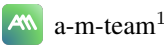


# AM-Thinking-v1: Advancing the Frontier of Reasoning at 32B Scale

Yunjie Ji, Xiaoyu Tian, Sitong Zhao, Haotian Wang,  
Shuaiting Chen, Yiping Peng, Han Zhao, Xiangang Li



## Abstract

We present AM-Thinking-v1, a 32B dense language model that advances the frontier of reasoning, embodying the collaborative spirit of open-source innovation. Outperforming DeepSeek-R1 and rivaling leading Mixture-of-Experts (MoE) models like Qwen3-235B-A22B and Seed1.5-Thinking, AM-Thinking-v1 achieves impressive scores of 85.3 on AIME 2024, 74.4 on AIME 2025, and 70.3 on LiveCodeBench, showcasing state-of-the-art mathematical and coding capabilities among open-source models of similar scale.

Built entirely from the open-source Qwen2.5-32B base model and publicly available queries, AM-Thinking-v1 leverages a meticulously crafted post-training pipeline — combining supervised fine-tuning and reinforcement learning — to deliver exceptional reasoning capabilities. This work demonstrates that the open-source community can achieve high performance at the 32B scale, a practical sweet spot for deployment and fine-tuning. By striking a balance between top-tier performance and real-world usability, we hope AM-Thinking-v1 inspires further collaborative efforts to harness mid-scale models, pushing reasoning boundaries while keeping accessibility at the core of innovation. We have open-sourced our model on Hugging Face<sup>2</sup>.

	AM-Thinking-v1	DeepSeek-R1	Qwen3-235B-A22B	Qwen3-32B	Seed1.5-Thinking	Nemotron-Ultra-253B	OpenAI-o1	OpenAI-o3-mini	Gemini2.5-Pro
	Dense, 32B	MoE, 671B	MoE, 235B	Dense, 32B	MoE, 200B	Dense, 256B	2024-12-17	Medium	
AIME2024	85.3	79.8	85.7	81.4	86.7	80.8	74.3	79.6	92.0
AIME2025	74.4	70.0	81.5	72.9	74.0	72.5	79.2	74.8	86.7
LiveCodeBench <small>(v4, 2024.10.2025.02)</small>	70.3	64.3	70.7	65.7	64.9	68.1	63.9	66.3	70.4
Arena-Hard	92.5	93.2	95.6	93.8	-	87.0	92.1	89.0	96.4

1. AIME 24/25: Each query is sampled 64 times. We report the average of the accuracy. AIME'25 consists of Part I and Part II, with a total of 30 questions.  
2. LiveCodeBench: Each query is sampled 16 times.  
3. Arena-Hard: Each query is sampled 1 times.  
4. The maximum inference length is 48k tokens.

Figure 1: Comparison of Model Performance on Reasoning Benchmarks

<sup>1</sup>The a-m-team is an internal team at Beike (Ke.com), dedicated to exploring AGI technology.

<sup>2</sup><https://huggingface.co/a-m-team/AM-Thinking-v1>

# 1 Introduction

Over the past six months, large language models (LLMs) have demonstrated remarkable improvements in reasoning, particularly in domains such as mathematical problem solving and code generation—tasks that require sophisticated logical inference. These advancements are expanding the practical applicability of LLMs across a broader range of real-world scenarios.

The release of DeepSeek-R1[1] has shown that open-source communities are increasingly capable of building models that rival proprietary systems such as OpenAI’s o1[2], Google’s Gemini 2.5[3], and Anthropic’s Claude 3.7[4]. More recently, the emergence of Qwen3-235B-A22B[5] has further advanced the reasoning frontier of open-source models. However, many recent breakthroughs rely on extremely large-scale Mixture-of-Experts (MoE) architectures, which impose significant infrastructure burdens and make model deployment and fine-tuning considerably more complex.

In contrast, dense models of moderate size (e.g., 32B) offer better efficiency and deployability, yet often lag behind their MoE counterparts in reasoning performance. This contrast raises a critical research question: *Can we unlock the reasoning potential of 32B-scale dense models—without relying on private data or massive MoE architectures—through a carefully designed post-training pipeline?*

To explore this question, we introduce and open-source AM-Thinking-v1, a reasoning-optimized language model built upon the publicly available Qwen2.5-32B[6, 7] base model. Our model achieves state-of-the-art performance among dense models of comparable size and even outperforms much larger MoE models in several reasoning benchmarks. Specifically, AM-Thinking-v1 achieves impressive scores of 85.3 and 74.4 on AIME2024[8] and AIME2025[9], two challenging math competition-style benchmarks, and 70.3 on LiveCodeBench[10], a widely used benchmark for evaluating code generation. It surpasses DeepSeek-R1 (671B MoE) and approaches or matches the performance of other top-tier MoE models such as Qwen3-235B-A22B and Seed1.5-Thinking[11], despite having only a fraction of their parameters.

Our success stems from a meticulously designed post-training framework that leverages publicly available training queries. We applied strict preprocessing to various open-source queries and instructions, including deduplication, removal of low-quality or multi-modal queries (e.g., those involving images), and thorough decontamination with respect to our evaluation benchmarks. In particular, for mathematical queries—where we observed a high prevalence of noisy items—we constructed a comprehensive data processing pipeline that spans query filtering and ground-truth verification.

The post-training pipeline comprises two main stages: Supervised Fine-Tuning (SFT) and Reinforcement Learning (RL). Starting from Qwen2.5-32B base model, we apply SFT using a cold-start dataset that encourages a *"think-then-answer"* pattern and builds initial reasoning capability. During RL, we incorporate difficulty-aware query selection and a two-stage training procedure to ensure both training stability and progressive improvement in performance.

In summary, AM-Thinking-v1 demonstrates that even without large-scale MoE architectures, dense models at the 32B scale can achieve reasoning capabilities comparable to the best available models. We hope this work serves as a practical reference for the community, highlighting how careful post-training design can bridge the performance gap while retaining the deployability advantages of moderate-scale models. This offers a promising direction for future research at the intersection of scalability, accessibility, and reasoning performance.

# 2 Data

All queries used in our training come from publicly available datasets. We begin by deduplicating the queries and filtering out low-quality ones. For mathematical queries with ground-truth answers, we further verify the correctness of the provided ground truth. Additionally, we filter model-generated responses based on quality and assign difficulty levels to each query based on the pass rate observed across multiple response attempts.

## 2.1 Data Collection

Our training data is collected from multiple publicly available open source datasets, spanning tasks such as mathematical reasoning, code generation, scientific reasoning, instruction follow, and general chat.

**Mathematical Reasoning** During the collection of mathematical data, we ensure that each data point include a verifiable ground truth. We incorporate datasets such as OpenR1-Math-220k[12], Big-Math-RL-Verified[13], data\_ablation\_full59K[14], NuminaMath[15], MetaMathQA[16], 2023\_amc\_data[17], DeepMath-103K[18], and AIME\_1983\_2024[19].

**Code Generation** We ensure that all collected code data include verifiable test cases. Datasets selected for this category include PRIME[20], DeepCoder[21], KodCode[22], liveincode\_generation[10], codeforces\_cots[23], verifiable\_coding[24], opencoder[25], OpenThoughts-114k-Code\_decontaminated[12], and AceCode-87K[26].

**Scientific Reasoning** This category includes natural sciences (physics, chemistry, natural sciences) and logical reasoning. They primarily consist of multiple-choice questions, each paired with a reliable ground truth. We include datasets such as task\_mmmlu[27], chemistryQA[28], Llama-Nemotron-Post-Training-Dataset-v1[29], LOGIC-701[30], ncert[31, 32, 33, 34, 35, 36], and logicLM[37].

**Instruction Follow (IF)** We select two instruction-following datasets: Llama-Nemotron-Post-Training-Dataset[38], tulu-3-sft-mixture[39].

**General Chat** This category includes a broad range of tasks, covering open-ended queries, general knowledge, and everyday reasoning, and it supports both single-turn and multi-turn interactions. The selected datasets are evol[40], InfinityInstruct[41], open\_orca[42], tulu-3-sft-mixture[39], natural\_reasoning[43], flan[44], ultra\_chat[45], and OpenHermes-2.5[46].

## 2.2 Query filtering

After collecting the data, we first remove duplicates, then apply two cleaning steps to address common query quality issues:

- **Removal of queries containing URLs.** Since the model cannot access external links during training, the presence of URLs may lead to hallucinations or misleading outputs.
- **Removal of image-referencing queries.** Since our model is purely text-based, it cannot perceive or process any visual information; such queries are therefore excluded from training.

Finally, we remove queries from the training set that are similar to those in the evaluation set, using both exact matching and semantic deduplication.

### 2.2.1 Mathematical query filtering

During our analysis of the mathematical data, we identify issues with unclear or incomplete query descriptions and incorrect ground truths. To address the former, we use an LLM to analyze and filter out queries lacking clear or complete descriptions. For the latter, we implement a rigorous ground truth validation process: for each query, we prompt DeepSeek-R1[1] to generate multiple responses and compare the most frequent answer (Deepseek-R1-common) with the original ground truth using `math_verify`<sup>3</sup>. Discrepancies between model predictions and the original ground truth prompt us to re-evaluate the correctness of certain annotations. For these cases, we consult o4-mini[47] to obtain an alternative answer (o4-mini-answer). If `math_verify` determines that o4-mini-answer and Deepseek-R1-common produce equivalent results, we consider the original ground truth potentially incorrect and revise it to o4-mini-answer.

Following this processing, we further identify and handle specific data types unsuitable for training: mathematical proof problems and queries with multiple sub-questions are filtered out. While multiple-choice questions are also deemed unsuitable, their significant volume prompt us to rewrite them as fill-in-the-blank questions instead of discarding them.

<sup>3</sup><https://github.com/huggingface/Math-Verify>

## 2.3 Synthetic response filtering

After query filtering, we apply three methods to filter out low-quality synthetic response:

- **Perplexity-based Filtering.** We use our previously trained 32B model[48] to compute the perplexity (PPL) of each model-generated response. Responses with PPL scores exceeding a predefined threshold are discarded.
- **N-gram-based Filtering.** We discard model responses containing repeated phrases of a certain minimum length that appear consecutively.
- **Structure-based Filtering.** For multi-turn dialogues, we ensure that the final turn is an assistant response. Additionally, we require that each model-generated reply contains both a complete think and answer component.

For each query, multiple responses are generated. We then compute a verify score for every query to assess response quality or query difficulty. For queries with ground truth answers, we calculate the pass rate across the multiple generated responses. For queries without ground truth, we employ a large language model (LLM)-based reward model to score each response, and use the average score as the final verification signal. The scoring procedure is detailed in Section 3.

## 3 Reward

### 3.1 Verifiable Queries

In the case of math, code, and instruction-following (IF) queries with available ground truth or test cases, we employ rule-based verification or code execution to assess the correctness of model responses.

#### 3.1.1 Math

For mathematical queries, the reward is determined by verifying the model’s final answer. The process begins by extracting the answer from the last boxed (`{}`) content of the model’s answer content. This extracted answer is then validated against the reference using `math_verify`<sup>4</sup>. This tool normalizes answers to handle different representations for comparison, outputting `true` on match or `false` otherwise. Reward score is 1 for a correct answer and 0 for an incorrect answer.

#### 3.1.2 Code

For code queries equipped with predefined test cases, the verification process is executed within a secure code sandbox environment. This sandbox currently supports evaluation for multiple programming languages, including Python and C++.

**Code Segmentation Extraction** Extraction of code segmentation is facilitated by specific delimiters. For example, Python code blocks are identified by enclosing them within ````python` and `````, while C++ code blocks utilize ````cpp` and `````. This delimitation approach is similar to that commonly used in Markdown.

**Test Case** The execution of code in our dataset primarily takes two forms: method call and standard input/output. For these two forms, two distinct test case types are employed. The details of each type are provided below:

- **Method Call Test Cases:** These queries require the implementation of a specific method or function. Test cases for this type are defined by a particular function name, along with input values and their corresponding expected outputs. To facilitate efficient execution within the sandbox environment, these test cases are automatically converted into assertion statements.
- **Standard Input/Output Test Cases:** These queries that typically do not specify a distinct entry function are common in scenarios such as competitive programming or scripting tasks, where code reads from standard input and writes to standard output. For these queries, test cases are handled via standard input (`stdin`) and standard output (`stdout`).

Method Call	Standard Input/Output								
<p><b>&lt;Function Name&gt;</b> filter_schedules_by_class</p> <p><b>&lt;INPUT&gt;</b> 0, 1, [[1, 0, 1], [0, 1, 0], [1, 1, 1]]</p> <p><b>&lt;OUTPUT&gt;</b> [[1, 1, 1], [1, 1, 1]]</p> <hr/> <p><b>&lt;TEST CASE&gt;</b> assert filter_schedules_by_class (0, 1, [[1, 0, 1], [0, 1, 0], [1, 1, 1]]) == [[1, 0, 1], [1, 1, 1]]</p>	<p><b>&lt;INPUT&gt;</b> [[2,1],[99,100],[98,99]]</p> <p><b>&lt;OUTPUT&gt;</b> 0.989898989899</p> <hr/> <p><b>&lt;TEST CASE&gt;</b></p> <table> <tr> <th>stdin</th><th>stdout</th></tr> <tr> <td>2 1</td><td>0.989898989899</td></tr> <tr> <td>99 100</td><td></td></tr> <tr> <td>98 99</td><td></td></tr> </table>	stdin	stdout	2 1	0.989898989899	99 100		98 99	
stdin	stdout								
2 1	0.989898989899								
99 100									
98 99									

Figure 2: Method Call And Standard Input/Output test case examples

**Robust Cloud Sandbox Environment** Meeting the dual requirements of secure code execution (mitigating malicious content risks) and robust performance under high concurrency (preventing unexpected timeouts) is central to our code sandbox design. This is achieved by deploying the sandbox as a distributed cloud service leveraging multiple machines. The distributed architecture, combined with load balancing and queue management, ensures both the isolation necessary for security and the capacity needed for reliable high-volume execution.

For each code query with its typically associated multiple test cases, the final reward score is 1 if all tests pass, and 0 otherwise.

### 3.1.3 Instruction Follow

To obtain a reward score for instruction follow, we employ the IFEval[49] validator as our verifier. The validator receives instruction identifiers (instruction\_id\_list) and their arguments (kwargs). Here is an example:

```
{
  "query": "What are some tips for managing stress during a pandemic?
Your response should contain at least 3 bullet points.
Use the markdown bullet points such as: * This is point 1.
Also, your response should contain less than 100 words.
In 2008, a groundbreaking 9-page PDF changed the money game:
Bitcoin's whitepaper. Authored by the mysterious Satoshi Nakamoto,
it introduced a peer-to-peer electronic cash system bypassing central banks.",
  "instruction_id_list": [
    "detectable_format:number_bullet_lists",
    "length_constraints:number_words"
  ],
  "kwargs": [
    {"num_bullets": 3},
    {"num_words": 100, "relation": "less than"}
  ]
}
```

Figure 3: Validator Input Example

For a given response, it returns a boolean result (True/False) for each individual instruction, indicating successful following or not. We used the strict mode of IFEval[49] validator, which assesses only the original response and is considered more rigorous than the loose mode. Since one instance

<sup>4</sup><https://github.com/huggingface/Math-Verify>

can be associated with multiple instructions, the final reward score is 1 if all these instructions are successfully followed, and 0 otherwise.

### 3.2 Non-Verifiable Queries

For queries lacking objective verification criteria, reward score is conducted using a reward model-based approach. We employ reward model, which provides three distinct scores for each generated response, measuring helpfulness, correctness, and coherence. Let  $S_{Help}$ ,  $S_{Corr}$ , and  $S_{Coher}$  denote the scores for helpfulness, correctness, and coherence, respectively. The final reward score ( $S_{final}$ ) for a response is then computed as the average of these three scores.

## 4 Approach

### 4.1 Supervised Fine-Tuning

**Data** Our Supervised Fine-Tuning (SFT) training uses approximately 2.84 million samples, covering five major categories: math, code, science, instruction follow, and general chat. Figure 4 illustrates the distribution of SFT data at both the instance level and the token level. For some data with relatively fewer samples, such as Instruction Follow, we upsample them by repeating the data several times during training to ensure balanced learning across tasks. In the case of more challenging queries, we include multiple synthetic responses to enhance diversity and robustness in training.

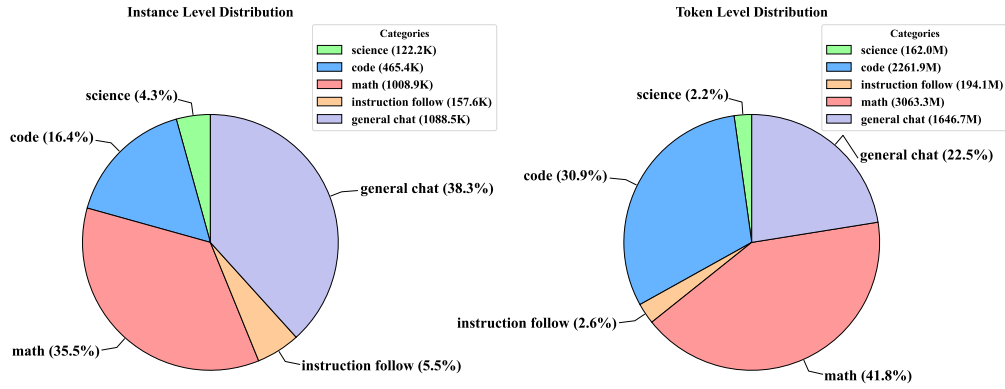


Figure 4: Instance Level Distribution (left) and Token Level Distribution (right) during SFT. It is worth noting that the proportions are computed over responses, not queries, since a single query can correspond to multiple responses in our training set.

**Training Configuration** We conduct SFT based on Qwen2.5-32B[6, 7] base model. We observed supervised fine-tuning pattern shifts[50] (See section 5.3 for more details), which prompted us to adopt a larger learning rate and batch size to ensure stable convergence and effective learning. The training uses a learning rate of  $8e-5$ , a maximum sequence length of 32k with sequence packing, and discards samples that exceed 32k tokens. The global batch size is set to 64, and the model is trained for 2 epochs. We employ a cosine warmup strategy, with warmup steps set to 5% of total training steps, and the learning rate decays to 0 thereafter. For multi-turn dialogue data, only the final response, which contains the reasoning process, is used as the training target and contributes to the loss, in order to focus learning on the reasoning component.

### 4.2 Reinforcement Learning

We observe that selecting training queries of appropriate difficulty plays a crucial role in ensuring stable performance improvements during the reinforcement learning (RL) stage[51]. To this end, prior to RL, we filter our math and code queries based on their pass rates obtained from the SFT model: we retain only those queries with pass rates strictly between 0 and 1. This ensures that the training data remains sufficiently challenging to drive learning, while avoiding instances that are

either too easy or excessively difficult, which could lead to stagnation or instability during training. We ultimately retain 32k and 22k math and code queries, respectively.

Our RL pipeline consists of two stages. When the model’s performance plateaus in the first stage, we transition to the second stage. In Stage 2, we remove all math and code queries that the model answered correctly with 100% accuracy in Stage 1, and supplement the training set with 15k general chat and 5k instruction-following data to improve broader generalization.

We adapt Group Relative Policy Optimization (GRPO)[52] as our training algorithm. Despite being a simplified and lightweight variant of Proximal Policy Optimization (PPO)[53], we find that GRPO offers strong training stability and effective performance gains. The training is configured as follows:

- **No KL Constraint.** We remove the KL penalty originally used in GRPO, allowing for more substantial policy updates.
- **Handling Overlong Responses.** For responses exceeding a certain length threshold during rollout, we set their advantages to zero to prevent them from influencing parameter updates.
- **Strict on-policy training.** Each training batch consists of 256 queries, and for every query, we sample 16 rollouts. The policy model only updates once following each exploration stage.
- **Two-stage Generation and Learning Rate Schedule.** In Stage 1, we limit the maximum response length to 24K tokens and use a relatively high learning rate of  $4 \times 10^{-6}$ . In Stage 2, we increase the maximum response length to 32K and reduce the learning rate to  $1 \times 10^{-6}$ .

In our early experiments across models of varying parameter scales, we observe that using a larger learning rate in the first training stage enables the model to reach convergence more quickly, whereas a smaller learning rate requires significantly more training steps to achieve similar performance. Although both approaches eventually lead to comparable outcomes, we adopt a larger learning rate in the first stage to accelerate convergence and reduce overall training costs.

### 4.3 RL Framework

Our training pipeline is built upon the verl framework [54], using GRPO[52] for reinforcement learning. verl is an open-source RL framework. Integrated with vLLM[55], FSDP, and Megatron-LM[56], verl enables scalable RL training across 1000+ GPUs.

We expand verl further with modifications to best suit our training strategy.

#### 4.3.1 Rollout Speed Optimization

RL with online sample generation (rollout) on large LLMs often suffers from long training periods. Each training step takes several minutes to tens of minutes. Unlike SFT or DPO, online GRPO requires policy model sample generation during each step, increasing per-step latency. This rollout phase occupies more than 70% elapsed time of one training step from our observation, thus optimization is needed.

Some recent works also investigated on the efficiency of the rollout phase. [57] proposed partial rollouts to divide long sequences into each rollout step. While feasible, it introduced additional effort to manage the replay buffer. [11] sought to decouple model evolution from runtime execution allowing for a dynamic mixture of on/off-policy samples. We use pure on-policy in this work.

Although already accelerated by fast inferencing frameworks like vLLM and sglang, the speed of the rollout phase in verl can still be optimized:

First, the training is synchronized. The whole generation batch must all be completed before we can move on to the next phase. We have to wait for the longest sequences in a batch to complete, causing a long-tail effect. Furthermore, longer sequences have lower tokens per second due to the nonlinear cost of self-attention and limited GPU memory bandwidth for kv-cache. For a 32B model on a typical 4x A100 vLLM instance, We can observe a roughly 60 tokens/s when sequence length is short and only about 50 tokens/s at 32k length. This long-tail effect leads to significant idle time across faster workers and underutilized GPU capacity.

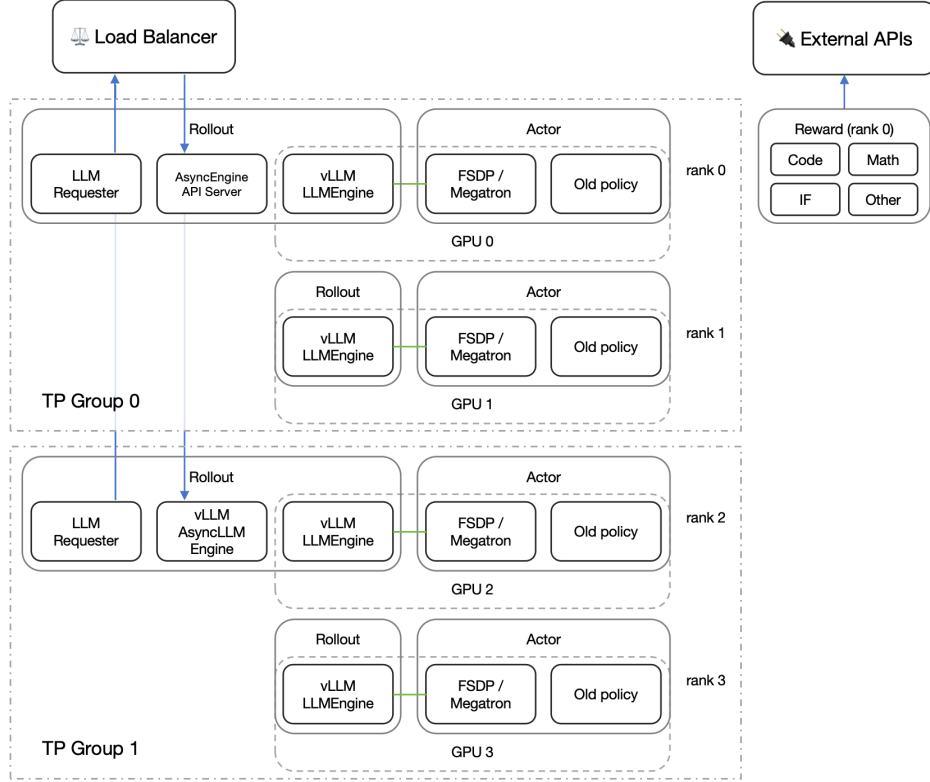


Figure 5: Detached Rollout and Upgrade with Streaming Load Balancing Architecture

Second, generation length varies between different prompts and random samples, this further imposes unbalanced load among inferencing instances, e.g. random samples of one hard problem which demands more tokens are all on one instance. This imbalance, together with the memory bandwidth issue, make the long-tail issue even worse. At 32k length and batch size 16, a total of 460 tokens/s throughput can be seen from the same instance, making only 28 tokens/s for a single sequence. Batch size 32 further decrease single sequence token throughput to 19.

In practice, we found that these "crowded" instances can take 30% longer time for rollout than others. Instead of directly speeding up the inferencing framework's generation speed which might involve scheduling and kv-cache management, we choose to optimize the load balancing strategy:

- For our first approach, we use static load balancing to spread the random sampling of one prompt across multiple instances. verl applies an SPMD design. By coupling the **rollout** and **update** workers, it achieves faster weight sharding and data sharing with simpler design. However the random sampling is done inside the rollout workers using an batched inference call, leaving the same prompt bound to the same inferencing instance. Simply moving the repeat sampling out of rollout worker into the trainer, with additional shuffling, relaxed this constraint. This change alleviates the imbalanced load, relieved the crowded instances from having to run many long sequences with low per sequence throughputs, as well as decouples the *world\_size* and *batch\_size*: now the *batch\_size* can be smaller than number of GPUs *world\_size* divided by tensor parallel size *tp\_size*.
- We further detach the rollout worker from the inference engine, enabling dynamic instance allocation via a custom load-balancer aware of real-time system metrics. The system now have the flexibility to *dynamically* allocate the inferencing instance, *on-the-fly*, for *each* generation sample. To achieve this, we add the frontend server to the offline vLLM engine inside the rollout worker, exposing an API endpoint, attach the endpoints of all instances to a custom load balancer, then invoke this aggregated endpoint from each rollout worker. By implementing the load balancer with awareness of each instances' current load and speed



metrics, we can reroute the long sequences on crowded workers to not-so-crowded replicas. Even without a disaggregated prefill and decode system, the streaming load balancer can still achieve more optimal global scheduling and throughput.

This "detached" (see figure 5) design allows the rollout logic independent of the engine, and is more suitable for future work e.g. agent and tool use scenarios.

### 4.3.2 Reward Computation

verl natively supports setting reward for each sources. We further add reward classes for instruction following (IF) and general (other) problems. Part of the resource-heavy or risky operations are executed via remote API, e.g. LLM-as-a-judge and code sandbox.

## 5 Experiments

### 5.1 Evaluation

#### 5.1.1 Benchmarks

We evaluate our models on a diverse set of challenging benchmarks:

- **American Invitational Mathematics Examination 2024 (AIME2024)**[8]: A challenging mathematical reasoning competition dataset comprising 30 integer-answer questions designed to assess precise mathematical reasoning.
- **American Invitational Mathematics Examination 2025 (AIME2025)**[9]: A set of problems for the 2025 AIME competition, which contains 30 problems from the 2025 AIME-part1 and AIME-part2.
- **LiveCodeBench (LCB)**[10]: A comprehensive, contamination-free coding benchmark, continuously aggregating new programming challenges from platforms such as LeetCode, AtCoder, and Codeforces. Similar to Qwen3[5], we use queries collected between October 2024 and February 2025 for evaluation purposes.
- **Arena-Hard**[58]: A data pipeline to build high-quality benchmarks from live data in Chatbot Arena, where model responses are judged via pairwise comparison using GPT-4-Turbo-1106[2] as the arbiter.

#### 5.1.2 Evaluation Methodology

Standardized evaluation conditions were maintained across all benchmarks. The maximum generation length was set to 49,152 tokens. For benchmarks requiring stochastic sampling, we uniformly employ a temperature of 0.6 and a top-p value of 0.95.

Specifically, for AIME2024[8] and AIME2025[9], we generate 64 responses per query to calculate pass@1 precision. For LiveCodeBench[10], 16 responses per query were generated to estimate pass@1. For Arena-Hard, one response per query was generated and evaluated using GPT-4 Turbo (1106).

#### 5.1.3 Prompting Strategy

A consistent system prompt was utilized for all evaluations to guide the model’s response format:

You are a helpful assistant. To answer the user’s question, you first think about the reasoning process and then provide the user with the answer. The reasoning process and answer are enclosed within <think> </think> and <answer> </answer> tags, respectively, i.e., <think> reasoning process here </think> <answer> answer here </answer>.

User prompts were adapted based on the benchmark:

- For AIME 2024 and AIME 2025, the following instruction was appended to each query: Let’s think step by step and output the final answer within `\box{}`.
- For LiveCodeBench and Arena-Hard, the default user prompts provided by the respective benchmarks were used without any additional modifications.

#### 5.1.4 Baselines

We compare AM-Thinking-v1 against a set of strong baseline models, including both proprietary and open-source systems. These baselines provide a representative view of the model’s performance in today’s competitive LLM landscape:

- **DeepSeek-R1**[1]: A powerful model known for its strong performance in code-related tasks and mathematical problem-solving. Its reported performance serves as a robust high-performing baseline.
- **Qwen3-235B-A22B**[5]: With 235 billion total parameters and 22 billion active parameters, this open-source MoE model demonstrates the strongest reasoning performance among existing open-source language models.
- **Qwen3-32B**[5]: Another release from the Qwen3 series, which is a dense 32-billion parameter model.
- **Seed1.5-Thinking**[11]: A reasoning model has a total parameter of 200 billion and activation parameters of 20 billion. Through the reinforcement learning framework and refined data strategy, it outperforms DeepSeek-R1 in tasks such as mathematics, programming, and scientific reasoning.
- **Nemo-Ultra-256B**[59]: A large-scale model from NVIDIA, representing the potential of scaling model size for improved performance on challenging tasks.
- **OpenAI-o1 (dated 2024-12-17)**[2]: An early model from OpenAI, providing a historical perspective on the evolution of their capabilities in these domains.
- **OpenAI-o3-mini (Medium)**[47]: OpenAI’s upgraded reasoning model that adopts the medium reasoning-effort setting.
- **Gemini2.5-Pro**[3]: Google’s latest generation model, representing the cutting edge of multi-modal and reasoning capabilities.

For evaluations on AIME2024[19], AIME2025[9], LiveCodeBench[10], and Arena-Hard benchmarks, the performance metrics for DeepSeek-R1[1], Qwen2-32B-A22B[5], Qwen3-32B[6], OpenAI-o1 (dated 2024-12-17)[2], OpenAI-o3-mini (Medium)[47], and Gemini2.5-Pro[3] are sourced from the Qwen3[5] technical report. The results for Seed1.5-Thinking[11] and Nemo-Ultra-256B[59] on these same benchmarks are directly cited from their respective technical reports.

## 5.2 Results

Table 1: Comparison across reasoning benchmarks

	AM-Thinking v1	DeepSeek R1	Qwen3-235B A22B	Qwen3 32B	Seed1.5 Thinking	Nemotron- Ultra-253B	OpenAI o1	OpenAI o3-mini	Gemini 2.5 Pro
<b>Math</b>									
AIME 2024	85.3	79.8	85.7	81.4	86.7	80.8	74.3	79.6	92.0
AIME 2025	74.4	70.0	81.5	72.9	74.0	72.5	79.2	74.8	86.7
<b>Code</b>									
LiveCodeBench (v5, 2024.10–2025.02)	70.3	64.3	70.7	65.7	64.9	68.1	63.9	66.3	70.4
<b>General Chat</b>									
Arena-Hard	92.5	93.2	95.6	93.8	–	87.0	92.1	89.0	96.4

We evaluate AM-Thinking-v1 on multiple reasoning benchmarks and compare it with several leading large-scale models, as shown in Table 1. On mathematical tasks, AM-Thinking-v1 achieves scores of 85.3 and 74.4 on AIME2024 and AIME2025, respectively, outperforming or closely matching larger models such as DeepSeek-R1 and Qwen3-235B-A22B.

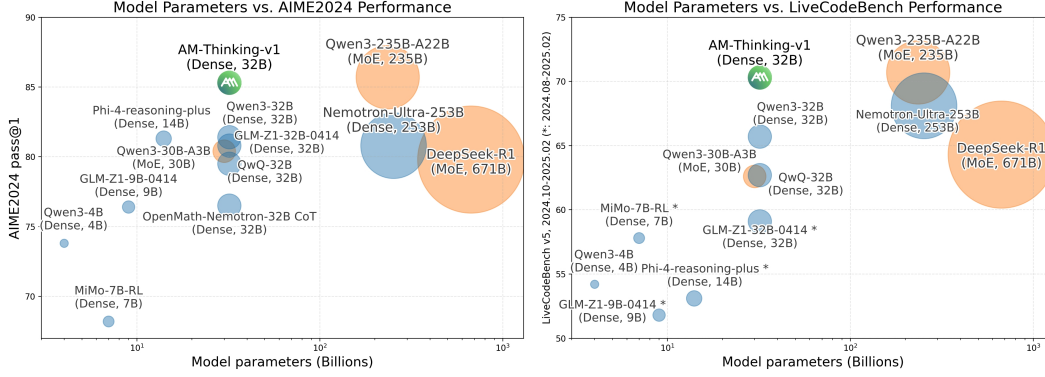


Figure 6: Performance versus model size on AIME2024 (left) and LiveCodeBench (right). Each point represents a model, with the x-axis indicating model size (in number of parameters) and the y-axis indicating benchmark score. Models closer to the top-left corner achieve better performance with smaller size.

On the LiveCodeBench benchmark, which focuses on code reasoning, AM-Thinking-v1 attains a score of 70.3, substantially surpassing DeepSeek-R1 (64.3), Qwen3-32B (65.7), and Nemotron-Ultra-253B (68.1), demonstrating strong capabilities in code understanding and generation.

On the general chat benchmark Arena-Hard, AM-Thinking-v1 obtains a score of 92.5, which is competitive with several proprietary models such as OpenAI-o1 (92.1) and o3-mini (89.0). However, its performance still lags behind Qwen3-235B-A22B (95.6), indicating that there remains room for improvement in general conversational capabilities.

Figure 6 illustrates the relationship between model size and performance on AIME2024 (left) and LiveCodeBench (right). AM-Thinking-v1 achieves the strongest performance among dense models of similar scale, and comes close to the performance of much larger MoE models, striking an effective balance between efficiency and performance.

### 5.3 Supervised Fine-Tuning Pattern Shift

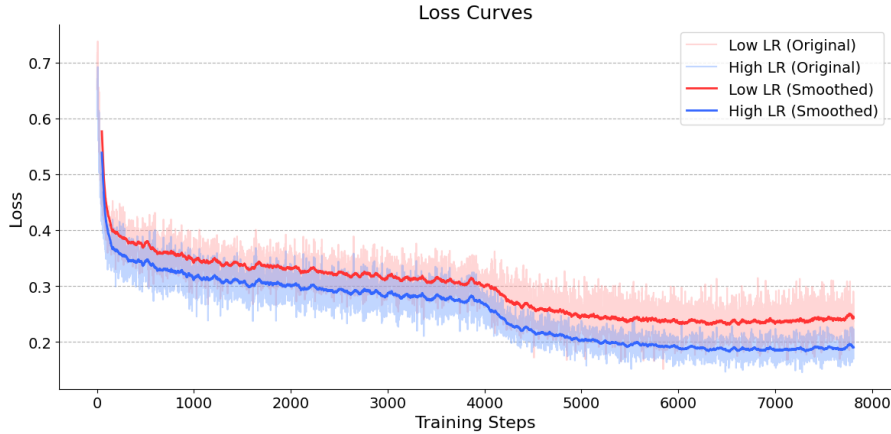


Figure 7: Supervised Fine-Tuning (SFT) training loss curves.

Compared to traditional SFT, we find that supervised fine-tuning on long-form reasoning tasks leads to a pattern shift. To achieve stable convergence, this stage requires a larger learning rate and batch size; otherwise, the model struggles to fit the data effectively. For example, while traditional SFT might use a learning rate around  $8 \times 10^{-6}$  with a batch size of approximately 0.5M tokens, supervised fine-tuning on long-form reasoning often requires a learning rate as high as  $8 \times 10^{-5}$  and a batch size of around 2M tokens. Figure 7 shows training loss during SFT.

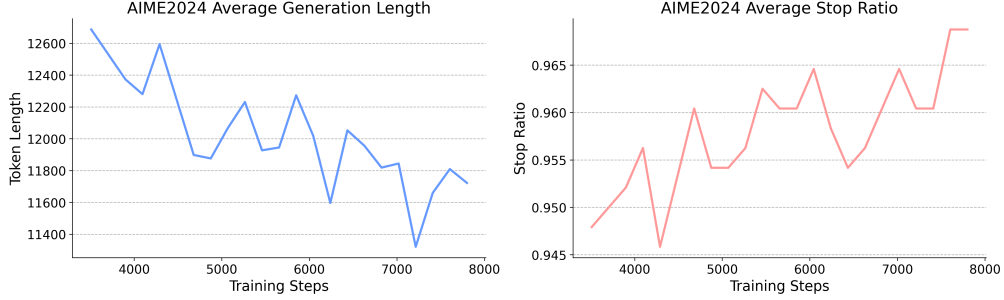


Figure 8: Variation in Average Generation Length (left) and Average Stop Ratio (right).

As shown in Figure 8, we track the evolution of Average Generation Length and Average Stop Ratio during SFT on the AIME2024. At the early stages of training, the model tends to generate excessively long outputs with a low stop ratio. This is largely due to the nature of the base model’s pretraining corpus, which predominantly consists of plain text, as well as the fact that reasoning examples in our dataset are significantly longer than standard instruction data. As training progresses, we observe a consistent decrease in average generation length alongside a steady increase in stop ratio. This trend indicates that the model is gradually learning the structural and semantic patterns inherent in long-form reasoning prompts. The alignment of these dynamic metrics suggests that our fine-tuning methodology effectively guides the model toward more coherent and task-aligned reasoning behavior.

## 6 Conclusion and Limitations

In this work, we present AM-Thinking-v1, a 32B dense language model that demonstrates state-of-the-art reasoning capabilities among open-source models of comparable size. Our model surpasses DeepSeek-R1 and even approaches the performance of top-tier Mixture-of-Experts (MoE) models like Qwen3-235B-A22B and Seed1.5-Thinking on reasoning-intensive tasks.

This result is made possible by a carefully designed post-training pipeline based on open-source training queries and base model. Through systematic data preprocessing, thorough ground truth verification, and a carefully designed SFT and RL framework, we successfully elicit advanced reasoning capabilities from a moderately sized model.

Our findings suggest that with the right data and training design, 32B-scale dense models remain a highly practical and competitive choice—offering a compelling balance between deployment efficiency and reasoning performance. We hope that AM-Thinking-v1 serves as a foundation for further research exploring the full potential of mid-scale models.

While AM-Thinking-v1 performs well in reasoning and open-domain chat, it lacks support for structured function-calling, tool use, and multimodal inputs, limiting its applicability in agent-based or cross-modal scenarios. Safety alignment remains preliminary, and further red-teaming is needed. Additionally, its performance may vary across low-resource languages and domain-specific tasks.

## References

- [1] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shihong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.
- [2] OpenAI. Learning to reason with llms, 2024.
- [3] Google DeepMind. Gemini 2.5: Our most intelligent ai model, 2025.
- [4] Anthropic. Claude 3.7 sonnet and claude code, 2025.
- [5] Qwen Team. Qwen3, April 2025.
- [6] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.
- [7] Qwen Team. Qwen2.5: A party of foundation models, September 2024.
- [8] MAA. American invitational mathematics examination - aime. <https://maa.org/math-competitions/american-invitational-mathematics-examination-aime>, feb 2024. Accessed in February 2024, from American Invitational Mathematics Examination - AIME 2024.
- [9] Yixin Ye, Yang Xiao, Tiantian Mi, and Pengfei Liu. Aime-preview: A rigorous and immediate evaluation framework for advanced mathematical reasoning. <https://github.com/GAIR-NLP/AIME-Preview>, 2025. GitHub repository.
- [10] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.
- [11] ByteDance Seed, :, Jiaze Chen, Tiantian Fan, Xin Liu, Lingjun Liu, Zhiqi Lin, Mingxuan Wang, Chengyi Wang, Xiangpeng Wei, Wenyan Xu, Yufeng Yuan, Yu Yue, Lin Yan, Qiying Yu, Xiaochen Zuo, Chi Zhang, et al. Seed1.5-thinking: Advancing superb reasoning models with reinforcement learning, 2025.
- [12] Hugging Face. Open r1: A fully open reproduction of deepseek-r1, January 2025.
- [13] Alon Albalak, Duy Phung, Nathan Lile, Rafael Rafailov, Kanishk Gandhi, Louis Castricato, Anikait Singh, Chase Blagden, Violet Xiang, Dakota Mahan, and Nick Haber. Big-math: A large-scale, high-quality math dataset for reinforcement learning in language models, 2025.
- [14] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling, 2025.
- [15] Jia LI, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. Numinamath. <https://huggingface.co/AI-MO/NuminaMath-CoT>, 2024.
- [16] Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.
- [17] Art of Problem Solving. 2023 AMC 8 Problems/Problem 10. [https://artofproblemsolving.com/wiki/index.php/2023\\_AMC\\_8\\_Problems/Problem\\_10](https://artofproblemsolving.com/wiki/index.php/2023_AMC_8_Problems/Problem_10), 2023. Accessed: 2025-04-23.
- [18] Zhiwei He, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xingyu Chen, Yue Wang, Linfeng Song, Dian Yu, Zhenwen Liang, Wenxuan Wang, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Deepmath-103k: A large-scale, challenging, decontaminated, and verifiable mathematical dataset for advancing reasoning. 2025.

- [19] Di Zhang. Aime\_1983\_2024 (revision 6283828), 2025.
- [20] Lifan Yuan, Wendi Li, Huayu Chen, Ganqu Cui, Ning Ding, Kaiyan Zhang, Bowen Zhou, Zhiyuan Liu, and Hao Peng. Free process rewards without process labels. *arXiv preprint arXiv:2412.01981*, 2024.
- [21] Michael Luo, Sijun Tan, Roy Huang, Ameen Patel, Alpay Ariyak, Qingyang Wu, Xiaoxiang Shi, Rachel Xin, Colin Cai, Maurice Weber, Ce Zhang, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepcoder: A fully open-source 14b coder at o3-mini level. <https://pretty-radio-b75.notion.site/>, 2025. Notion Blog.
- [22] Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. Kodcode: A diverse, challenging, and verifiable synthetic dataset for coding. 2025.
- [23] Guilherme Penedo, Anton Lozhkov, Hynek Kydlíček, Loubna Ben Allal, Edward Beeching, Agustín Piqueres Lajarín, Quentin Gallouédec, Nathan Habib, Lewis Tunstall, and Leandro von Werra. Codeforces cots. <https://huggingface.co/datasets/open-r1/codeforces-cots>, 2025.
- [24] Open R1. Verifiable coding problems (python). <https://huggingface.co/datasets/open-r1/verifiable-coding-problems-python>, 2025. Hugging Face Dataset.
- [25] Siming Huang, Tianhao Cheng, Jason Klein Liu, Jiaran Hao, Liuyihan Song, Yang Xu, J. Yang, J. H. Liu, Chenchen Zhang, Linzheng Chai, Ruifeng Yuan, Zhaoxiang Zhang, Jie Fu, Qian Liu, Ge Zhang, Zili Wang, Yuan Qi, Yinghui Xu, and Wei Chu. Opencoder: The open cookbook for top-tier code large language models. 2024.
- [26] Huaye Zeng, Dongfu Jiang, Haozhe Wang, Ping Nie, Xiaotong Chen, and Wenhui Chen. Acecoder: Acing coder rl via automated test-case synthesis. *ArXiv*, abs/2207.01780, 2025.
- [27] Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Anjana Arunkumar, Arjun Ashok, Arut Selvan Dhanasekaran, Atharva Naik, David Stap, Eshaan Pathak, Giannis Karamanolakis, Haizhi Gary Lai, et al. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks, 2022.
- [28] Microsoft. Chemistry-qa. <https://github.com/microsoft/chemistry-qa>, 2021. [GitHub repository].
- [29] NVIDIA. Llama-3\_1-nemotron-ultra-253b-v1. [https://huggingface.co/nvidia/Llama-3\\_1-Nemotron-Ultra-253B-v1](https://huggingface.co/nvidia/Llama-3_1-Nemotron-Ultra-253B-v1), 2025. Released on 2025-04-07 under the NVIDIA Open Model License.
- [30] hivaze. Logic-701: A benchmark dataset for logical reasoning in english and russian. <https://huggingface.co/datasets/hivaze/LOGIC-701>, 2023. Hugging Face Dataset.
- [31] Parth Kadam. Ncert physics 12th dataset. [https://huggingface.co/datasets/KadamParth/NCERT\\_Physics\\_12th](https://huggingface.co/datasets/KadamParth/NCERT_Physics_12th), 2023. Accessed: 2024-04-23.
- [32] Parth Kadam. Ncert physics 11th dataset. [https://huggingface.co/datasets/KadamParth/NCERT\\_Physics\\_11th](https://huggingface.co/datasets/KadamParth/NCERT_Physics_11th), 2023. Accessed: 2024-04-23.
- [33] Parth Kadam. Ncert chemistry 11th dataset. [https://huggingface.co/datasets/KadamParth/NCERT\\_chemistry\\_11th](https://huggingface.co/datasets/KadamParth/NCERT_chemistry_11th), 2023. Accessed: 2024-04-23.
- [34] Parth Kadam. Ncert chemistry 12th dataset. [https://huggingface.co/datasets/KadamParth/NCERT\\_chemistry\\_12th](https://huggingface.co/datasets/KadamParth/NCERT_chemistry_12th), 2023. Accessed: 2024-04-23.
- [35] Parth Kadam. Ncert biology 11th dataset. [https://huggingface.co/datasets/KadamParth/NCERT\\_Biology\\_11th](https://huggingface.co/datasets/KadamParth/NCERT_Biology_11th), 2023. Accessed: 2024-04-23.
- [36] Parth Kadam. Ncert biology 12th dataset. [https://huggingface.co/datasets/KadamParth/NCERT\\_Biology\\_12th](https://huggingface.co/datasets/KadamParth/NCERT_Biology_12th), 2023. Accessed: 2024-04-23.
- [37] longface. logiclm. <https://huggingface.co/datasets/longface/logicLM>, 2025. Accessed: 2025-04-22.

- [38] Akhiad Bercovich, Itay Levy, Izik Golan, Mohammad Dabbah, Ran El-Yaniv, Omri Puny, Ido Galil, Zach Moshe, Tomer Ronen, Najeeb Nabwani, Ido Shahaf, Oren Tropp, Ehud Karpas, Ran Zilberstein, Jiaqi Zeng, Soumye Singhal, Alexander Bukharin, Yian Zhang, Tugrul Konuk, Gerald Shen, Ameya Sunil Mahabaleshwarkar, et al. Llama-nemotron: Efficient reasoning models, 2025.
- [39] Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, et al. Tulu 3: Pushing frontiers in open language model post-training. 2024.
- [40] Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. WizardLM: Empowering large pre-trained language models to follow complex instructions. In *The Twelfth International Conference on Learning Representations*, 2024.
- [41] Beijing Academy of Artificial Intelligence (BAAI). Infinity instruct. *arXiv preprint arXiv:2406.XXXX*, 2024.
- [42] Wing Lian, Bleys Goodson, Eugene Pentland, Austin Cook, Chanvichet Vong, and "Teknium". Openorca: An open dataset of gpt augmented flan reasoning traces. <https://huggingface.co/datasets/Open-Orca/OpenOrca>, 2023.
- [43] Weizhe Yuan, Jane Yu, Song Jiang, Karthik Padthe, Yang Li, Dong Wang, Ilia Kulikov, Kyunghyun Cho, Yuandong Tian, Jason E Weston, and Xian Li. Naturalreasoning: Reasoning in the wild with 2.8m challenging questions, 2025.
- [44] 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, 2023.
- [45] Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations, 2023.
- [46] Teknium. Openhermes 2.5: An open dataset of synthetic data for generalist llm assistants, 2023.
- [47] OpenAI. Introducing openai o3 and o4-mini, 2025.
- [48] Han Zhao, Haotian Wang, Yiping Peng, Sitong Zhao, Xiaoyu Tian, Shuaiting Chen, Yunjie Ji, and Xiangang Li. 1.4 million open-source distilled reasoning dataset to empower large language model training. *arXiv preprint arXiv:2503.19633*, 2025.
- [49] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models, 2023.
- [50] Xiaoyu Tian, Sitong Zhao, Haotian Wang, Shuaiting Chen, Yiping Peng, Yunjie Ji, Han Zhao, and Xiangang Li. Deepdistill: Enhancing llm reasoning capabilities via large-scale difficulty-graded data training, 2025.
- [51] Yunjie Ji, Sitong Zhao, Xiaoyu Tian, Haotian Wang, Shuaiting Chen, Yiping Peng, Han Zhao, and Xiangang Li. How difficulty-aware staged reinforcement learning enhances llms' reasoning capabilities: A preliminary experimental study, 2025.
- [52] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024.
- [53] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.
- [54] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

- [55] 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.
- [56] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.
- [57] Kimi AI. Kimi-k1.5: Reinforcement learning enhanced llm reasoning. <https://github.com/Kimi-AI/Kimi-K1.5>, 2024. Accessed: March 2025.
- [58] Tianle Li\*, Wei-Lin Chiang\*, Evan Frick, Lisa Dunlap, Banghua Zhu, Joseph E. Gonzalez, and Ion Stoica. From live data to high-quality benchmarks: The arena-hard pipeline, April 2024.
- [59] NVIDIA. Nemo-ultra-256b. <https://developer.nvidia.com/nemo>, 2024. Large language model released by NVIDIA as part of the NeMo framework.