on Machine Learning*, pages 12333–12343. PMLR, 2021.

[51] Jérôme Arjonilla, Abdallah Saffidine, and Tristan Cazenave. Perfect information monte carlo with postponing reasoning. In *2024 IEEE Conference on Games (CoG)*, page 1–8. IEEE, August 2024. doi: 10.1109/cog60054.2024.10645574. URL http://dx.doi.org/10.1109/CoG60054.2024.10645574.

[52] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[53] Wanjun Zhong, Siyuan Wang, Duyu Tang, Zenan Xu, Daya Guo, Jiahai Wang, Jian Yin, Ming Zhou, and Nan Duan. Ar-lsat: Investigating analytical reasoning of text. *arXiv preprint arXiv:2104.06598*, 2021.

[54] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL https://www.gurobi.com.

[55] Stephen Maher, Matthias Miltenberger, João Pedro Pedroso, Daniel Rehfeldt, Robert Schwarz, and Felipe Serrano. PySCIPOpt: Mathematical programming in python with the SCIP optimization suite. In *Mathematical Software – ICMS 2016*, pages 301–307. Springer International Publishing, 2016. doi: 10.1007/978-3-319-42432-3_37.

[56] Steven Diamond, Eric Chu, and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization, version 0.2. http://cvxpy.org/, May 2014.

[57] Skipper Seabold and Josef Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

[58] Nikhil Khandekar, Qiao Jin, Guangzhi Xiong, Soren Dunn, Serina S Applebaum, Zain Anwar, Maame Sarfo-Gyamfi, Conrad W Safranek, Abid A Anwar, Andrew Zhang, Aidan Gilson, Maxwell B Singer, Amisha Dave, Andrew Taylor, Aidong Zhang, Qingyu Chen, and Zhiyong Lu. Medcalc-bench: Evaluating large language models for medical calculations, 2024. URL https://arxiv.org/abs/2406.12036.

[59] Yunhui Xia, Wei Shen, Yan Wang, Jason Klein Liu, Huifeng Sun, Siyue Wu, Jian Hu, and Xiaolong Xu. Leetcodedataset: A temporal dataset for robust evaluation and efficient training of code llms, 2025. URL https://arxiv.org/abs/2504.14655.

[60] Anthropic. Introducing claude 3.5 sonnet, June 2024. URL https://www.anthropic.com/news/claude-3-5-sonnet. Accessed: 2025-06-04.

A Details of LOONGBENCH

Advanced Math To construct a high-quality seed dataset for advanced mathematical problem-solving, we begin by selecting problems from the training split of the MATH [10] dataset, focusing on those labeled with difficulty levels 4 or 5. For each selected problem, we prompt the `o3-mini` model [2] to generate corresponding `SymPy` code intended to solve it. We then filter out any outputs that fail to produce executable code or yield incorrect solutions. The remaining `SymPy` programs are executed to produce numerical or symbolic results, which are then compared against the ground truth answers using MathVerifier[5]–a tool designed to parse and evaluate LaTeX-formatted mathematical expressions. Only those samples with verified correct solutions are retained and added to the seed dataset. We ended up collecting 1,611 problems.

Advanced Physics We select the physics problems from Scibench [43] and Olympiadbench [44]. We asked `o3-mini` [2] to generate `sympy` code which solves the problem. We execute the code and compare the result with the ground truth answer. The answer has two parts: numerical value and unit. Unit conversion and dynamic tolerance are used in the verification process. unit conversion: If the response unit doesn't match with the ground truth unit, perform unit conversion to match them (e.g. 1km v.s. 1000m, $200,000$ vs $2 * 10^5$). We then compare the converted numerical value and unit to the ground truth ones. dynamic tolerance: Adjust relative tolerance when comparing the numerical results if needed (the default tolerance is 0.01, but sometimes the ground truth answer is given in lower tolerance, e.g. when ground truth is $1.3e+02$, we should allow the answers to sit within a difference of $0.05e+02$) For each problem, we perform a maximum of three attempts to generate the correct rationale. The correct ones are included in the seed dataset.

Chemistry We extracted examples from the ChemistryQA dataset available on HuggingFace [6], which contains a wide range of chemistry reasoning question-answer pairs. We selected examples from this dataset that present conceptually clear and computationally relevant problems. When the original examples provided concise and understandable questions with well-formatted answers, we retained them directly. For examples where the format was unclear or the answer presentation lacked structure, we reformulated them using the `o3-mini` [2] model. This model allowed us to generate well-structured questions that emphasize general chemistry reasoning, such as stoichiometry, thermodynamics, or molecular structure, while avoiding references to implementation-specific details. For instance, an original question about molar mass calculation was rewritten as: *"What is the molar mass of sulfuric acid (H_2SO_4)?"* This process resulted in 3076 seed data points, each aligned with core chemistry concepts and designed to support both human and model-based reasoning.

Computational Biology We collected datasets from the Biochemistry, Genetics, Molecular Biology, and Microbiology domains under the Biology category on the General Reasoning website[7]. These datasets were then preprocessed using a custom script to merge and transform the raw data into a unified question-answering format.

We employed `GPT-4o` [45] to generate code-based solutions for each question. Subsequently, we filtered out instances where the generated code could not be executed or failed to address the problem correctly. To ensure the quality and validity of the resulting dataset, we further utilized `GPT-4o-mini` [45] to evaluate whether: (1) the generated code genuinely solved the problem rather than merely paraphrasing it; (2) the execution result of the code was semantically consistent with the ground truth answer; (3) the code included necessary computational steps instead of directly hardcoding the output. To more accurately evaluate whether the results of code execution align with the ground truth, we employed rule-based regular expression matching for both. Finally, we retained the question-answer data that showed consistent matches. Only samples that passed all these criteria were included in the final seed dataset used for our experiments. We ended up collecting 51 samples.

Finance We focus on extracting examples from a variety of QuantLib [46] resources, including the official documentation[8], online tutorials[9], and the FinAI Financial Reasoning Dataset on Hugging

[5] `https://github.com/camel-ai/camel/blob/master/camel/verifiers/math_verifier.py`

[6] `https://huggingface.co/datasets/avaliev/ChemistryQA`

[7] `https://gr.inc`

[8] `quantlib-python-docs.readthedocs.io`

[9] `http://gouthamanbalaraman.com/blog/quantlib-python-tutorials-with-examples.html`

Face[10]. From these, we identified instances that contain well-defined financial modeling problems and accompanying code examples. When the original examples included clearly presented questions and answers, we retained them directly. For those that lacked clarity or structure, we used the `o3-mini` model [2] to reformulate them into precise question formats that emphasize financial reasoning and conceptual understanding, while avoiding low-level implementation details. For instance, an example involving bond pricing using QuantLib was rewritten into the question: *"What is the clean price of a fixed-rate bond with a face value of $1000, a 5% annual coupon, and 10 years to maturity if the yield is 4.5%?"* This process yielded a curated set of seed data points suitable for financial reasoning tasks, each paired with a corresponding Python-based solution.

Board Game We adopt *Blackjack* as the canonical imperfect-information board game and generate interaction data through a hybrid pipeline that combines reproducible simulation with quality-controlled traces derived from *Agent Pro* [47]. All fresh episodes are played in the RLCARD environment [48] against two fixed baseline opponents–Deep Q-Network (DQN) [49] and Deep Monte-Carlo (DMC) [50]–so that the strategic background remains constant. We execute games until we collect 50 distinct *losing* rounds; these serve as the seed corpus for policy-level reflection. Following the protocol of Zhang et al. [47], every state-action pair is augmented with (i) a natural-language rationale extracted from an expert Blackjack playbook and (ii) a behaviour guideline produced by the reflection mechanism.

To suppress hallucination and retain only strategically sound trajectories, we run perfect-information Monte-Carlo roll-outs [51] from each decision point to estimate the true probability of winning for every legal action. A trajectory is kept only if the chosen action belongs to the top-k actions ranked by this oracle probability (with $k{=}1$ in all experiments). This filtering step aligns the sampled behaviour with optimal-play statistics and yields a high-fidelity corpus for subsequent safety and alignment analysis. We ended up generating 926 questions as described.

Graph & Discrete Math We focus on extracting examples from the documentation of `networkx`[11] [52] (abbreviated as `nx`), a widely used library for network science. We began by inspecting all 896 functions under `nx.algorithms`. From these, we identified 370 functions that included usage examples and extracted the corresponding code snippets as rationales. To ensure reproducibility, we filtered out examples involving multiple valid outputs or random sampling. The remaining examples were rewritten using `GPT-4o-mini` [45] to formulate questions that reflect the exact outcome of the code in general graph-theoretic terms, avoiding references to implementation details. For instance, the code snippet `nx.is_k_regular(nx.Graph([(1, 2), (2, 3), (3, 4), (4, 1)]), k=3)` was transformed into the question: *"For a graph defined by the edges connecting nodes as follows: (1, 2), (2, 3), (3, 4), and (4, 1), is the graph 3-regular?"* This process yielded 178 seed data points, each corresponding to a distinct algorithm implementation in the `networkx` library. All data points were then manually checked to ensure that the code executes correctly and produces the intended result.

Logic We currently have two types of constraint satisfaction problems (CSPs): Einstein's Puzzle[12] and Analytical Reasoning questions from the Law School Admission Test (AR-LSAT)[13].

Einstein's Puzzle is a deductive reasoning task where a set of positions must each be assigned a unique combination of items across several categories, guided by a set of interrelated clues (constraints). Solutions can be represented as dictionaries, with categories as keys and ordered lists of items as values. To generate these puzzles, we define a pool of 10 possible categories (e.g., `Job`, `Beverage`, `Food`) and 25 items per category (e.g., `Accountant` and `Doctor` belong to `Job`). For each instance, we randomly select a subset of categories and items, enumerate all possible permutations, and iteratively add constraints until only one valid solution remains. The final puzzle is constructed using a natural language template that integrates the scenario description, sampled categories, items, and constraints.

AR-LSAT questions assess logical reasoning within a structured system of relationships by requiring valid conclusions based on a set of rules and conditions. For example, a problem might involve

[10] `https://huggingface.co/TheFinAI`

[11] `https://networkx.org/`

[12] `https://en.wikipedia.org/wiki/Zebra_Puzzle`

[13] `https://www.lsac.org/lsat`

scheduling presentations for `Mary`, `John`, and `Alice` on `Monday`, `Tuesday`, and `Wednesday`, with constraints such as `John` presenting after `Mary`. The task is to determine which conclusions follow logically. We use questions collected from [53] as data points.

Since both Einstein's Puzzle and AR-LSAT are CSPs, we employ `GPT-4.1-mini` [18] to generate Python code using the `python-constraint` library[14], ensuring each problem is solvable and has a unique solution.

Mathematical Programming The mathematical programming domains focus on solving optimization problems for an objective function that is subject to constraints. We mainly obtain questions directly from the existing tutorials from Gurobipy[15] [54], PySCIPOpt[16] [55], CVXPY[17] [56], and Statsmodels[18] [57]. For all these datasets, we began by reviewing all official example notebooks provided in their corresponding documentation and identified a subset that met our criteria for practical relevance and technical clarity. Specifically, we selected notebooks that included detailed real-world problem descriptions, utilized data from identifiable sources, and were compatible with the latest Jupyter environment. From these, we curated a collection of problems where the accompanying data were either regenerated using `GPT-4o` [45]–guided by the original sources–or partially retained to preserve solvability. Markdown formatting issues were corrected, and scripts were streamlined by removing non-essential outputs such as extraneous `print` statements and visualizations. Logical inconsistencies and errors within the solution code were minimally edited, and final answers were marked using the `\boxed` notation for clarity.

Medicine We constructed our dataset from the MedCalc-Bench benchmark[19] through the following procedure. We start by merging the "patient note" and "question" columns into a unified question field, and designating the "Ground Truth Answer" column as the "final answer". To ensure diversity across categories, we sampled up to 30 entries for each unique "Calculator Name." For calculators with fewer than 30 available entries, we included all available data. This yielded a total of 1,192 examples.

We then improved the official tool code [58] by adding docstrings and performing additional code optimizations. The entire codebase was then consolidated into a Python package and also stored in a single text file. We prompted `GPT-4.1` [18] to generate code (as rationale) capable of solving each question using functions from the medcalc-bench package. We filtered the data by discarding any question-answer pairs for which the generated rationale code failed to execute successfully. This filtering step resulted in 1,117 valid examples. We then compared the outputs of the executed rationales with the corresponding "final answer" fields using a rule-based regular expression matching system. Through this process, we identified 916 question-answer pairs with matched outputs.

Security & Safety We focused on extracting representative examples from existing cryptographic Capture-The-Flag (CTF) problems and their associated explanations. Specifically, we manually curated problems from the CTF-Wiki[20], which encompasses 3 major categories and 44 subcategories of cryptographic challenges. For each subcategory, we ensured coverage by including at least 3-5 representative problems. To diversify and scale the dataset, we systematically replaced the plaintext and ciphertext in the original problems to generate multiple variants, ensuring each subcategory included at least 5 distinct instances. Where applicable, we used `GPT-4o` [45] to rewrite the textual components of the problems, making them more natural while preserving the cryptographic core.

For each problem, we also collected the corresponding solution logic and reference implementation. With the assistance of `GPT-4o` [45], we manually refined these into coherent rationales that explain the step-by-step solution process. All solution scripts were executed in a Python interpreter, and their outputs were compared against the official flags or expected results. Any examples with mismatched outputs were discarded. The final dataset consists of problem statements, rationale code, and verified

[14] `https://pypi.org/project/python-constraint/`
[15] `https://github.com/Gurobi/modeling-examples`
[16] `https://github.com/scipopt/PySCIPOpt`
[17] `https://www.cvxpy.org/examples/`
[18] `https://www.statsmodels.org/stable/examples/index.html`
[19] `https://ncbi/MedCalc-Bench-v1.0`
[20] `https://ctf-wiki.org/crypto/introduction/`

outputs for each data point. All entries were manually reviewed to ensure correctness, clarity, and reproducibility.

Programming The dataset taken from the LeetCode dataset [59] and is constructed through a two-stage rollout procedure aimed at building a code critic model–an LLM designed to identify and fix errors in solutions to competitive programming problems. Each data point consists of a programming problem, an initial candidate solution, an automatic correctness judgment, and a potential correction. In the first round rollout, an initial candidate solution is generated using `Claude-3.5-Sonnet` [60] for each programming task. These candidate solutions then go through a second round evaluation where `DeepSeek-R1` [1]. This model acts as an automated code critic: it first determines whether the initial solution is correct, outputting a binary true/false label. If the solution is found to be incorrect, the model attempts to revise it by producing a corrected version that better aligns with the problem specification.

B System Prompt for Benchmarking Experiments

We use tailored prompts to standardize the generation and evaluation processes across all models tested in our benchmark. These prompts are described below.

Solver System Prompt

```
You are an {{DOMAIN}} solver.  It is imperative that you end with your answer wrapped in a
\boxed{} statement.  It should only be the final answer, e.g., correct:  \boxed{\frac{1}{4}},
incorrect:  \boxed{x = \frac{1}{4}}.
```

LLM-as-Judge System Prompt

```
You are an answer judge.  Your task is the following:  You will be provided with a GROUND TRUTH
ANSWER, and with an entire answer from an LLM output.  At the end of the LLM output, there should
be a \boxed{} statement containing an answer.
Your task is to compare this answer to the GROUND TRUTH ANSWER. You should only focus on the boxed
statement at the end of the LLM response, disregard any reasoning steps before it.
You should end your response with your judgement wrapped in a \boxed{} statement.
Wrap your evaluation result as \boxed{1} if the provided LLM answer is mathematically equivalent
(EVEN IF THE FORMAT IS DIFFERENT!!)  to the GROUND TRUTH ANSWER. Wrap it as \boxed{0} if the
provided LLM answer is either not existent or not mathematically equivalent.
```

These prompts ensure consistency in both generation and evaluation phases and are critical to maintaining reproducibility and fairness in our benchmarking pipeline.

C Broader Impacts

This work contributes LOONGBENCH and LOONGENV as resources for studying reasoning and alignment at scale, which may positively impact the community by providing diverse, verifiable benchmarks and reducing reliance on costly human supervision; however, synthetic data generation also carries risks of propagating biases from seed datasets or being misused in high-stakes applications, so we encourage responsible use with attention to limitations, safeguards, and clear documentation.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction accurately reflect the paper's contributions and scope.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We explicitly acknowledge assumptions, dataset coverage, and scope, and the potential future works that show the current limitations.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: The paper does not introduce formal theorems or proofs.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide task setups, prompts, model settings, and evaluation procedures sufficient to reproduce key results.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: We will release the full seed dataset, generation framework, and instructions in the supplemental material with reproduction steps should the paper get accepted.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: We specify datasets, models, hyperparameters, and evaluation configurations.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: We report accuracy without error bars due to single-pass API evaluations and compute limits.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We document hardware (e.g., H100 80GB) and API usage.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work adheres to the NeurIPS Code of Ethics, including anonymity and data-use considerations.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss potential positive uses (e.g., verifiable reasoning data) and risks in Appendix C.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: We do not release high-risk foundation models or scraped sensitive datasets requiring access controls.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All third-party assets are cited with versions and terms, and licenses are respected.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Newly released datasets/code are documented with usage instructions, limitations, and licensing.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing or human-subjects studies.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: No human-subjects research was conducted, so IRB approval is not applicable.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: We describe LLM usage for data generation and judging where it affects methodology and evaluation.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.