KIMI K1.5: SCALING REINFORCEMENT LEARNING WITH LLMS

TECHNICAL REPORT OF KIMI K1.5

Kimi Development Team

ABSTRACT

Language model pretraining with next token prediction has proved effective for scaling compute but is limited to the amount of available training data. Scaling reinforcement learning (RL) unlocks a new axis for the continued improvement of artificial intelligence, with the promise that large language models (LLMs) scale their training data by learning to explore with rewards, and thus also scaling compute. However, prior published work has not produced competitive results. In light of this, we report on the training practice of Kimi k1.5, our latest multi-modal LLM trained with RL, including RL training techniques, multi-modal data recipes, and infrastructure optimization. We study the training and test time scaling properties of RL with LLMs. Our approach is simplistic and shows that strong performance can be achieved without complex techniques such as Monte Carlo tree search, value functions, and process reward models. Notably, our system achieves state-of-the-art reasoning performance across multiple benchmarks and modalities—e.g., xx on AIME, xx on Codeforces, xx on MathVista—matching OpenAI's o1. The models can be accessed from kimi.ai.

1 Introduction

2 Pretraining

@zhuhan @haoyu

2.1 Language Data

Our pretraining corpus is designed to provide comprehensive and high-quality data for training large language models (LLMs). It encompasses five major domains: English, Chinese, Code, Mathematics & Reasoning, and Knowledge, amounting to a total of 15T tokens. The data distribution across these domains is illustrated in Figure 1a. We employ sophisticated filtering and quality control mechanisms for each domain to ensure the highest quality training data. For all pretraining data, we conducted rigorous individual validation for each data source to assess its specific contribution to the overall training recipe. This systematic evaluation ensures the quality and effectiveness of our diverse data composition.

English and Chinese textual data we developed a multi-dimensional quality filtering framework that combines multiple scoring methods to reduce individual biases and ensure comprehensive quality assessment. Our framework incorporates:

1. **Rule-based filtering**: We implement domain-specific heuristics to remove problematic content, including duplicate content, machine-translated text, and low-quality web scrapes. We also filter out documents with excessive special characters, unusual formatting, or spam patterns.
2. **FastText-based classification**: We trained specialized FastText models to identify content quality based on linguistic features and semantic coherence. This helps identify documents with natural language flow and proper grammatical structure.
3. **Embedding-based similarity analysis**: Using document embeddings, we compute document-level similarity scores to identify and remove near-duplicates while preserving semantically valuable variations. This approach helps maintain diversity in our training corpus.

4. **LLM-based quality assessment**: We leverage LLMs to score documents based on coherence, informativeness, and potential educational value. This method is particularly effective at identifying nuanced quality indicators that simpler methods might miss.

The final quality score for each document is computed as a combination of these individual scores. Based on extensive empirical analysis, we implement dynamic sampling rates where high-quality documents are upsampled while low-quality documents are downsampled during training. Figure 1b shows the final data quality distribution.

Code data The code data primarily consists of two categories. For the pure code data derived from code files, we adhered to the methodology of BigCode [Li et al., 2023, Lozhkov et al., 2024] and conducted a comprehensive preprocessing of the dataset. Initially, we eliminated miscellaneous languages and applied a rule-based cleaning procedure to enhance data quality. Subsequently, we addressed language imbalance through strategic sampling techniques. Specifically, markup languages such as JSON, YAML, and YACC were down-sampled, while 32 major programming languages, including Python, C, C++, Java, and Go, were up-sampled to ensure a balanced representation. Regarding the text-code interleaved data sourced from various data sources, we trained an embedding based scoring model to recall high-quality data. This approach ensures the diversity of the data and maintains its high quality.

Math & Reasoning data The mathematics and reasoning component of our dataset is crucial for developing strong analytical and problem-solving capabilities. Mathematical pre-training data is primarily retrieved from web text data and PDF documents collected from public internet sources. Initially, we discovered that our general-domain text extraction, data cleaning process and OCR models exhibited high false negative rates in the mathematical domain. Therefore, we first developed specialized data cleaning procedures and OCR models specifically for mathematical content, aiming to maximize the recall rate of mathematical data. Subsequently, we implemented a two-stage data cleaning process:

1. Using FastText model for initial cleaning to remove most irrelevant data.
2. Utilizing a fine-tuned language model to further clean the remaining data, resulting in high-quality mathematical data.

Knowledge data The knowledge corpus is meticulously curated to ensure comprehensive coverage across academic disciplines. Our knowledge base primarily consists of academic exercises, textbooks, research papers and other general educational literature. A significant portion of these materials are digitized through OCR processing, for which we have developed proprietary models optimized for academic content, particularly for handling mathematical formulas and special symbols.

We employ internal language models to annotate documents with multi-dimensional labels, including:

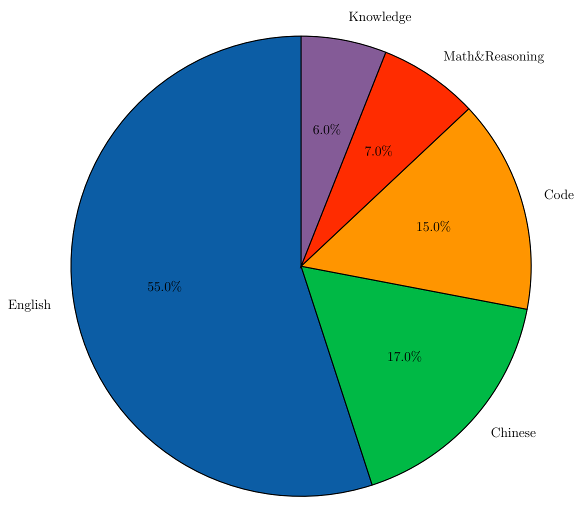
1. OCR quality metrics to assess recognition accuracy
2. Educational value indicators measuring pedagogical relevance
3. Document type classification (e.g., exercises, theoretical materials)

Based on these multi-dimensional annotations, we implement a sophisticated filtering and sampling pipeline. First and foremost, documents are filtered through OCR quality thresholds. Our OCR quality assessment framework places special attention on detecting and filtering out common OCR artifacts, particularly repetitive text patterns that often indicate recognition failures.

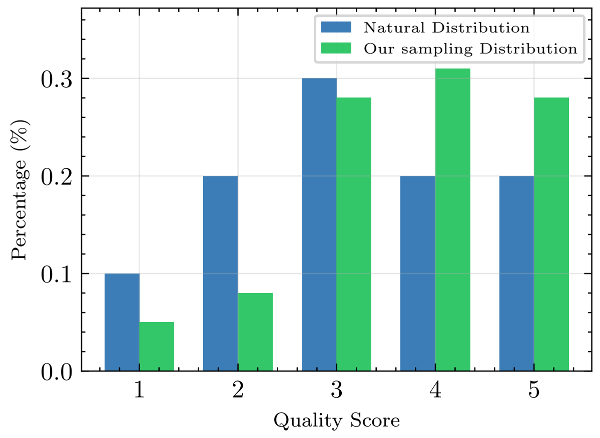
Beyond basic quality control, we carefully evaluate the educational value of each document through our scoring system. Documents with high pedagogical relevance and knowledge depth are prioritized, while maintaining a balance between theoretical depth and instructional clarity. This helps ensure that our training corpus contains high-quality educational content that can effectively contribute to the model's knowledge acquisition.

Finally, to optimize the overall composition of our training corpus, the sampling strategy for different document types is empirically determined through extensive experimentation. We conduct isolated evaluations to identify document subsets that contribute most significantly to the model's knowledge acquisition capabilities. These high-value subsets are upsampled in the final training corpus. However, to maintain data diversity and ensure model generalization, we carefully preserve a balanced representation of other document types at appropriate ratios. This data-driven approach helps us optimize the trade-off between focused knowledge acquisition and broad generalization capabilities.

Text Cooldown Data Following the pre-training phase, we implement a text cooldown training stage. The cooldown corpus encompasses five distinct domains, maintaining consistency with the pre-training data distribution. Through empirical investigation, we observed that the incorporation of synthetic data during the cooldown phase yields significant



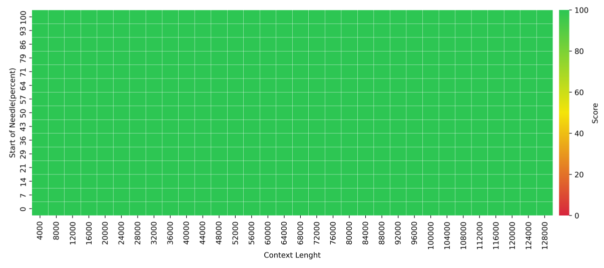
(a) Domain Distribution



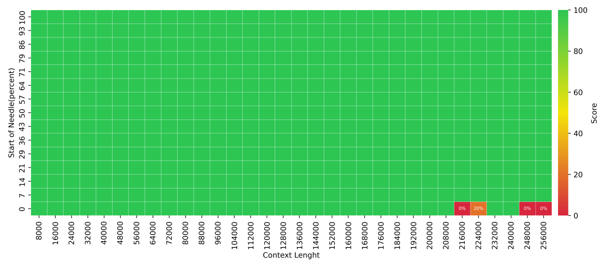
(b) Quality score distribution

performance improvements, particularly in mathematical reasoning, knowledge-based tasks, and code generation. The English and Chinese components of the cooldown dataset are curated from high-fidelity subsets of the pre-training corpus. For math, knowledge, and code domains, we employ a hybrid approach: utilizing selected pre-training subsets while augmenting them with synthetically generated content. Specifically, we leverage existing mathematical, knowledge and code corpora as source material to generate question-answer pairs through a proprietary language model, implementing rejection sampling techniques to maintain quality standards. These synthesized QA pairs undergo comprehensive validation before being integrated into the cooldown dataset.

Text Long-context Cooldown Data To ensure excellent long-text capabilities of the base model, the RoPE frequency [**?**] was set to 1,000,000. During training, we conducted length activation training in two stages with context lengths of 32k and 128k. After activation training, we used YARN [Peng et al., 2023] technology to further extend the context length to 256k. [can we not specify YARN here? –dylan] The figure 2 includes NIAH test results for the model at both 128k and 256k lengths. [We need an all-green needle test for 256k. We had it for 2m. –dylan] In length activation training, we used 40% full attention data and 60% partial attention data for training. The full attention data came partly from high-quality natural data and partly from synthetic long context Q&A and summary data. The partial attention data came from uniform sampling of cooldown data.



(a) 128k context length



(b) 256k context length supported by YARN

Figure 2: Needle in a Haystack Test

2.2 Multimodal Data

During pretraining, we performed joint training of vision and language in the later stages of the language model training to optimize training efficiency. The multi-modal pretraining dataset used in k1.5 consists of approximately 900B tokens [current 900B tokens are composed of both image and text, maybe we don't need to mention this explicitly. –ychen], which comprises a diverse range of publicly available sources and in-house data. We provide a detailed description of these sources in this section, which is organized into the following categories:

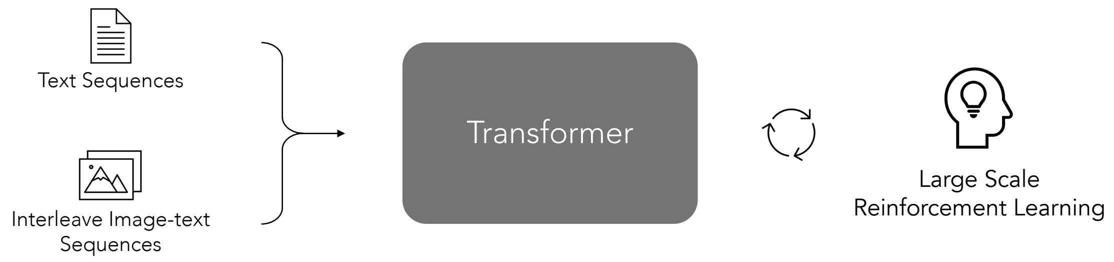

Text Sequences
Interleave Image-text Sequences
Transformer
Large Scale Reinforcement Learning

Figure 3: Kimi k1.5 supports interleaved images and text as input, leveraging large-scale reinforcement learning to enhance the model's reasoning capabilities.

Caption data provides the model with fundamental modality alignment and rich world knowledge. In this study, we utilized various opensource Chinese and English caption datasets like [Schuhmann et al., 2022, Gadre et al., 2024]. We also collected a significant amount of in-house caption data. A series of data cleaning processes were applied to the caption data, including but not limited to global deduplication, CLIP score filtering, image quality screening and rephrasing, to ensure the quality of the caption data.

Interleave data helps further enhance the model's few-shot and multi-image capabilities. We also observed that it contributes positively to maintaining the model's language abilities. In this study, we considered open-source interleave datasets [Zhu et al., 2024, Laurençon et al., 2024] and constructed large-scale in-house data using resources like textbooks. Moreover, we discovered that many interleave images were improperly positioned within the text. Therefore, in addition to the filters applied to the caption data, we also performed a complete document restructuring for the interleave data.

OCR data is also crucial for aligning the model more closely with human values. In addition to open-source datasets, we manually constructed a large amount of in-house OCR data, which includes multilingual, dense, web-based, and handwritten data, among others. We applied extensive data augmentation techniques to enhance its robustness, including but not limited to rotation, distortion, color modification, and noise addition. These efforts have endowed our model with strong OCR capabilities.

General QA Data We also found that incorporating a large amount of high-quality QA datasets during pretraining yields significant benefits. Specifically, we considered high-quality academic datasets on tasks includinggrounding, table/chart question answering, web agents, and general question answering. Additionally, we constructed a substantial amount of in-house QA data to further enhance the model's performance. We ensure that the difficulty and diversity of our general question answering dataset are evenly distributed by using scoring models and detailed manual classification, resulting in improved performance.

Vision Long-context Cooldown Data Following the text setting, we also incorporated high-quality long-context vision data for joint vision-language cooldown training. Most vision long-context data comes from long docs in the interleave dataset, with a smaller portion constructed by concatenating data from other sources. This enables our model to handle both long-text vision and text inputs simultaneously.

2.3 Model Architecture

Kimi k-series models employ a variant of the Transformer decoder that integrates multimodal capabilities alongside improvements in architecture and optimization strategies, illustrated in Figure 3. These advancements collectively support stable large-scale training and efficient inference, tailored specifically to large-scale reinforcement learning and the operational requirements of Kimi users.

Extensive scaling experiments indicate that most of the base model performance comes from improvements in the quality and diversity of the pretraining data. Specific details regarding model architecture scaling experiments lie beyond the scope of this report and will be addressed in future publications.

3 Post Training

3.1 Supervised Finetuning

The supervised finetuning (SFT) for K1.5 model has two major stages. The vanilla SFT to activates the model's ability of following instructions, answering questions and basic reasoning, etc., and during the Long-CoT SFT stage, the model learns meta-reasoning abilities such as reflection and verification.

3.1.1 Vanilla Supervised Finetuning

We create the vannila SFT corpus covering multiple domains. For non-reasoning tasks, such as question-answering, writing and text processing, we first build a seed dataset via human annotation to train a seed model, and then generate multiple responses on a diverse prompt set with the seed model. The annotators then rank these responses and modify the best response to be the final one. For reasoning tasks such as math and coding problems, where rule-based verification is more accurate and efficient than human, we use rejection sampling to enlarge the SFT dataset.

Finally, our vanilla SFT dataset contains about 1 million examples, composed of 500k examples in general question-answering, 200k in coding, 200k in math and k12 problems, 5k in creative writing and 20k in long-context tasks, including summarization, doc-qa, translation and writing.

Additionally, we construct a text-vision joint SFT dataset containing 500k examples, encompassing chart interpretation, OCR, image-grounded conversations, visual coding, vision reasoning, and math and K-12 problems with visual aids.

We first train the model at the sequence length of 32k tokens for 1-epoch, where over-length data samples are ignored, followed by another epoch at the sequence length of 128k tokens. In the first stage (32k), the learning rate decays from 2×10^{-5} to 2×10^{-6}, before it re-warmups to 1×10^{-5} in the second stage (128k) and finally decays to 1×10^{-6}. To improve training efficiency, we pack multiple training examples into each single training sequence.

3.1.2 Long-CoT Supervised Finetuning

@yangyang We collect a diverse set of problems, including those in STEM fields, various competitions, and general reasoning domains, with both text-only and image-text combined questions. From this collection, we filter out questions that require rich reasoning while remaining straightforward to evaluate. Specifically, we removed questions that could lead to false positive reasoning paths, such as multiple choice and true/false questions, as well as those with overly complex answers. In addition, we used a model to predict answers directly; If the model could easily guess the answer, those questions were also filtered out. This process allowed us to curate a diverse and balanced QA dataset, considering both difficulty levels and the distribution between textual and visual content.

Using the refined QA dataset, we engaged in extensive prompt engineering to construct a series of long-chain-of-thought (long-CoT) warm-up data with correct answers for both text and image input. These data were rich in human-like cognitive processes such as planning, evaluation, reflection, and verification. Through a lightweight supervised fine-tuning (SFT) process on this warm-up dataset, we trained a long-CoT model that exhibits more detailed and systematic reasoning patterns.

3.2 Reinforcement Learning

3.2.1 Problem Setting

Given a training dataset $\mathcal{D} = \{(x_i, y_i^*)\}_{i=1}^n$ of problems x_i and corresponding ground truth answers y_i^*, our goal is to train a policy model π_θ to accurately solve test problems. In the context of complex reasoning, the mapping of problem x to solution y is non-trivial. To tackle this challenge, the *chain of thought* (CoT) method proposes to use a sequence of intermediate steps $z = (z_1, z_2, \dots, z_m)$ to bridge x and y, where each z_i is a coherent sequence of tokens that acts as a significant intermediate step toward solving the problem [Wei et al., 2022]. When solving problem x, thoughts $z_t \sim \pi_\theta(\cdot|x, z_1, \dots, z_{t-1})$ are auto-regressively sampled, followed by the final answer $y \sim \pi_\theta(\cdot|x, z_1, \dots, z_m)$. We use $y, z \sim \pi_\theta$ to denote this sampling procedure. Note that both the thoughts and final answer are sampled as a language sequence.

To further enhance the model's reasoning capabilities, *planning* algorithms are employed to explore various thought processes, generating improved CoT at inference time [Yao et al., 2024, Wu et al., 2024, Snell et al., 2024]. The core insight of these approaches is the explicit construction of a search tree of thoughts guided by value estimations. This allows the model to explore diverse continuations of a thought process or backtrack to investigate new directions when encountering dead ends. In more detail, let $\mathcal{T}$ be a search tree where each node represents a partial solution

$s = (x, z_{1:|s|})$. Here s consists of the problem x and a sequence of thoughts $z_{1:|s|} = (z_1, \dots, z_{|s|})$ leading up to that node, with $|s|$ denoting number of thoughts in the sequence. The planning algorithm uses a critic model v to provide feedback $v(x, z_{1:|s|})$, which helps evaluate the current progress towards solving the problem and identify any errors in the existing partial solution. We note that the feedback can both be provided by a discriminative score or language sequence[Zhang et al., 2024]. Guided by the feedbacks for all $s \in \mathcal{T}$, the planning algorithm selects the most promising node for expansion, thereby growing the search tree. The above process repeats iteratively until a full solution is derived.

We can also approach planning algorithms from an *algorithmic perspective*. Given past search history available at the t-th iteration $(s_1, v(s_1), \dots, s_{t-1}, v(s_{t-1}))$, a planning algorithm $\mathcal{A}$ iteratively determines the next search direction $\mathcal{A}(s_t|s_1, v(s_1), \dots, s_{t-1}, v(s_{t-1}))$ and provides feedbacks for the current search progress $\mathcal{A}(v(s_t)|s_1, v(s_1), \dots, s_t)$. Since both thoughts and feedbacks can be viewed as intermediate reasoning steps, and these components can both be represented as sequence of language tokens, we use z to replace s and v to simplify the notations. Accordingly, we view a planning algorithm as a mapping that directly acts on a sequence of reasoning steps $\mathcal{A}(\cdot|z_1, z_2, \dots)$. In this framework, all information stored in the search tree used by the planning algorithm is flattened into the full context provided to the algorithm. This provides an intriguing perspective on generating high-quality CoT: Rather than explicitly constructing a search tree and implementing a planning algorithm, we could potentially train a model to approximate this process. Here, the number of thoughts (i.e., language tokens) serves as an analogy to the computational budget traditionally allocated to planning algorithms. Recent advancements in long context windows facilitate seamless scalability during both the training and testing phases. If feasible, this method enables the model to run an implicit search over the reasoning space directly via auto-regressive predictions. Consequently, the model not only learns to solve a set of training problems but also develops the ability to tackle individual problems effectively, leading to improved generalization to unseen test problems.

We thus consider training the model to generate CoT with reinforcement learning (RL) [**?**]. Let r be a reward model that justifies the correctness of the proposed answer y for the given problem x based on the ground truth y^*, by assigning a value $r(x, y, y^*) \in \{0, 1\}$. Given a problem x, the model π_θ generates a CoT and the final answer through the sampling procedure $z \sim \pi_\theta(\cdot|x)$, $y \sim \pi_\theta(\cdot|x, z)$. The quality of the generated CoT is evaluated by whether it can lead to a correct final answer. In summary, we consider the following objective to optimize the policy

$$\max_\theta \mathbb{E}_{(x,y^*)\sim\mathcal{D},(y,z)\sim\pi_\theta}\left[r(x, y, y^*)\right] . \tag{1}$$

By scaling up RL training, we aim to train a model that harnesses the strengths of both simple prompt-based CoT and planning-augmented CoT. The model still auto-regressively sample language sequence during inference, thereby circumventing the need for the complex parallelization required by advanced planning algorithms during deployment. However, a key distinction from simple prompt-based methods is that the model should not merely follow a series of reasoning steps. Instead, it should also learn critical planning skills including error identification, backtracking and solution refinement by leveraging the entire set of explored thoughts as contextual information.

3.2.2 Policy Optimization

We apply a variant of online policy mirror decent as our training algorithm [Abbasi-Yadkori et al., 2019, Mei et al., 2019, Tomar et al., 2020]. The algorithm performs iteratively. At the i-th iteration, we use the current model π_{θ_i} as a reference model and optimize the following relative entropy regularized policy optimization problem,

$$\max_\theta \mathbb{E}_{(x,y^*)\sim\mathcal{D}}\left[\mathbb{E}_{(y,z)\sim\pi_\theta}\left[r(x, y, y^*)\right] - \tau \mathrm{KL}(\pi_\theta(x)||\pi_{\theta_i}(x))\right] , \tag{2}$$

where $\tau > 0$ is a parameter controlling the degree of regularization. This objective has a closed form solution

$$\pi^*(y, z|x) = \pi_{\theta_i}(y, z|x)\exp(r(x, y, y^*)/\tau)/Z .$$

Here $Z = \sum_{y',z'} \pi_{\theta_i}(y', z'|x)\exp(r(x, y', y^*)/\tau)$ is the normalization factor. Taking logarithm of both sides we have for *any* (y, z) the following constraint is satisfied, which allows us to leverage off-policy data during optimization

$$r(x, y, y^*) - \tau \log Z = \tau \log \frac{\pi^*(y, z|x)}{\pi_{\theta_i}(y, z|x)} .$$

This motivates the following surrogate loss

$$L(\theta) = \mathbb{E}_{(x,y^*)\sim\mathcal{D}}\left[\mathbb{E}_{(y,z)\sim\pi_{\theta_i}}\left[\left(r(x, y, y^*) - \tau \log Z - \tau \log \frac{\pi_\theta(y, z|x)}{\pi_{\theta_i}(y, z|x)}\right)^2\right]\right] .$$

To approximate $\tau \log Z$, we use samples $(y_1, z_1), \dots, (y_k, z_k) \sim \pi_{\theta_i}$: $\tau \log Z \approx \tau \log \frac{1}{k}\sum_{j=1}^k \exp(r(x, y_j, y^*)/\tau)$. We also find that using empirical mean of sampled rewards $\overline{r} = \mathrm{mean}(r(x, y_1, y^*), \dots, r(x, y_k, y^*))$ yields effective practical results. This is reasonable as $\tau \log Z$ approaches the expected reward under π_{θ_i} as $\tau \to \infty$. Finally, we conclude our learning algorithm by taking the gradient of surrogate loss. For each problem x, k responses are sampled using the reference policy π_{θ_i}, and the gradient is given by

$$\frac{1}{k}\sum_{j=1}^{k}\left(\nabla_\theta \log \pi_\theta(y_j, z_j|x)(r(x, y_j, y^*) - \overline{r}) - \frac{\tau}{2}\nabla_\theta\left(\log\frac{\pi_\theta(y_j, z_j|x)}{\pi_{\theta_i}(y_j, z_j|x)}\right)^2\right), \tag{3}$$

To those familiar with policy gradient methods, this gradient resembles the policy gradient of (2) using the mean of sampled rewards as the baseline [Kool et al., 2019, Ahmadian et al., 2024]. The main differences are that the responses are sampled from π_{θ_i} rather than on-policy, and an l_2-regularization is applied. Thus we could see this as the natural extension of a usual on-policy regularized policy gradient algorithm to the off-policy case [Nachum et al., 2017]. We sample a batch of problems from $\mathcal{D}$ and update the parameters to θ_{i+1}, which subsequently serves as the reference policy for the next iteration. Since each iteration considers a different optimization problem due to the changing reference policy, we also reset the optimizer at the start of each iteration.

We exclude the value network in our training system which has also been exploited in previous studies [Ahmadian et al., 2024]. While this design choice significantly improves training efficiency, we also hypothesize that the conventional use of value functions for credit assignment in classical RL may not be suitable for our context. Consider a scenario where the model has generated a partial CoT $(z_1, z_2, \ldots, z_t)$ and there are two potential next reasoning steps: z_{t+1} and z'_{t+1}. Assume that z_{t+1} directly leads to the correct answer, while z'_{t+1} contains some errors. If an oracle value function were accessible, it would indicate that z_{t+1} preserves a higher value compared to z'_{t+1}. According to the standard credit assignment principle, selecting z'_{t+1} would be penalized as it has a negative advantages relative to the current policy. However, exploring z'_{t+1} is extremely valuable for training the model to generate long CoT. By using the justification of the final answer derived from a long CoT as the reward signal, the model can learn the pattern of trial and error from taking z'_{t+1} as long as it successfully recovers and reaches the correct answer. The key takeaway from this example is that we should encourage the model to explore diverse reasoning paths to enhance its capability in solving complex problems. This exploratory approach generates a wealth of experience that supports the development of critical planning skills. Our primary goal is not confined to attaining high accuracy on training problems but focuses on equipping the model with effective problem-solving strategies, ultimately improving its performance on test problems.

3.2.3 Long-to-Short

3.2.4 Coding

@zhuhan

In many online code contests, test cases for problems are not available. Instead, users must submit their own code to the platform and receive a verdict from the official judging servers. This limits the number of programs that we can submit and receive a verdict, since each submission will put load on the official servers and thus limit the scope of code problems. To combat this, we developed a novel approach to automatically generate test cases based on problem statements so that we can judge our programs with our code sandbox (Section 4.1). We target problems without special judge, such that a simple equality test is sufficient for grading answers.

Specifically, we leverage a popular test case generation library CYaRon[1]. We provided the documentation for CYaRon to Kimi, to generate a test input generator based on the problem statement, which in turn generates 50 test cases for each problem. We then randomly sample 30 submissions with verdict accepted. For each test case, we run 10 submissions on the test case input to obtain the output. Only when the answers of at least 7 out of 10 submissions match, we consider the test case output to be standard answer. If at least 40 test cases with a standard answer are generated, we sample 10 accepted submissions and grade the submissions against our generated test cases. Only when at least 9 out of 10 submissions pass the test do we put the problem and generated test cases into our training set.

To be more specific in the statistics, out of 1000 sampled online contest problems, about 614 problems do not require a special judge. We were able to write 463 test case generators with at least 40 valid test cases, out of which 323 problems were added to our training set.

3.2.5 Math

For our mathematics component, we gathered hundreds of thousands of high-quality RL training examples spanning a broad range of difficulties—from K12 through undergraduate, graduate, and competition levels. Each example is

[1]https://github.com/luogu-dev/cyaron

structured as a question paired with a unique, definitive answer. During RL training, we evaluate only the model's final output.

Because mathematical answers can be expressed in numerous ways, we developed a specialized "judge" model that compares any generated answer to the reference solution. This judge model then provides feedback on the LLM's answers throughout the RL training process.

3.2.6 Multi Modalities

@zhiqi @haoyu In the stage of reinforcement learning (RL) training, visual data can be categorized into three primary types. The first type encompasses real-world datasets that incorporate both visual content and tasks necessitating visual reasoning capabilities. The second type pertains to artificially generated, or synthetic, visual reasoning data. The third type involves images that are synthesized from textual information, which ensures that for pure text-based questions, the model can provide consistent responses whether it receives text input or screenshots/photographs.

Vision Data OCR Visual Grounding Vision Reasoning

3.2.7 Sampling Strategies

@zhuhan @dehao

3.2.8 Partial Rollout for Long Context Training

@shupeng

3.2.9 Long2Short Training

@longhui

Although the effectiveness of long-cot in reasoning tasks has been thoroughly validated in our paper (see Figure **??**), where the degree of solved problem for reasoning tasks is positively correlated with token length. However, an excessively lengthy reasoning process is both costly and not human-preferred. In this section, we aim to maintain the model's reasoning performance while minimizing the token count overhead. We use a RL approach to enhance token efficiency. Specifically, we select a model with the best token efficiency as the initial model and train it with RL to maintain a balance between performance and token count. To achieve this, we design a special length penalty for the RL training process. Our approach ads both length rewards and length penalties for all correcrt rollouts, while only adding length penalties for incorrect rollouts.

4 Infrastructure

@haiqing @weixiao @weiran @shupeng

4.1 Code Sandbox

[qiu review]

[too many adjectives. needs significant rewrites. –kk]

The sandbox provides a secure and efficient platform for executing user-submitted code. This environment is particularly advantageous for code reinforcement learning (RL) and code benchmark evaluation. By utilizing the image from MultiPL-E [Cassano et al., 2023], the sandbox can execute code in various languages such as Python, Java, C++, and JavaScript. Additionally, the DMOJ image enables controlled resource evaluation for algorithm competitions, ensuring fair and consistent assessments. The sandbox's flexibility allows it to support additional environments, such as Lean for theorem proving and Jupyter Notebooks for interactive computing, making it adaptable to various specialized tasks. The ability to dynamically switch container images enhances flexibility and efficiency, making the sandbox a powerful tool for both research and practical applications.

For RL, the sandbox offers a controlled setting for consistent and repeatable code execution, essential for training and evaluating models. Real-time feedback mechanisms facilitate immediate assessment and rapid iteration, significantly enhancing the development and optimization of RL models. In benchmark evaluation, the sandbox ensures a standardized environment for executing and comparing various benchmarks across different programming languages.

4.1.1 Scalable Deployment

The code execution service is deployed on Kubernetes (k8s), providing scalability and resilience. Kubernetes allows the service to dynamically scale based on demand, ensuring efficient resource utilization and the ability to handle peak workloads.

Accessible via HTTP endpoints, the service is easy to integrate with external systems, enabling users to submit code and retrieve results seamlessly. Kubernetes features, such as automatic restarts and rolling updates, ensure high availability and fault tolerance, maintaining a reliable execution environment.

These methods collectively provide a robust, flexible, and high-performance platform suitable for a wide range of code benchmarks and advanced use cases like code reinforcement learning.

4.1.2 Performance Enhancements

To ensure optimal performance and support for code reinforcement learning (RL) environments, the code execution service incorporates several effective techniques. These optimizations enhance efficiency, speed, and reliability, making the service suitable for high-demand scenarios and specialized use cases.

- **Using crun:** The service leverages `crun` directly as the container runtime instead of Docker, avoiding Docker's overhead. `crun` is a lightweight and fast container runtime that reduces the overhead associated with container management, leading to faster startup times and lower resource consumption compared to Docker.
- **Cgroup Reusing:** Efficient resource allocation is achieved by pre-creating cgroups for containers to use, reducing the overhead associated with creating new cgroups for each container.
- **Disk Usage Optimization:** The service employs an overlay filesystem with an upper layer mounted as `tmpfs`, providing a fixed-size, high-speed storage space. This approach is particularly beneficial for ephemeral workloads.

Method	Time (s)
Docker	0.12
Sandbox	0.04

(a) Container startup times

Method	Containers/sec
Docker	27
Sandbox	120

(b) Maximum containers started per second on a 16-core machine

These performance optimizations are particularly beneficial for code reinforcement learning. The controlled and efficient execution environment allows for consistent and repeatable evaluation of RL-generated code, essential for iterative training and model refinement. By providing a reliable and high-performance platform, the service significantly enhances the development and evaluation of RL models, contributing to advancements in code generation and optimization.

4.2 Hybrid Deployment of Training and Inference

The entire RL process comprises the following phases:

- **Initial Phase:** At the outset, Megatron [**?**] and vLLM [**?**] are executed within separate containers, which are encapsulated by a shim process known as checkpoint-engine. The checkpoint-engine (Section 4.2.2) functions as an intermediate layer, orchestrating the execution environment for both Megatron and vLLM, and ensuring their integration and management. Neither training nor inference functionalities have been initiated at this stage. The system remains in a preparatory state, awaiting the commencement of either process.
- **Training Phase:** Upon the initiation of training step, the client uploads training data to the distributed storage. Once detected, Megatron commences the training procedure. After the training is completed, Megatron offloads the GPU memory and prepares to transfer current weights (Section 4.2.3) to vLLM.
- **Inference Phase:** Following Megatron's offloading, vLLM starts with dummy model weights and updates them with the latest ones transferred from Megatron via Mooncake [**?**]. After updating the weights, vLLM deploys the trained model to generate messages, which is called rollout. Upon completion of the rollout, the checkpoint-engine halts all vLLM processes, ensuring an orderly shutdown of inference functionalities. Concurrently, data generated from rollout will be collected for the subsequent training phase.

- **Subsequent Training Phase:** Once the memory allocated to vLLM is released, Megatron onloads the memory and initiates another round of training. This illustrates how resources are efficiently recycled between training and inference phases. This cycle enables iterative improvements in the model's capabilities through successive training sessions.

We find existing works challenging to simultaneously support all the following characteristics.

- Complex parallelism strategy: Megatron may have different parallel strategy with vLLM. For instance, we may enable Pipeline Parallelism and Expert Parallelism in Megatron while only Tensor Parallelism in vLLM for better throughput. Training weights distributing in several nodes in Megatron could be challenging to be shared with vLLM.
- Minimizing idle GPU resources: Recent releases such as SGLang [**?**] and vLLM have proposed RL support. For training efficiency, GPU reservation might be necessary for these services, which could remain idle for extended periods during the training process especially for On-Policy RL. Conversely, GPUs used for training might also be underutilized. It is more efficient to share the same GPU nodes between training and inference to minimize idle GPU resources.
- Capability of dynamic scaling: In some cases, the inference process may take longer than the training process. A significant acceleration can be achieved by increasing the number of inference nodes while keeping the training process constant. Our system enables the efficient utilization of idle GPU nodes when needed.

We implement this hybrid deployment framework (Section 4.2.1) on top of Megatron and vLLM, achieving less than one minute from training to inference phase and about ten seconds conversely.

4.2.1 Hybrid Deployment Strategy

In the pursuit of minimizing idle GPU resources, we propose a hybrid deployment strategy for training and inference tasks. This strategy leverages Kubernetes Sidecar containers to collocate both workloads in one pod on the same machine, enabling both containers to share all available GPUs. This approach facilitates the deployment of training and inference with entirely distinct images, thereby fostering a more decoupled architecture. The primary advantages of this strategy are:

- It facilitates efficient resource sharing and management, preventing train nodes idling while waiting for inference nodes when both are deployed on separate nodes.
- Leveraging distinct deployed images, training and inference can each iterate independently for better performance. This also minimizes the shared design space, thereby avoiding influences from mutual dependencies.
- The architecture is not limited to vLLM, other frameworks can be conveniently integrated, enhancing its versatility.

4.2.2 Checkpoint Engine

Each kubernetes pod is equipped with an agent known as Checkpoint Engine, which is responsible for managing the lifecycle of the vLLM process. Checkpoint Engine exposes HTTP APIs that enable triggering various operations on vLLM, including starting, killing, transferring weights and updating vLLM. In order to ensure the overall consistency and reliability of the entire cluster, we utilize a global metadata system managed by the etcd service to broadcast operations and their associated statuses of each node within the cluster.

It could be challenging to entirely release GPU memory by vLLM offloading primarily due to CUDA graphs, NCCL buffers and NVIDIA driver. To minimize modifications to vLLM, we terminate the vLLM process and restart it at the beginning of a rollout. The cost associated with this restart can be overlapped with the time taken to transfer weights. In certain scenarios, such as when a GPU XID error occurs, vLLM may exit unexpectedly. Checkpoint Engine is designed to automatically restart vLLM to increase fault tolerance, thereby minimizing downtime to a brief period of service unavailability.

4.2.3 Weight Transfer

Since we terminate the vLLM process to release GPU memory, we are unable to leverage NCCL to transfer model weights between vLLM and Megatron. Moreover, conducting communication layer by layer in Megatron when saving checkpoints to avoid CUDA out-of-memory (OOM) errors can be time-consuming. As an alternative, the worker in Megatron converts the owned checkpoints into the Hugging Face format and saves them in shared memory. This conversion also takes Pipeline Parallelism and Expert Parallelism into account so that only Tensor Parallelism remains in these checkpoints. Checkpoints in shared memory are subsequently divided into shards and registered in the global metadata system. We employ Mooncake to transfer checkpoints between peer nodes over RDMA. This peer-to-peer networking architecture enhances the efficiency of large-scale data distribution and alleviates the bandwidth pressure on checkpoint producers.

We have enhanced vLLM to support weight updates. Specifically, vLLM reads weight files and performs tensor parallelism conversion. It is not necessary for Megatron and vLLM to employ the same parallelism strategy. To update the model, we simply replace the old weight data in CUDA memory with the new weight data.

5 Experiments

5.1 Main Results

@zhuhan @dehao @zhiqi @haoyu

	Benchmark (Metric)	**Qwen2.5 72B-Inst.**	**LLaMA-3.1 405B-Inst.**	**Claude-3.5-Sonnet-1022**	**GPT-4o 0513**	**DeepSeek V3**	**MiniMax Text-01**	**Kimi k1.5**
English	**MMLU** (EM)	85.3	**88.6**	**88.3**	87.2	**88.5**		
	DROP (3-shot F1)	76.7	88.7	88.3	83.7	**91.0**		
	IF-Eval (Prompt Strict)	84.1	86.0	**86.5**	84.3	86.1		
	GPQA-Diamond (Pass@1)	49.0	51.1	**65.0**	49.9	59.1		
	LongBench v2 (Acc.)	39.4	36.1	41.0	48.1	**48.7**		
Code	HumanEval-Mul (Pass@1)	77.3	77.2	81.7	80.5	**82.6**		
	HumanEval (Pass@1)							
	LiveCodeBench (Pass@1-COT)	31.1	28.4	36.3	33.4	**40.5**		
Math	**AIME 2024** (Pass@1)	23.3	23.3	16.0	9.3	**39.2**		
	MATH-500 (EM)	80.0	73.8	78.3	74.6	**90.2**		
Chinese	**CLUEWSC** (EM)	**91.4**	84.7	85.4	87.9	90.9		
	C-Eval (EM)	86.1	61.5	76.7	76.0	**86.5**		

Table 2: Main short text perf. Comparing with flagship opensource and proprietary models, including DeepSeek-V3, Qwen2.5 72B-Instruct, LLaMa-3.1 405B-Instruct, claude, openai-4o. Must report in **Bold**

	Benchmark (Metric)	**OpenAI o1**	**QwQ-32B preview**	**DeepSeek-R1 Lite-Preview**	**Kimi k1.5**
Code	HumanEval-Mul (Pass@1)				
	LiveCodeBench (Pass@1-COT)				
	LiveCodeBench (Pass@1)				
	Codeforces (Percentile)				
	SWE Verified (Resolved)				
	Aider-Edit (Acc.)				
	Aider-Polyglot (Acc.)				
Math	AIME 2024 (Pass@1)				
	MATH-500 (EM)				
	CNMO 2024 (Pass@1)				
Chinese	CLUEWSC (EM)				
	C-Eval (EM)				
	C-SimpleQA (Correct)				

Table 3: Main long perf.

5.2 Scaling Properties of RL

@zhuhan @dehao

needs a couple of graphs for RL scaling law for both training and test on multiple datasets (other better axes to consider?):

- train acc vs training flops
- length vs training flops
- test acc vs inference flops
- length vs inference flops

5.3 Ablation Studies

@chenjun @zhuhan @shupeng

6 Conclusions

References

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. Starcoder: may the source be with you!, 2023. URL `https://arxiv.org/abs/2305.06161`.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. Starcoder 2 and the stack v2: The next generation, 2024. URL `https://arxiv.org/abs/2402.19173`.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models. *arXiv preprint arXiv:2309.00071*, 2023.

Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al. Laion-5b: An open large-scale dataset for training next generation image-text models. *Advances in Neural Information Processing Systems*, 35:25278–25294, 2022.

Samir Yitzhak Gadre, Gabriel Ilharco, Alex Fang, Jonathan Hayase, Georgios Smyrnis, Thao Nguyen, Ryan Marten, Mitchell Wortsman, Dhruba Ghosh, Jieyu Zhang, et al. Datacomp: In search of the next generation of multimodal datasets. *Advances in Neural Information Processing Systems*, 36, 2024.

Wanrong Zhu, Jack Hessel, Anas Awadalla, Samir Yitzhak Gadre, Jesse Dodge, Alex Fang, Youngjae Yu, Ludwig Schmidt, William Yang Wang, and Yejin Choi. Multimodal c4: An open, billion-scale corpus of images interleaved with text. *Advances in Neural Information Processing Systems*, 36, 2024.

Hugo Laurençon, Lucile Saulnier, Léo Tronchon, Stas Bekman, Amanpreet Singh, Anton Lozhkov, Thomas Wang, Siddharth Karamcheti, Alexander Rush, Douwe Kiela, et al. Obelics: An open web-scale filtered dataset of interleaved image-text documents. *Advances in Neural Information Processing Systems*, 36, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*, 2024.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction, 2024. *URL https://arxiv. org/abs/2408.15240*, 2024.

Yasin Abbasi-Yadkori, Peter Bartlett, Kush Bhatia, Nevena Lazic, Csaba Szepesvari, and Gellért Weisz. Politex: Regret bounds for policy iteration using expert prediction. In *International Conference on Machine Learning*, pages 3692–3702. PMLR, 2019.

Jincheng Mei, Chenjun Xiao, Ruitong Huang, Dale Schuurmans, and Martin Müller. On principled entropy exploration in policy optimization. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 3130–3136, 2019.

Manan Tomar, Lior Shani, Yonathan Efroni, and Mohammad Ghavamzadeh. Mirror descent policy optimization. *arXiv preprint arXiv:2005.09814*, 2020.

Wouter Kool, Herke van Hoof, and Max Welling. Buy 4 reinforce samples, get a baseline for free! 2019.

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. *arXiv preprint arXiv:2402.14740*, 2024.

Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Bridging the gap between value and policy based reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, Arjun Guha, Michael Greenberg, and Abhinav Jangda. Multipl-e: A scalable and polyglot approach to benchmarking neural code generation. *IEEE Transactions on Software Engineering*, 49(7):3675–3691, 2023. doi:10.1109/TSE.2023.3267446.

Appendix

A Contributions

Angang Du
Bofei Gao
Changjiu Jiang
Chenjun Xiao
Chenzhuang Du
Congcong Wang
Dehao Zhang
Enming Yuan
Enzhe Lu
Flood Sung
Guokun Lai
Haiqing Guo
Han Zhu
Hao Ding
Hao Hu
Hao Yang
Hao Zhang
Haotian Yao
Haotian Zhao
Haoyu Lu
Hongcheng Gao
Huan Yuan
Huabin Zheng
Jingyuan Liu
Jianlin Su
Jianzhou Wang
Jin Zhang
Junjie Yan
Lidong Shi
Longhui Yu
Mengnan Dong
Shupeng Wei
Shaowei Liu
Sihan Cao
Tao Jiang
Weimin Xiong
Weiran He
Weihao Gao
Weixiao Huang
Wenhao Wu
Wenyang He
Xianqing Jia
Xinran Xu
Xinyu Zhou
Xinxing Zu
Yang Li
Yangyang Hu
Yangyang Liu
Yanru Chen
Yejie Wang
Yidao Qin
Yibo Liu
Yiping Bao
Yulun Du
Yuzhi Wang
Y. Charles
Zaida Zhou
Zhaoji Wang
Zhaowei Li
Zhexu Wang
Zhiqi Huang
Zhilin Yang
Ziyao Xu
Zonghan Yang
Zheng Zhang

The listing of authors is in alphabetical order based on their first names.

B Qualitative Examples

B.1 Math

B.2 Code

B.3 K12

B.4 Visual Reasoning