arXiv:2507.18013v3 [cs.CL] 29 Jul 2025

TECHNICAL REPORT OF TELECHAT2, TELECHAT2.5 AND T1

Zihan Wang*, **Xinzhang Liu***, **Yitong Yao***, **Chao Wang***, **Yu Zhao***, **Zhihao Yang***, **Wenmin Deng***, **Kaipeng Jia**, **Jiaxin Peng**, **Yuyao Huang**, **Sishi Xiong**, **Zhuo Jiang**, **Kaidong Yu**, **Xiaohui Hu**, **Fubei Yao**, **Ruiyu Fang**, **Zhuoru Jiang**, **Ruiting Song**, **Qiyi Xie**, **Rui Xue**, **Xuewei He**, **Yanlei Xue**, **Zhu Yuan**, **Zhaoxi Zhang**, **Zilu Huang**, **Shiquan Wang**, **Xin Wang**, **Hanming Wu**, **Mingyuan Wang**, **Xufeng Zhan**, **Yuhan Sun**, **Zhaohu Xing**, **Yuhao Jiang**, **Bingkai Yang**, **Shuangyong Song**, **Yongxiang Li**, **Zhongjiang He**†, **Xuelong Li**†

TeleAI

ABSTRACT

We introduce the latest series of TeleChat models: **TeleChat2**, **TeleChat2.5**, and **T1**, offering a significant upgrade over their predecessor, TeleChat. Despite minimal changes to the model architecture, the new series achieves substantial performance gains through enhanced training strategies in both pre-training and post-training stages. The series begins with **TeleChat2**, which undergoes pretraining on 10 trillion high-quality and diverse tokens. This is followed by Supervised Fine-Tuning (SFT) and Direct Preference Optimization (DPO) to further enhance its capabilities. **TeleChat2.5** and **T1** expand the pipeline by incorporating a continual pretraining phase with domain-specific datasets, combined with reinforcement learning (RL) to improve performance in code generation and mathematical reasoning tasks. The **T1** variant is designed for complex reasoning, supporting long Chain-of-Thought (CoT) reasoning and demonstrating substantial improvements in mathematics and coding. In contrast, **TeleChat2.5** prioritizes speed, delivering rapid inference. Both flagship models of **T1** and **TeleChat2.5** are dense Transformer-based architectures with 115B parameters, showcasing significant advancements in reasoning and general task performance compared to the original TeleChat. Notably, **T1-115B** outperform proprietary models such as OpenAI's o1-mini and GPT-4o. We publicly release **TeleChat2**, **TeleChat2.5** and **T1**, including post-trained versions with 35B and 115B parameters, to empower developers and researchers with state-of-the-art language models tailored for diverse applications.

Model	**Link**
TeleChat2-35B	https://modelscope.cn/models/TeleAI/TeleChat2-35B
TeleChat2-115B	https://modelscope.cn/models/TeleAI/TeleChat2-115B
TeleChat2.5-35B	https://modelscope.cn/models/TeleAI/TeleChat2.5-35B
TeleChat2.5-115B	https://modelscope.cn/models/TeleAI/TeleChat2.5-115B
T1-35B	https://modelscope.cn/models/TeleAI/T1-35B
T1-115B	https://modelscope.cn/models/TeleAI/T1-115B

*These authors contributed equally to this work
†Correspondence to hezj@chinatelecom.cn; xuelong_li@chinatelecom.cn

Model Series	Github Link
TeleChat2	`https://github.com/Tele-AI/TeleChat2`
TeleChat2.5	`https://github.com/Tele-AI/TeleChat2.5`
T1	`https://github.com/Tele-AI/T1`

CONTENTS

1 Introduction

In recent years, there has been swift advancement and growth in Large Language Models (LLMs), indicating strides toward Artificial General Intelligence (AGI). Proprietary products such as GPT4 (OpenAI et al., 2024b), Claude (Anthropic, 2024), and Gemini(Google, 2024) demonstrate performance at par with human capabilities, elevating the community's expectations for the potential of open-source LLMs. In addition to proprietary models, several notable open-source LLMs, such as LLaMA series (Touvron et al. (2023a); Touvron et al. (2023b); Grattafiori et al. (2024)), Qwen series(Bai et al. (2023); Yang et al. (2024); Qwen et al. (2025)), Mistral series(Jiang et al. (2023);Jiang et al. (2024)), Deepseek series(DeepSeek-AI et al. (2024a); DeepSeek-AI et al. (2024b)), and our TeleChat series (Wang et al. (2024b)) are also making significant progress, striving to narrow the gap with their proprietary counterparts. The open-weight models have made large language models accessible to developers, enabling wider participation in research, promoting innovation through community collaboration, and speeding up the development of AI applications across various fields. Recent advancements, such as the success of DeepSeek-R1 (DeepSeek-AI et al., 2025), demonstrate the critical role of long Chain-of-Thought (COT) and reinforcement learning (RL) in enhancing the reasoning capabilities of large language models (LLMs). Notable examples, including OpenAI-o1 (OpenAI et al., 2024a), Skywork OR1 (He et al., 2025), Qwen3 (Yang et al., 2025), and Kimi-K1.5 (Team et al., 2025), exemplify how RL can significantly improve performance in complex tasks such as mathematical reasoning and code generation.

To advance open-source contributions, we have enhanced our models and introduced the latest series including **TeleChat2**, **TeleChat2.5**, and **T1**, representing a significant upgrade over their predecessor, TeleChat. The open-weight releases include post-trained variants of 35B and 115B parameter language models. We have made the model parameters publicly available on platforms such as HuggingFace and ModelScope. Additionally, we provide the full codebase on GitHub, which includes comprehensive tools for model fine-tuning, quantization, deployment, and integration with LangChain, enabling a wide range of practical applications.

The development of the new series of TeleChat consists of two main stages: (1) A pre-training stage in which the model is trained on massive datasets by predicting the next word in continuous text. The pre-training process of TeleChat2 is meticulously detailed, highlighting the preparation of diverse data types and data composition adjustment. TeleChat2 efficiently captures long-term dependencies, initially trained on 8K tokens before advancing to 256K tokens in the pre-training stages, exhibiting remarkable performance on long-context benchmarks. (2) Following this, we conduct post-training, including Supervised Fine-Tuning (SFT, Ouyang et al. (2022)) and Direct Preference Optimization (DPO, Rafailov et al. (2023)) on the base model of TeleChat2, to align it with human preferences and further improve specific capabilities. This paper also present our insights into enhancing specific capabilities such as code, reasoning, tool use, and precise instruction following.

The **TeleChat2**, **TeleChat2.5**, and **T1** series were trained on the Atlas 900 A2 cluster, powered by 8,000 Ascend NPUs. Distributed training leverages the 4D parallelism strategy provided by MindSpore's large-model parallel framework[1], enabling efficient scaling for trillion-parameter models. The computational infrastructure was hosted at CTYun's Shanghai Compute Center, which delivered the high-performance resources required for large-scale training.

The new series is developed to enhance the model's ability to understand and generate natural language text, particularly in complex and nuanced contexts. To evaluate their performance in these scenarios, we test **TeleChat2**, **TeleChat2.5**, and **T1** across a comprehensive set of benchmarks spanning mathematics, reasoning, tool usage, precise instruction following, and open-ended tasks. Evaluation results demonstrate that these models achieve significant advancements in reasoning and general task performance compared to there predecessor, TeleChat. Notably, **T1-115B** outperforms proprietary models like `OpenAI's o1-mini` and `GPT-4o`.

- **Better in training data.** We improve both the quality and quantity of the data used for pre-training and post-training. During the pretraining stage, we expand the high-quality pre-training datasets from the previous 3 trillion tokens to 10 trillion tokens, laying a robust groundwork for common sense, expert knowledge, and reasoning capabilities. As for the post-training data, we employ more rigorous quality control and filtering process, and meticulously gather high-quality data to enhance several specific capabilities. Additionally, we conduct data blending experiments to identify the optimal data composition for both pre-training and post-training phases.
- **Better in model size.** We develop the new model series at a significantly larger scale compared to previous iterations of the TeleChat series. Our flagship language models of **TeleChat2**, **TeleChat2.5** and **T1** feature 115 billion trainable parameters. Furthermore, we also introduce 35B variants, providing a more cost-effective solution for resource-constrained scenarios. Given that the new series of models share a uniform model architecture and are trained on the homogeneous source data but in varying sizes, they can collaborate under the AI-Flow framework (Shao & Li, 2024) (An et al., 2025), distributing the workload across multiple models situated in diverse computational nodes, including end devices, edge nodes, and cloud servers. This facilitates a seamless flow of intelligence across networks.
- **Better in real-life applications.** The new model series are trained to significantly extend the model's contextual length beyond that of TeleChat, supporting context window up to 128K tokens. This enhancement is essential for real-life applications such as lengthy conversations, long-distance reasoning and understanding, and other tasks that require the model to consider a substantial amount of context. Additionally, the new series also provides better and easier tool usage, making it more accessible and user-friendly for a wide range of applications.
- **Better in Reasoning & Coding.** The training of **TeleChat2.5** and **T1** incorporates reinforcement learning (RL) to explicitly optimize the models' ability to solve mathematical and coding problems, demonstrating substantial improvements compared to their predecessors when tackling complex problems in these domains.

In the remainder of this paper, we first present the model architecture in Section 2. Next, we describe our pre-training process, including the pre-training recipe, construction of training data, and long context extension techniques in Section 3. Thereafter, we discuss our post-training methodology, including the composition of training data and specific methods during Supervised Fine-Tuning (Section 4.1), Direction preference optimization (Section 4.2) and Reinforcement Learning (Section 4.3). We highlight special efforts to improve performance for specific capabilities such as code, math & reasoning, tool use and precise instruction following in Section 5. We describe our hardware and infrastructure that powered training and discuss several optimizations that leads to improvements in training efficiency in Section 6. We then present the detailed evaluation results in Section 7, covering both the base and chat models.

[1]`https://www.mindspore.cn/`

2 Model Architecture

TeleChat2, **TeleChat2.5**, and **T1** share a unified model architecture, largely retaining the design of their predecessor, TeleChat. This architecture incorporates a Pre-Norm design with RMSNorm normalization (Zhang & Sennrich, 2019), employs SwiGLU (Shazeer, 2020) as the activation function for the Feed-Forward Network (FFN), and integrates Rotary Positional Embeddings (Su et al., 2022). Detailed network specifications can be found in Table 1. There are several minor adjustments compared to TeleChat, which are detailed below:

- **Grouped Query Attention (GQA).** We use Grouped Query Attention with 8 key-value heads instead of the traditional Multi-Head Attention(MHA) for models with 115 billion parameters, achieving both accelerated training and efficient KV cache utilization.
- **RoPE base frenquency.** By increasing the RoPE base frequency hyperparameter, we improve our capacity to handle longer contexts more effectively. See Section 3.3 for Details.

$Params$	n_{layers}	d_{model}	d_{ffn}	n_{heads}	n_{kv_heads}	n_{vocab}
35B	64	6144	20480	48	48	131072
115B	96	8192	40960	64	8	131072

Table 1: Detailed model architecture hyperparamters of TeleChat2, TeleChat2.5 and T1 model family.

3 Pre-training

3.1 Overall Pre-training Recipe

Our training process for TeleBase2 comprises two main stages. First, during the Initial Pre-training Stage (Section 3.2), we curate high-quality training data using filtering and data mixture, resulting in a total of 10 trillion tokens. In this stage, the model acquires foundational language structures and accumulates extensive world knowledge from textual data. Second, during the Long-Context Annealing Stage (Section 3.3), we refine the model's capabilities through curated and synthetic datasets, particularly enhancing performance on reasoning and knowledge-based tasks. Simultaneously, we extend the model's context length to 256K tokens. In subsequent sections, we will elaborate on these stages from both data composition and training methodology perspectives. The pre-training framework is illustrated in Figure 1.

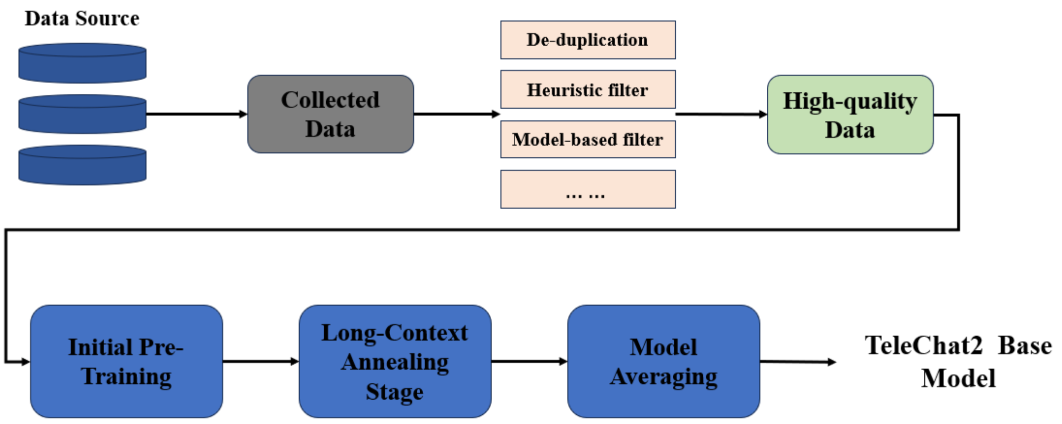

Figure 1: The pre-training framework.

3.2 Initial Pre-training Stage

In the initial pre-training phase, our primary goal is to equip the model with broad and comprehensive world knowledge. To achieve this, we train the model on an extensive, high-quality and diverse corpus. The pretraining dataset is meticulously curated and filtered to ensure linguistic richness, topical diversity, and coverage across languages and writing styles. The model is trained on 10 trillion tokens, enabling it to develop a robust understanding of language structures, factual knowledge, and common-sense reasoning, which lays a solid foundation for subsequent finetuning and domain-specific adaptation.

3.2.1 Data Collection

Compared to its predecessor, TeleBase2 utilizes a more extensive and higher-quality training dataset during pre-training. We build the pre-training corpus by aggregating diverse, high-quality data from multiple sources to create a robust knowledge foundation. The dataset comprises general-domain and domain-specific content. General-domain sources include web pages, books, encyclopedias, news articles, social media platforms, academic papers, code repositories, and other resources. Domain-specific data is curated from over twenty specialized industries, such as finance, construction, healthcare, and other technical fields.

3.2.2 Data Cleaning

Data cleaning is critical to improving model performance by ensuring the quality, consistency, and relevance of training data. We implement a suite of data cleaning and quality assurance techniques, which are outlined below:

De-duplication. We implement a hierarchical de-duplication strategy comprising *URL-level*, *document-level*, and *paragraph-level* de-duplication. Following a similar approach to TeleChat (Wang et al., 2024b), this multi-tiered framework ensures the removal of redundant data while preserving the diversity and quality of the training corpus.

Heuristic filtering. We devise several heuristic approaches to enhance the overall quality of the data. Some of the heuristic rules are listed as follows. (1) We exclude texts that are exceedingly brief or lack of substantial informational content. (2) We filter out texts exhibiting an anomalously high or low frequency of punctuation marks. (3) Texts containing an excessive number of dirty words are excluded from the dataset. (4) The code-related data is processed using evaluation criteria specific to the source website. For instance, GitHub project code with a low number of repository stars are excluded from the dataset.

Model-based quality filtering. We integrate large language models (LLMs) into our data filtering pipeline to strengthen quality control. After an initial automated filtering step, we deploy LLMs to conduct in-depth semantic analysis. These models evaluate the text's relevance, coherence, and fluency while identifying and flagging potentially toxic, biased, or inappropriate content. Additionally, they detect nuanced issues such as logical inconsistencies, off-topic segments, and unnatural language patterns that might not be captured by rule-based systems.

Math and Code Data Cleaning. For mathematical and code-related data, we prioritize correctness and executability during quality assurance. To ensure syntactic correctness, we employ automated scripts and static analysis tools to filter out erroneous data. Code samples are then verified using code execution feedback, while mathematical problems are verified using symbolic computation tools to confirm the accuracy of equations and solutions. We also integrate large language models (LLMs) into our validation workflow. Specifically, LLMs are tasked with reviewing code logic, identifying potential errors, and generating expected outputs for comparison with original data. For mathematical content, models independently solve problems and cross-reference their results with provided solutions. This hybrid approach enables efficient detection of subtle errors that traditional rule-based methods might overlook. Finally, a subset of the data undergoes manual review by human experts to ensure clarity, completeness, and relevance. This includes verifying that code samples are self-contained and well-documented, while mathematical content adheres to standard notation and formatting conventions.

3.2.3 Determine data composition

Data composition has a significant impact on model performance. However, for very large models like TeleBase2-115B, it is not feasible to do extensive data composition tuning. To address this problem, we conduct a series of experiments on smaller models (3B and 7B) to evaluate the effect of data mix on model performance. For example, we test varying proportions of Chinese and English corpora in training and observe that the English corpus proportion should not be excessively reduced. This finding can be attributed to two factors: (1) the inherently higher linguistic complexity of Chinese, and (2) the generally lower quality of Chinese corpora compared to English. Based on these insights, we predict the performance of larger models under different data compositions and select the most promising mix for scaling up training.

During model training, we adopt a curriculum learning strategy to dynamically adjust the proportions of different data types. In the initial training phases, we emphasize simpler and more general data to help the model establish a strong foundation in language understanding and basic reasoning. As training progressed, we gradually increase the proportion of more complex and specialized data, such as mathematical problems and code-related tasks, allowing the model to incrementally build advanced capabilities. To ensure balanced learning, we conduct comprehensive evaluations every 100 billion tokens using a diverse set of benchmarks or validation set covering all major data types. Based on the evaluation results, we adjust the data sampling ratios in the subsequent training stages, increasing the representation of data types where the model shows relative weakness. This dynamic adjustment process enables the model to maintain steady improvements across all domains, resulting in a more robust and versatile language model.

3.2.4 Training Details

We employ the Adam optimizer to pre-train TeleBase2, with the following hyperparameter settings: $\beta_1 = 0.9, \beta_2 = 0.95, \epsilon = 1 \times 10^{-8}$. Cosine learning rate scheduler is used to regulate the learning rate, with the peak learning rate scaled according to the model size. After reaching its maximum value following the warm-up steps, the learning rate gradually decays to 10% of the peak rate. Weight decay with a factor of 0.01 is applied to all model parameters except for bias terms. Gradient clipping is enforced with a norm of 1.0. All learnable parameters are initialized using a normal distribution with a standard deviation of 0.006. Further hyperparameter configurations are detailed in Table 2. Following the methodology of TeleChat1, we concatenate data from the same source without applying cross-sample attention masking. We set the maximum sequence length to 8K during the first-stage pre-training, and pre-train TeleBase2 on 10T tokens.

HyperParams	**TeleBase2-35B**	**TeleBase2-115B**
Peak lr	3×10^{-4}	2×10^{-4}
Batch tokens	8M	6M
Warm-up fraction	0.001	0.001
Rms Norm Epsilon	1×10^{-5}	1×10^{-5}

Table 2: The hyperparameter details during the pretraining stage of TeleBase2-35B and TeleBase2-115B.

3.3 Long-Context Annealing Stage

To optimize the balance between training efficiency and effectiveness, we integrate long-context extension during the annealing phase. This approach introduces a unified training stage designed to simultaneously improve both general capabilities and long-context understanding in the base model. Specifically, we extend the context window to 256K tokens for TeleBase2-35B and 128K tokens for TeleBase2-115B, while maintaining general capability parity with their 8K-token counterparts. This integration ensures that the model retains strong foundational skills while adapting to extended context requirements.

3.3.1 Data Curation

The training data is categorized into five distinct length intervals: 0–8K, 8K–16K, 16K–32K, 32K–128K, and 128K+. Within each interval, the data is further subdivided by domain (e.g., exams, web pages, code, and other categories) to enable fine-grained analysis. During the annealing phase, the 0–8K interval is combined with other intervals at a 7:3 ratio, prioritizing shorter sequences while gradually introducing longer contexts. Simultaneously, high-quality data from important domains (e.g., exams and code) is upsampled across all length intervals, ensuring robust coverage of critical knowledge sources. This structured approach aligns with principles of data engineering for long-context training (Fu et al., 2025).

3.3.2 Training Details

The context length of the pre-trained model is extended sequentially in stages, with the learning rate decreasing successively according to cosine annealing. The initial learning rate for the first annealing stage is equivalent to the learning rate employed during 8K pre-training. Subsequent annealing progresses based on the weights from the 1/3 steps of the preceding training stage, with the learning rate at that time serving as the initial value. As the base in Rotary Position Encoding (RoPE) is a pivotal factor in determining the effective context length of a LLM (Liu et al., 2024b); (Xu et al., 2024), we set the RoPE's base to 1×10^6 for 32K annealing, 8×10^6 for 128K and4×10^7 for 256K. Moreover, 50B training tokens are sufficient for each complete annealing phase. After multiple stages of context extension annealing and fine-tuning, the TeleBase2 famliy of models perform well on the "Needle In A Haystack" (NIAH) test over 4K to 128K context lengths. Figure 2 illustrates the evaluation results of TeleBase2-115B.

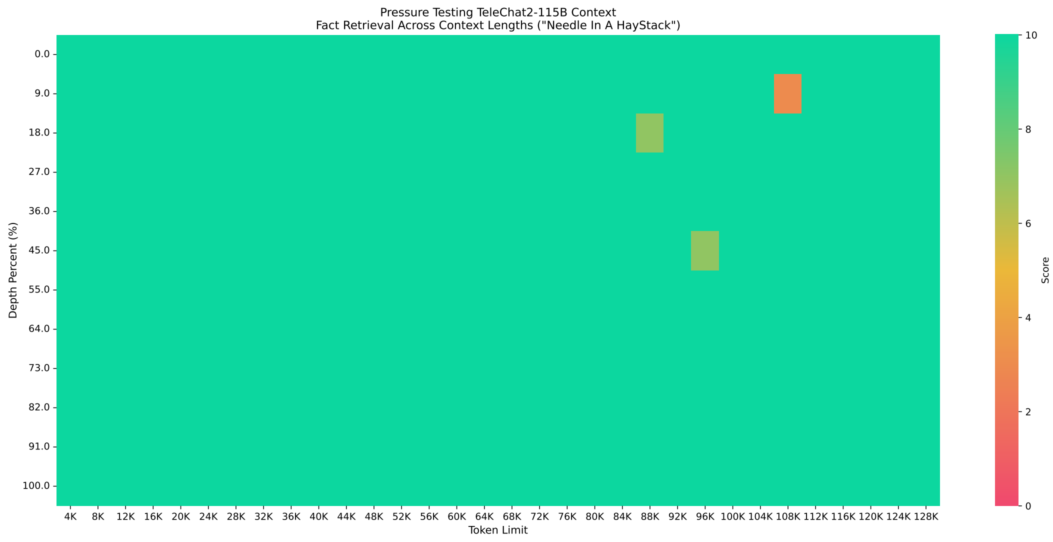

Figure 2: Evaluation results of TeleBase2-115B on the "Needle In A Haystack" test.

3.4 Model Averaging

To enhance the robustness and generalization of the final model, we apply checkpoint averaging after the training process. Specifically, we compute the element-wise average of parameters of the last five checkpoints. By averaging these checkpoints, we effectively smooth the parameter distribution and improve model stability.

3.5 Tokenizer

For our tokenizer, we implement BPE with byte-level fallback in SentencePiece, splitting numbers into individual digits as in the approach described by (Touvron et al., 2023b). We augment the final vocabulary with special tokens to differentiate dialogue roles and to support tool functionality. To ensure computational efficiency during training and to reserve space for any additional special tokens that might be needed in the future, we configure the model's vocabulary size to 131072. We establish a unified vocabulary across all **TeleChat2**, **TeleChat2.5** and **T1** model family, enhancing consistency and reducing potential compatibility issues.

4 Post-Training

As illustrated in Figure 3, the **TeleChat2** model is trained directly on the base model through a supervised fine-tuning (SFT) stage and a direct preference optimization (DPO) stage to enhance its general capabilities. On the other hand, the **TeleChat2.5** and **T1** models first undergo a continual pretraining stage, followed by a three-stage post-training process. This process comprises: (1) an SFT stage with both thinking and non-thinking modes, (2) a DPO stage to improve general capabilities, and (3) a reinforcement learning (RL) stage to strengthen math and coding abilities. This pipeline yields **T1** (thinking variant) and **TeleChat2.5** (non-thinking variant).

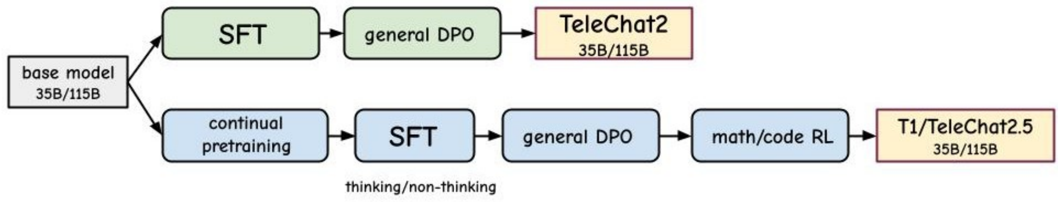

Figure 3: The development pipelines of TeleChat2, TeleChat2.5 and T1.

4.1 Supervised Fine-Tuning

We finetune the pretrained model using high-quality, domain-diverse data including mathematics, coding, reasoning, conversation, model identity, safety, etc. To create high-quality SFT data, we develop a two-stage pipeline comprising (1) diverse query collection and (2) response generation with quality verification.

4.1.1 Query Collection

To systematically organize and classify our SFT data, we develop a tagging system that categorizes prompts by domain and discipline, ensuring balanced representation across diverse subject areas. This hierarchical system includes major categories such as mathematics, coding, reasoning, conversational safety, instruction following, tool use, and more. Each category is further subdivided into granular classifications to comprehensively capture the required capabilities.

Sourcing from Public Datasets. We source queries from a wide array of open-source datasets and employ rigorous data-cleaning processes to eliminate duplicates or highly similar entries. To identify semantic relationships, we map the queries into a high-dimensional embedding space and applied the K-means clustering algorithm to group them effectively.

Enhancing Dataset Diversity and Balance. After cleaning and organizing the data within our tagging system, we identify gaps in certain categories and observe uneven task difficulty distributions. To address these challenges, we utilize self-instruct and instruct-evolution techniques to generate synthetic queries. These methods allow us to construct a query set that not only fully covers knowledge system but also achieves a well-balanced distribution of complexity and diversity. Specifically, we design separate difficulty-scoring prompts for different data categories and leverage LLMs to individually score each data type within every source. For domains such as mathematics and code, we employ a pass rate metric to distinguish the learning difficulty. Regarding certain data types (e.g., creative writing, role-playing, instruction-following, and structured data generation), we observe generally low difficulty levels in open-source datasets. To address this, we manually curate high-quality seed examples and reconstruct datasets through instruction evolution. This approach ensures the data difficulty closely aligns with real-world usage complexity.

Query Quality Scoring. To guarantee query quality, we implement LLM-based scoring mechanisms. Queries are evaluated against predefined criteria including fluency, standardized formatting, and completeness of contextual information necessary for generating robust responses. Low-scoring queries are either excluded or down-weighted in training to prioritize high-quality data. After thorough review and refinement, we curate a dataset with broad coverage and a well-calibrated difficulty range.

4.1.2 Response Generation and Quality Control

We employ both human annotation and synthetic data generation to generate responses.

Human annotation collection. We assemble a team of internal annotators and external contractors to perform manual data annotation. Our annotators possess diverse expertise across a wide range of disciplines. To address queries that challenge current large language models (LLMs), particularly in math and reasoning tasks, we rely on our annotation team to generate high-quality responses. For non-reasoning tasks such as creative writing, role-play, and open-ended QA, we engage human annotators to validate the accuracy of synthetic data.

Synthetic data generation. For the collected queries, we first generate responses using high-performance models and then select the optimal answer based on task-specific evaluation criteria. Specifically, for tasks with verifiable correctness (e.g., mathematics, code generation, instruction-following, STEM exams), we employ rule-based reward systems to evaluate responses through predefined metrics, and only correct answers are retained. For subjective tasks (e.g., humanities, creative writing, open-ended QA), we utilize LLM-as-judge frameworks, where independent large language models score responses based on fluency, coherence, and relevance. Only the highest-scoring response is retained.

We implement a comprehensive suite of rule-based data verification mechanisms to further ensure data accuracy. The primary rules are listed as follows. (1) During generation, problems including duplicate content, truncated outputs, and illegible characters frequently occur. We strictly filter out such erroneous data. (2) We enforce constraint compliance through rule-based validation scripts, ensuring adherence to format-specific requirements like output length, paragraph count, or structural guidelines imposed by user queries. (3) We implement a content filter using a sensitive keyword database to filter answers potentially containing safety risks. The flagged data is then executed further validation by human annotators for quality assurance.

4.1.3 Determine Data Mix

The composition of post-training data critically influences the behavior of language models. To optimize performance, we employ an iterative algorithm that upsamples high-quality data sources while downsampling lower-quality ones in the final data mix. Our analysis reveals a potential negative correlation between model performance and perplexity on the validation set $\mathcal{V}$, which is created by extracting 1% of the data from our training set $\mathcal{T}$. Specifically, models achieving the better evaluation performance typically exhibit the lower perplexity on validation set. However, when the validation set is partitioned by category, not all subsets reach their minimum perplexity at the same training steps. To address this problem, we designed an algorithm that iteratively adjusts the representation ratio r_i of each category subset i within the overall training data, where $i \in \mathbb{N}^*, 1 \leq i \leq |\mathcal{V}|$.

During the t-th round fine-tuning experiment, we divide the training data into various subsets by their classifications and record the perplexity of each of them at regular training intervals. We set the maximum iteration count to T for termination assurance, $t \in \mathbb{N}, 0 \leq t < T$. Next, we use cubic spline interpolation to fit a curve $p = f_i^{(t)}(s)$, representing the perplexity p of the subset i as a function of the training step s in iteration t. Denote the lowest point of this curve as $(s_i^{(t)}, p_i^{(t)})$. Similarly, we compute the weighted average of the perplexities according to the tokens of each subset and fit a curve whose lowest point is denoted as $(\bar{s}^{(t)}, \bar{p}^{(t)})$.

The new proportion can be calculated as follows.

$$r_i^{(t+1)} = r_i^{(t)} \kappa^{\frac{s_i^t - \bar{s}^{(t)}}{\mu}}, \tag{1}$$

where κ and μ are hyper-parameters dynamically calibrated based on dataset characteristics, with optimal values of 10 and 15,000 respectively in our experiment.

$$\hat{r}_i^{(t+1)} = \frac{r_i^{(t+1)}}{\sum_{i=1}^{|\mathcal{V}|} r_i^{(t+1)}}. \tag{2}$$

After regularization, we apply $\hat{r}_i^{(t+1)}$ as the new proportion in the next round of experiment. The data distribution of supervised finetuning data is demonstrated in Figure 4.

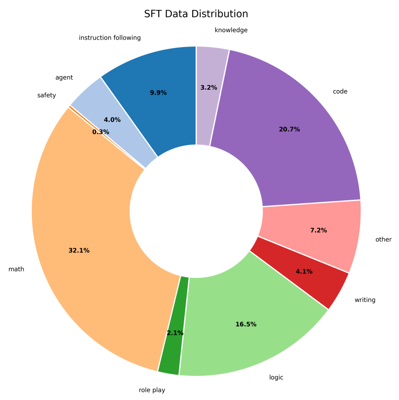

Figure 4: Data Distribution of Supervised Finetuning data.

4.1.4 Training Details

We optimize hyperparameters for fine-tuning through grid search, achieving model-specific training configurations. For the 35B variant, the cosine decay learning rate scheduling starts at 3×10^{-5} and gradually decays to 1×10^{-5} with a batch size of 8; for the 115B variant, the learning rate starts at 1.5×10^{-5} and decays to 1.5×10^{-6} with a batch size of 16. To enhance training efficiency and reduce sequence padding overhead, we implement a packing strategy that concatenates multiple training samples into a single sequence, while strategically combining single-turn samples into multi-turn dialogues whenever possible, enhancing the model's multi-turn conversational capability.

4.2 Direct Preference Optimization

Direct Preference Optimization (DPO, (Rafailov et al., 2023)) is an offline training algorithm designed for learning from preference feedback. It enables direct optimization of the policy using preference data, eliminating the need to construct reward models or sample online from the active policy. The primary objective of DPO is to maximize the margin between the log-likelihood of selected responses and the log-likelihood of rejected responses, while ensuring that the model remains close to the initial policy (Ivison et al., 2024). We apply DPO to align the model with human preferences in general tasks by leveraging pairs of accepted and rejected outputs to discourage undesirable behaviors. In this section, we detail our preference data construction pipeline in section 4.2.1, and our training details in section 4.2.2.

4.2.1 Preference Data Curation

The preference data is of utmost importance in improving in the generation quality and performance of large language models. A preference dataset P typically comprises prompts, responses, and rankings. Each prompt x is associated with a pair of response y^+, y^- (where y^+ is the chosen response and y^- is the rejected response), along with a preference ranking between them (indicated as $y^+ \succ y^-|x$). Our process for creating preference data involves four stages: prompt selection, generating responses from a pool of models, annotating preferences using LLM-as-a-judge, and constructing pairs. The specifics of this process are outlined below.

Prompt Selection. The first step in preparing a dataset for Direct Preference Optimization (DPO) involves the selection of prompts or user instructions for generating responses and gathering preferences. The quality and diversity of the prompt set are essential for ensuring the effectiveness of DPO. Specifically, a high-quality prompt set should fulfill two key criteria: (1) It should demonstrate diversity and cover a wide range of domains, including math & reasoning, coding, creative writing, and more, to enhance the model's adaptability and enable it to address a variety of real-world challenges and inquiries. (2) It should encompass a varied mix of easy, moderate, and challenging questions to promote a comprehensive understanding and reduce the risk of overfitting to specific difficulty levels.

We divide the SFT prompts into two parts, allocating 90% for SFT training and 10% for DPO. As our SFT prompts offer a well-rounded exposure to diverse domains and different complexity levels, our DPO prompts are able to meet the aforementioned criteria. Additionally, we integrate new instruction-following constraints to enhance the model's ability to adhere to instructions, and also introduce pairs based on the badcases of the previous model to address its weaknesses.

Response Generation. When given a prompt, we start by sampling from a pool of state-of-the-art open-source and proprietary models, which differ in parameter size and model family. We use greedy sampling and only sample once for each model. Next, we incorporate on-policy data by sampling completions from the latest **TeleChat2.5** and **T1** models, utilizing high-temperature sampling to produce multiple responses. To improve the efficiency of rejection sampling, we employ vllm (Kwon et al., 2023) to speed up the inference process.

Preference Annotation. After generating multiple responses for each prompt, it is necessary to assign a reward to each response. (1) For verifiable problems, the reward is determined based on specific criteria or rules. For instance, in coding problems, we evaluate if the solution passes the unit test. In math, reasoning, and standard exam problems, we assess if the generated answer leads to the correct solution. For instruction-following constrained prompts, we verify if the generated answer adheres to the constraints. (2) For open-ended problems with free-form answers, we use an LLM-as-a-judge (Zheng et al., 2023) to assess every answer on a scale of 0 to 10 based on four distinct factors: usefulness, adherence to instructions, integrity, and accuracy.

Preference Pair Construction. The construction of preference pairs follows several key principles.

- **Chosen Responses** are exclusively selected from highest-scoring responses. To maintain response quality standards, we impose a minimum threshold of score ≥ 8 for chosen response eligibility. When multiple responses achieve identical maximum scores, priority is given to responses generated by TeleChat series itself rather than off-policy candidates. This design choice mitigates potential distribution shift issues inherent in DPO training, as demonstrated in previous work (Rafailov et al., 2023).
- **Rejected responses** are strictly sampled from TeleChat series model's own generations. This approach allows the model to self-correct by learning from its own error patterns.

A minimum absolute score difference ($\Delta \geq 2$) is enforced between chosen and rejected pairs. This threshold accounts for the documented instability of LLM-as-a-judge scoring, effectively filtering out ambiguous comparisons where minor score variations may not reflect genuine quality differences. For input prompts that generate multiple valid pairs, we randomly sample $K = 4$ distinct pairs per input prompt. This results in $98,273$ pairs for DPO training.

4.2.2 Training Details

We train an epoch for DPO, with a learning rate of 5×10^{-7} and and batch size of 256. We use a learning rate warm-up and cosine learning rate scheduler. The β hyper-parameter is set to be 0.1. We conduct DPO training on our long-context SFT checkpoints, but only select samples with token length shorter than 8,192. Our observation indicates that utilizing only short-context training data in DPO does not negatively impact long-context performance.

During DPO training, we add an additional negative log-likelihood (NLL) loss term for the pair winners with a scaling coefficient of 0.2, which also proves crucial for performance (Pang et al., 2024). Additionally, we employ a technique of masking out the termination tokens from both selected and rejected responses in the loss function to enhance the stability of DPO training, following a similar approach as described in (Grattafiori et al., 2024). This is necessary because the existence of shared tokens in both selected and rejected responses creates a conflicting learning objective, requiring the model to simultaneously increase and decrease the likelihood of these tokens.

4.2.3 Model Merging

we merge models derived from experiments involving different data versions or hyperparameters during DPO stage. In particular, we merge the multiple models by simply averaging their weights, and observe that this merging process is beneficial for enhancing the model's robustness and overall capabilities.

4.2.4 Iterative DPO

The iterative application of the offline preference tuning method procedure has proven beneficial, with the updated model used to construct new preference pairs that are more informative and lead to further improvements. As a result, we apply these methods in three rounds, collecting new preference pairs during each cycle by sampling synthetic data from the latest models.

4.3 Reinforcement Learning

Reinforcement learning (RL) has proven to be effective in enhancing the reasoning capabilities of large language models (LLMs) beyond the Supervised Fine-Tuning (SFT) stage (Shao et al., 2024). In this work, we focus on optimizing the model's performance in both mathematical reasoning and code generation through reinforcement learning strategies.

(1) Mathematical RL. We curate a dataset from two publicly available sources: OpenR1-Math-220k [2] and Synthetic-1 [3]. To ensure data quality, we filter out problems requiring proofs and those with incomplete or inconsistent references. Specifically, we retain only problems that could be automatically verified using the *math_equal* function[4], which checks numerical or analytical equivalence of answers. For answer extraction, we prompt the model to wrap its final answer in *boxed*{} and run the verification process to confirm correctness.

(2) Coding RL. We extract a subset of coding problems from the SFT dataset, and only retain samples that are capable of performing code execution feedback. For unit testing, we develop a secure local code sandbox environment supporting diverse testing methods, including standard input-output validation and assertion-based verification.

(3) Tool use RL. We curate the function-call data for reinforcement learning following a two-step strategy: **(i) Initial Candidate Set Construction.** We select a batch of function call data originating from the same source as the supervised fine-tuning (SFT) data as candidates. Subsequently, multiple large language models (LLMs) are used to perform multiple inferences on each query. Queries where the outputs are consistent across models, along with their corresponding ground-truth answers, are selected as training inputs. **(ii) Difficulty Stratification and Data Curation.** The target model is used to perform multiple inferences on the queries. The model outputs are compared against reference answers to calculate the `pass@5` rate. Queries are categorized into difficulty levels based on `pass@5`:

- `pass@5` $= 1$: These queries are too easy for the current model (Easy).
- $0 <$ `pass@5` < 1: The model has the potential to answer correctly but exhibits unstable performance on these queries (Medium).
- `pass@5` $= 0$: These queries are difficult for the model to answer correctly (Hard).

The RL training dataset is composed of medium and hard data in a 2:1 ratio. For the reward function design, we implement category-specific processing based on data type. Specifically, data is divided into tool-requiring and tool-free categories, with the calculation formula as follows:

$$\text{reward} = \begin{cases} 1, & \text{if } I_{\text{tool}} = 1 \land M_{\text{format}} = 1 \land M_{\text{match}} = 1 \\ -1, & \text{if } I_{\text{tool}} = 1 \land (M_{\text{format}} = 0 \lor M_{\text{match}} = 0) \\ 2 \times \dfrac{S - S_{\min}}{S_{\max} - S_{\min}} - 1, & \text{if } I_{\text{tool}} = 0 \end{cases}$$

Our reward function design distinguishes based on whether a task requires tool calls. For tasks that do require tool calling, we establish a binary reward: if the model's output format is perfectly correct and the specific content of the tool call exactly matches the reference answer, it receives the full reward ($+1$); if the output format is incorrect or the tool call content deviates from the reference answer, a penalty (-1) is given. For pure text tasks that do not require tool calls, we employ a relatively flexible scoring mechanism: First, we use another Large Language Model (LLM) to perform a quality

[2] `https://huggingface.co/datasets/open-r1/OpenR1-Math-220k`

[3] `https://huggingface.co/datasets/PrimeIntellect/verifiable-math-problems`

[4] Available at `https://github.com/hkust-nlp/simpleRL-reason/tree/v0`

assessment on the model's output, resulting in a raw quality score S; Then, we map this raw score onto a unified reward value range of $[-1, 1]$ through a linear transformation formula, in order to enable unified comparison and optimization with the rewards for tool-calling tasks.

We utilize the OpenRLHF [5] framework for training and employ the *reinforce++* algorithm (Hu et al., 2025). To ensure stable training, we implement dynamic sampling, which continues sampling until the batch is fully filled with examples whose accuracy is neither 0 nor 1, as proposed in (Yu et al., 2025). For hyperparameters, we use the AdamW optimizer (Loshchilov & Hutter, 2019) with a constant learning rate of 5×10^{-7}, combined with a linear warm-up over 20 rollout steps. During the rollout phase, the prompt batch size is set to 128, and we generate 16 responses per prompt. For training, the mini-batch size is also configured to 128.

5 Key Abilities

We make special efforts to improve performance for specific capabilities including code, reasoning, tool use, long context and precise instruction following.

5.1 Code

Two-Stage Training Strategy. We implement a coarse-to-fine two-stage fine-tuning approach. In the first stage, the base model is trained on tens of millions of diverse instruction samples synthesized from large-scale open-source datasets (e.g., CodeAlpaca, CodeSearchNet) and code extracted from GitHub repositories. This foundational phase broadens the model's capabilities by exposing it to a wide spectrum of tasks. In the subsequent fine-tuning phase, we employ high-quality, meticulously curated instruction datasets. These include multilingual code generation tasks, programming contests (sourced from Codeforces and LeetCode via web crawling), and programming tutorials. For each query, the LLM generates multiple candidate responses. Verifiable problems are evaluated using code execution feedback, while unverifiable problems leverage the LLM itself to rank and select the most suitable example for supervised fine-tuning.

Code Execution Feedback. For problems that support test case verification, we automatically generate 10 test cases using LLMs. These test cases comprehensively cover normal scenarios, boundary conditions, exceptional cases, and complex inputs to rigorously evaluate correctness. The test cases are categorized by programming language (e.g., Python, C, C++, Java, JavaScript) and executed in a secure sandbox environment. Code correctness is validated through runtime execution verification. Samples failing due to errors in code execution (e.g., invalid syntax or assertion error) are filtered out to ensure training data quality.

Curriculum Learning. We implement a model-driven curriculum learning strategy that leverages the model's own generative capacity to assess prompt difficulty during the second training stage. Specifically, we generate ten responses using a high sampling temperature (e.g., $T = 0.6$) for each prompt. The pass rate (determined by code execution feedback for verifiable tasks) is calculated as a proxy for difficulty, which dynamically construct a training curriculum. Initially, the model focuses on prompts with higher pass rates, ensuring stable learning and foundational skill acquisition. As training progresses, it gradually transits to prompts with lower pass rates, iteratively refining its coding capabilities while systematically expanding its limits.

5.2 Math and Reasoning

Two-Stage Training Strategy. For math and reasoning tasks, we adopt a two-stage fine-tuning strategy consistent with code tasks, transitioning from broad capability construction to in-depth precision optimization. In the first stage, the base model is trained on over ten million synthetic samples sourced from extensive open-source datasets (e.g., StackExchange), synthetic K-12 math problems with answers, and synthetic university instructional materials. All data undergoes source quality assessment, deduplication, format cleaning, synthetic data generation, and quality sampling checks. The second stage employs a smaller yet higher-quality curated dataset. Logical reasoning samples are manually collected with ground-truth answers and cover domains such as causal inference, operations research and game theory. Math data includes high-quality open-source datasets

[5] https://github.com/OpenRLHF/OpenRLHF

(e.g., MATH, GSM8K training sets), licensed K-12 math problems with verified answers, global competition problems (e.g., IMO, AMC), and a small amount of synthetic data to balance distributions. All samples undergo triple verification: problem quality scoring, answer consistency checks, and reasoning process validation. A difficulty-grading mechanism ensures balanced distribution of data across different difficulty levels.

Answer Verification Mechanism. To validate the accuracy of math answers, we implement a multi-model collaborative verification strategy combined with manual supervision for consensus screening. Specifically, for a target set of math problems, we use multiple large models to independently generate answers. A dedicated answer consistency judgment mechanism analyzes and compares the outputs. Samples with complete agreement across all model outputs proceed to manual sampling quality checks, while inconsistent outputs are re-examined through manual annotation to ensure final answer correctness.

5.3 Tool use

Data Curation. We collect mainstream open-source function call datasets (Zhang et al., 2024a) (”interstellarninja”) (Qin et al., 2023) (Toshniwal et al., 2024) (Li et al., 2023a) and perform data cleaning and restructuring. Our validation focuses on two key aspects:

- Format Validation, where we rigorously check the alignment of the tool calls with the provided function list. This involved verifying: 1) the correct correspondence of tool names, 2) the matching of parameter names, and 3) the compliance of parameter types with requirements.
- Tool Call Result Validation, where we utilize a Large Language Model (LLM) to assess the validity of the tool calls and the accuracy of the tool names and parameter configurations.

Furthermore, referencing the methodology used in constructing the BFCL benchmark, we categorize the collected function call data to ensure a balanced distribution of function call types within the training dataset.

Tool Graph based Data Construction. After cleaning open-source data, we collect approximately 110K samples. However, during the cleaning process, we identify issues including insufficient Chinese data, limited conversational turns, and low difficulty levels. To address these challenges, we construct a tool-graph structure based on dependency relationships between APIs, leveraging various graph sampling methods to create tasks with balanced difficulty distribution. Furthermore, we utilize the dependency relationships within the tool-graph to facilitate verification of multi-turn tool-calling accuracy, which demonstrates significant optimization effectiveness.

5.4 Precise instruction following

To improve the model’s instruction following ability, we develop a systematic pipeline for constructing SFT training datasets. In this process, we construct high-quality training data through three key stages: constraint set construction, instruction evolution, and response generation with validation filtering.

Constraint Set Construction. Following IFEval (Zhou et al. (2023)), we identify representative application scenarios and construct a constraint set composed entirely of verifiable constraints which can be rigorously validated through automated scripts. For example, these constraints include response length requirements, linguistic norms, formatting guidelines, etc. By leveraging automated validation, this approach eliminates the need for manual intervention.

Instruction Evolution. Based on the constraint set, we prompt the LLM to evolve seed instructions into new ones by explicitly incorporating a randomly sampled subset of constraints (typically no more than six). These constraints guide the LLM to generate instructions with clear operational requirements. In addition, the LLM is required to explicitly specify the parameter values corresponding to these constraints (e.g., number of keywords, word limits), which are recorded for subsequent validation.

Response Generation with Validation Filtering. Finally, we utilize LLM to generate responses for the newly constructed instructions. Leveraging both the constraint definitions and the parameter values associated with each instruction, we design specialized validation scripts for each type of constraint. These scripts evaluate the model's outputs based on execution feedback and automatically filter out responses that fail to meet the constraints. This process ensures that the resulting instruction-response pairs consistently adhere to predefined quality standards.

6 Engineering

We describe the hardware and infrastructure that powered pre-training at scale and describe several optimizations that leads to improvements in training efficiency.

6.1 Infrastructure

The previous version of TeleChat(Wang et al., 2024b) was trained on a cluster with 640 NVIDIA A100 GPUs. As we scaled up to new series of TeleChat, training was migrated to ctyun's Shanghai compute center, which provides the computational power essential for training trillion-scale models.

Compute. The new series of TeleChat family were trained on up to Atlas 900 A2 cluster with 8k Ascend NPUs. Each node in the cluster contains 8 HCCS-connected NPUs. Training jobs are scheduled using a MindCluster-based platform.

Storage. Storage resources comprise Cluster Management (CM) nodes, Metadata Server (MDS) nodes, Object Storage Server (OSS) nodes, and physical storage devices known as OceanDisk. CM nodes are connected to cloud-based storage systems via dual 25 Gbps links, providing a management interface for distributed storage operations. OceanDisk devices are directly connected to the MDS and OSS nodes using a Fibre Channel (FC) network, ensuring high-speed and low-latency communication for data storage and retrieval. These four types of nodes and devices collectively form the High Performance File System (HPFS) shared storage system, which is optimized for distributed and high-throughput workloads. The HPFS shared storage system is uplinked to the RDMA over Converged Ethernet (RoCE) switches via dual 100GE links, enabling seamless integration with the larger network infrastructure and ensuring high-bandwidth access for compute and storage nodes.

Network. The parameter communication network adopts a two-layer Clos architecture (Charles Clos topology). Each training server connects its 200GE uplink to the RoCE switch, achieving high-speed 200GE RoCE interconnection between processing units. The Spine/Leaf hierarchy is configured with a nonconverged design to ensure maximum bandwidth availability. The parameter communication network incorporates Network-Side Load Balancing (NSLB) to ensure efficient load balancing at the link layer during large model training. This approach mitigates hash collisions and improves the overall throughput efficiency of the computational cluster.

6.2 Parallel Computing

6.2.1 Parallelism Strategies

The distributed training of Telechat2 is based on the 4D parallelism strategy provided by MindSpore's general-purpose large-model parallel framework (MindSpore, 2025). This framework is designed to support efficient and scalable training of large-scale models by integrating four key parallelism strategies: Data Parallelism (DP; Rajbhandari et al. (2020); Zhao et al. (2023)), Tensor Parallelism (TP; Shoeybi et al. (2020)), Pipeline Parallelism (PP; Huang et al. (2019); Narayanan et al. (2021)), and Context Parallelism (CP; Liu et al. (2023a)).

Data Parallelism (DP): The input dataset is partitioned along the batch dimension, with different device groups independently processing separate data batches. During backpropagation, gradient synchronization is performed across all devices, ensuring consistent updates to model parameters. This approach is particularly effective for scaling to larger datasets and improving hardware utilization across distributed systems.

Tensor Parallelism (TP): Model weights are partitioned across devices to reduce memory usage and computational overhead. Intermediate results are exchanged and aggregated using collective communication primitives such as All-Gather and ReduceScatter, which enable efficient distributed computation of tensor operations.

Table 3: Scaling configurations and MFU for TeleChat 2 115B pre-training.

Model Size	Cards	DP	TP	PP	CP	VPP	BS	Seq Len	Tokens/Batch	MFU (%)
115B	512	8	8	8	1	1	128	8k	1M	36.3
115B	4096	64	8	8	1	2	512	8k	4M	33.8
115B	6144	96	8	8	1	3	768	8k	6M	34.1
115B	4096	8	8	4	16	2	32	128k	4M	34.5

Pipeline Parallelism (PP): The model is divided into layers, or stages, with each stage assigned to a specific group of devices. Forward and backward passes are executed in a pipelined manner to maximize parallelism. To mitigate the inefficiencies caused by pipeline bubbles, strategies such as load balancing and virtual pipeline scheduling are employed.

Context Parallelism (CP): This strategy, unique to MindSpore, implements a 3D sequence parallelism scheme designed to handle long-sequence tasks efficiently. By splitting sequence computations across devices, CP alleviates memory and computation constraints associated with large input sequences.

To determine the optimal parameters for distributed parallelism, we conducted extensive experiments across various configurations. Tensor Parallelism (TP) incurs communication overhead due to operations such as All-Gather and ReduceScatter, while Pipeline Parallelism (PP) is affected by inefficiencies introduced by Bubble and Send/Recv communications. By employing load balancing and other optimization techniques to reduce pipeline bubbles, we found that PP parallelism consistently outperformed TP in terms of efficiency. After carefully tuning the parallelism configuration, hardware resources, and software optimizations, we achieved a Model FLOPs Utilization (MFU; Chowdhery et al. (2022)) of 33.8%-36.3% for the configurations shown in Table 3.

In large-scale distributed training, maintaining precise control over the global batch size is critical for ensuring model convergence and achieving optimal performance. It is well-documented that excessively large batch sizes can adversely affect convergence dynamics and final model quality. For this reason, the global batch size is typically constrained between 4M and 8M tokens during the initial stages of training. When training Telechat-115B on a 4096-NPU cluster, the increased data-parallel (DP) dimension led to a larger tokens per batch. To constrain tokens per batch to 4M, the number of micro-batches in the pipeline was reduced, which increased pipeline bubbles and lowered overall efficiency. To address this, we utilized the Virtual Pipeline Parallelism (VPP) feature to minimize bubbles, resulting in an MFU of 33.8%. When scaling to a 6144-NPU cluster, we increased the VPP factor to 3, further reducing the pipeline bubble ratio and improving the MFU to 34.1%. For ultra-long sequence training with a sequence length of 128k, we leveraged Context Parallelism (CP) to alleviate the memory and computational pressure associated with long sequences. This approach enabled training Telechat-115B on a 4096-NPU cluster, achieving an MFU of 34.5%.

These results demonstrate the effectiveness of carefully balancing parallelism strategies and leveraging advanced features such as VPP and CP to optimize distributed training efficiency, particularly when scaling to large clusters and handling long-sequence datasets.

6.2.2 Training Optimizations

In addition to these foundational parallelism strategies, Telechat's distributed training integrates several advanced optimizations enabled by MindSpore. Selective Re-computation is utilized to reduce memory overhead by recomputing select activations during backpropagation instead of storing them. Optimizer Parallelism enhances training efficiency by distributing the computational workload of optimizer operations across devices. Fine-grained multi-replica features allow for overlapping of computation and communication, effectively masking communication latency and improving end-to-end throughput. Furthermore, Pipeline Parallelism Optimizations leverage Virtual Pipeline Parallelism (VPP), employing a 1F1B (one forward, one backward) scheduling strategy combined with pipeline load balancing adjustments to achieve higher utilization of computational resources.

Selective Re-computation. During large-scale model training, activations generated in the forward pass are typically stored for use in the backward pass, resulting in significant memory consumption. This issue is exacerbated in Pipeline Parallelism (PP), where activations from multiple micro-batches must be accumulated, further increasing memory pressure. For models exceeding 70B parameters, a common approach is to omit activation storage and recompute activations during the backward pass, thereby reducing memory usage. However, this method introduces additional computation during backpropagation, potentially lowering computational efficiency.

To address this, the new series of TeleChat training leverages the Selective Re-computation capability provided by MindSpore. This approach selectively applies re-computation to key operators, balancing memory savings with computational overhead. Specifically, we targeted operators within the Feed-Forward Network (FFN), including Silu and Mul, as well as the Cast operator (from fp32 to bf16) in RMSNorm (Root Mean Square Normalization). These operators were chosen for their low computational cost and significant impact on reducing memory allocated for activations. This strategy allowed us to optimize memory usage while maintaining training efficiency.

Additionally, MindSpore supports communication-aware selective re-computation, which, when combined with optimizer parallelism, achieves effects similar to Zero3. MindSpore also enables layer-wise re-computation, selective re-computation, and communication-aware re-computation, further integrated with pipeline parallelism optimizations. These advanced techniques collectively optimize memory allocation and computation, ensuring efficient training of large-scale models.

Optimizer Parallelism. In data-parallel training, parameter updates are redundantly computed across devices, leading to inefficient memory usage and suboptimal performance in large-scale networks. Optimizer Parallelism addresses this issue by distributing optimizer computations across the devices in the data-parallel dimension. Specifically, model parameters and gradients are divided into slices based on device IDs, with each device independently updating its assigned slice. Once updated, the parameters are aggregated across devices using communication operations. This approach offers the benefit of natural load balancing, ensuring that each device has an equal share of parameters and computations. However, it imposes a constraint that parameter shapes must be divisible by the number of devices. The theoretical gains of this method align with parameter sharding, and several optimizations were introduced in TeleChat's distributed training to enhance its effectiveness.

- **Weight Sharding for Static Memory Reduction:**Model weights are partitioned to further reduce static memory consumption. To preserve the original tensor shapes for forward and backward passes, shared weights are aggregated at the end of each iteration and redistributed before the next iteration's forward pass.
- **Overlap Communication to Improve Performance:**A primary drawback of optimizer parallelism is the communication overhead associated with sharing weights. By overlapping communication operations with forward computations, we can minimize the perceived communication latency. Specifically, cross-iteration execution of communication allows communication operators to be grouped and fused, enabling efficient interleaving of communication and computation.

Pipeline Parallelism Optimization. In pipeline-parallel training scenarios, memory imbalance is a prominent challenge, particularly as the frontend stages often face significant memory pressure. To address this issue, we implemented an optimization strategy that combines adjusting the number of layers assigned to each stage with differentiated recomputation strategies:

- **Memory-Intensive Stages:**For stages experiencing high memory pressure, we reduced the number of layers allocated to these stages and adopted selective recomputation for all layers. This approach maximizes memory savings while balancing computational trade-offs.
- **Memory-Light Stages:**Conversely, stages with less memory pressure were assigned additional layers and employed selective recomputation for only a subset of layers, striking a balance between memory usage and computational efficiency.

To ensure the effectiveness of large-scale model training, batch token sizes are typically constrained (e.g., 8M or 16M). When training with a large cluster, the significant increase in data parallelism (DP) results in a smaller micro-batch size. Under a fixed number of pipeline stages, smaller micro-batches lead to larger pipeline bubbles, negatively impacting training efficiency. To enhance the efficiency of pipeline parallelism and reduce the proportion of bubbles, we adopted Virtual Pipeline Parallelism (VPP) during the training of the TeleChat2 model with 115B parameters. Traditional pipeline parallelism generally assigns consecutive layers (e.g., Transformer layers) to a single stage.

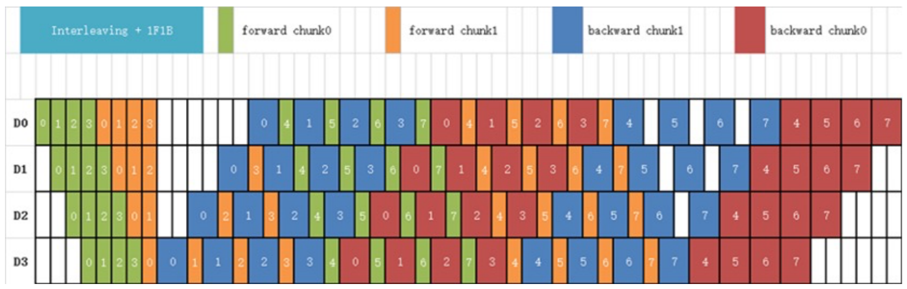

Figure 5: Example of Virtual Pipeline Parallelism (VPP) Scheduling: Demonstrates the interleaved computation of non-consecutive layers using the IF1B (One Forward One Backward) strategy. The scheduling mechanism reduces pipeline bubbles by overlapping communication and computation phases, while maintaining load balancing across stages.

In contrast, VPP scheduling employs interleaved computation of non-consecutive layers within each stage(figure 5). By increasing communication overhead, this approach significantly reduces the bubble ratio, thereby improving overall training performance.

Long-Sequence Optimization. To support long-sequence training with lengths of 128k–256k tokens, we implemented Sequence Parallelism (also known as Context Parallelism) by splitting the sequence dimension of the query, key, and value (QKV) tensors. This approach effectively reduces memory consumption. During the Attention computation phase, the sequence dimensions of the key and value tensors are reassembled using all-gather communication.

To achieve sequence load balancing, we utilized point-to-point all-gather communication to exchange sequence dimension data of the query and Attention results across devices. This enables swapping computationally intensive sequences from later stages with lighter sequences from earlier stages, ensuring a balanced computation load across devices.

For even longer sequences (e.g., on the order of millions of tokens), we employed the Ring-Attention algorithm provided by MindSpore. This method avoids fully reassembling the sequence dimensions of the key and value tensors during Attention computation. Instead, it performs blockwise computation on local QKV data, ensuring mathematical equivalence while achieving complete load balancing and overlapping computation with communication. This optimization further reduces memory consumption and enhances performance when training on ultra-long sequences.

6.2.3 Reliability and Challenges

During the pretraining phase of TeleChat2, hardware failures were the primary cause of service interruptions, including issues with optical modules, HBM (High Bandwidth Memory), and memory components. In response to these challenges, we implemented the following measures:

Recovery Mechanism Optimization. Optimized failure recovery by improving the mechanisms for storing and loading logs, checkpoints, and data, while upgrading the training framework and scheduling platform. These enhancements significantly reduced the time needed for resuming training after interruptions and preemptively addressed cluster environment issues through version checks.

Hardware Reliability Improvements. Reinforced inspection routines for critical hardware such as HBM, optical modules, and memory. Additionally, stricter standards for hardware replacement were established, and the hardware issue resolution process was streamlined.

As a result of these efforts, the weekly failure rate in the mid-to-late stages of pretraining was maintained below 1%. Training interruptions caused by hardware failures were significantly reduced, with an average Mean Time Between Failures (MTBF) of 4 days and a maximum interval of 21 days for core cluster hardware. The cluster availability metrics were strong, with weekly uptime consistently exceeding 99% and the longest uninterrupted training session lasting 288 hours.

Despite significant improvements, several critical challenges remain unresolved, which continue to hinder further advancements in training reliability and efficiency. The absence of efficient fault diagnosis tools has resulted in cascading issues, such as difficulty interpreting error codes, challenges

Table 4: Hardware Failure Statistics During Pretraining

Failure Category	**Count**	**Proportion (%)**
Optical Module Failure	36	19
HBM Failure	33	18
Memory Failure	14	8
DPU Failure	14	8
AI Core Failure	6	3
Optical Cable Failure	5	3
Motherboard Failure	5	3
Hard Drive Failure	3	2
Overheating	3	2
CPU Failure	2	1
NPU Failure	2	1
Power Supply Failure	2	1
RAID Failure	1	1
Controller Failure	1	1
Others	57	31

in pinpointing the root node of errors, and the inability to monitor shared storage utilization effectively in terms of space and performance. These deficiencies not only prolong downtime during failures but also increase the complexity of troubleshooting and system recovery. Developing robust diagnostic tools and monitoring systems to address these gaps will be essential for minimizing disruptions, optimizing resource utilization, and ensuring seamless scaling in distributed training environments.

7 Evaluation

7.1 Pre-trained Model

The performance of TeleBase2 is evaluated across a diverse range of benchmarks based on the internal evaluation framework. For base models, the assessment focus on their performance in general knowledge, commonsense, logical reasoning, mathematical problem-solving, and coding capabilities. The benchmarks we evaluated are listed as follows:

- **General knowledge** benchmarks include C-Eval (Huang et al., 2023) (zero-shot), MMLU (Hendrycks et al., 2021a) (5-shot), MMLU-pro (Wang et al., 2024a) (5-shot), CMMLU (Li et al., 2023b) (5-shot), GAOKAO (Zhang et al., 2024b) (zero-shot), AGIEval (Zhong et al., 2023) (zero-shot), GPQA (Rein et al., 2023) (5-shot) and TheoremQA (Chen et al., 2023) (5-shot).
- **Commonsense** benchmarks include CommonsenseQA (Talmor et al., 2019) (5-shot) and TruthfulQA (Lin et al., 2022) (zero-shot).
- **Logical Reasoning** benchmarks include BBH (Suzgun et al., 2022) (3-shot) and HellaSwag (Zellers et al., 2019) (zero-shot).
- **Mathematical Problem-Solving** benchmarks include GSM8K (Hendrycks et al., 2021b) (4-shot), MATH (Hendrycks et al., 2021c) (4-shot) and Ape210K (Zhao et al., 2020) (1-shot).
- **Coding** benchmarks include HumanEval (Chen et al., 2021) (zero-shot), MBPP (Austin et al., 2021) (3-shot), Humaneval+ (zero-shot), MBPP+ (3-shot) (Liu et al., 2023b).

In Table 5, we compare TeleBase2-35B, trained at context lengths of 8K, 32K, and 256K, with Qwen2.5-32B-base. In Table 6, we compare TeleBase2-115B, trained at context lengths of 8K, 32K, and 128K, with Qwen2.5-72B-base. All models are systematically evaluated using a customized evaluation framework with standardized settings to ensure a fair and rigorous comparison.

7.2 Post-trained Model

To comprehensively evaluate the quality of instruction-tuned models, we utilized automated benchmarking frameworks to assess performance of the thinking model (**T1**) and the non-thinking model

Table 5: Comparison among TeleBase2-35B trained under 8K, 32K and 256K context length and Qwen2.5-32B base model.

Benchmark	**TeleBase2-35B-8K**	**TeleBase2-35B-32K**	**TeleBase2-35B-256K**	**Qwen2.5-32B**
		General Knowledge		
C-Eval	87.2	87.8	86.2	86.1
MMLU	72.4	74.2	71.0	75.6
MMLU-pro	47.0	48.4	43.0	62.1
CMMLU	77.2	77.9	76.7	88.3
GAOKAO	68.6	63.2	59.1	52.1
AGIEval	68.9	71.3	69.3	82.7
GPQA	36.5	37.8	38.0	41.5
TheoremQA	41.0	42.8	40.3	44.3
		Commonsense		
CommonsenseQA	88.4	85.7	85.3	83.4
TruthfulQA	57.2	54.0	55.0	70.0
		Logical Reasoning		
BBH	81.7	82.6	82.5	70.0
HellaSwag	96.2	91.6	90.2	93.0
		Mathematical Problem-Solving		
GSM8K	85.2	86.2	86.3	75.0
MATH	69.2	71.6	70.0	61.2
Ape210K	66.8	66.0	67.0	65.5
		Coding		
HumanEval	73.8	70.7	73.8	78.0
MBPP	65.2	65.2	68.9	74.0
Humaneval+	66.0	66.0	67.4	69.5
MBPP+	70.5	71.4	70.5	70.5

Table 6: Comparison among TeleBase2-115B under 8K, 32K and 128K context length and Qwen2.5-72B base model.

Benchmark	**TeleBase2-115B-8K**	**TeleBase2-115B-32K**	**TeleBase2-115B-128K**	**Qwen2.5-72B**
		General Knowledge		
C-Eval	94.0	92.3	91.0	89.5
MMLU	81.0	79.9	78.9	77.2
MMLU-pro	53.2	53.0	52.5	63.8
CMMLU	82.0	81.3	80.0	90.3
GAOKAO	73.6	72.3	73.7	68.9
AGIEval	69.7	70.0	71.8	84.7
GPQA	41.3	41.3	38.3	40.3
TheoremQA	44.8	45.8	45.3	46.5
		Commonsense		
CommonsenseQA	86.7	85.3	85.7	87.1
TruthfulQA	62.6	61.6	61.0	71.0
		Logical Reasoning		
BBH	81.5	82.7	82.8	85.1
HellaSwag	97.4	92	92.6	96.8
		Mathematical Problem-Solving		
GSM8K	90.3	84.5	86.0	76.5
MATH	72.0	74.0	72.4	62.0
Ape210K	68.8	72.0	67.7	66.5
		Coding		
HumanEval	72.6	69.5	67.7	78.7
MBPP	70.0	69.4	68.0	75.2
Humaneval+	65.9	63.4	61.6	71.3
MBPP+	71.0	71.9	68.8	71.4

(**TeleChat2** and **TeleChat2.5**). The instruct models are evaluated under the following benchmarks to compare their capabilities.

- **AlignBench** (Liu et al., 2024a) is a comprehensive, multi-dimensional benchmark for evaluating Chinese large language models (LLMs) in alignment with human values and real-world requirements. It comprises 8 core categories, 683 real-scenario queries, and human-verified references.
- **IFEval** (Zhou et al., 2023) is a benchmark that evaluates large language models' ability to follow verifiable instructions. It provides 25 instruction types and around 500 prompts, each with quantifiable criteria.
- **BFCL** (Berkeley Function-Calling Leaderboard) (Patil et al., 2025) is a benchmark designed to evaluate large language models' (LLMs) function calling and tool use capabilities. The benchmark employs multi-dimensional evaluation methodologies, including single-turn function calling, multi-turn function calling, and hallucination detection. The BFCL benchmark results presented in this paper specifically reflect the single-turn performance on python-ast track, reporting averages for both non-live and live subtasks.
- **MATH500** is derived from the original MATH dataset (Hendrycks et al., 2021c), which comprises 5K mathematical problems.

For **T1** models, we employ a sampling temperature of 0.6, top-p of 0.95, top-k of 50, and repetition penalty of 1.05. For **TeleChat2** and **TeleChat2.5**, models use greedy search with a repetition penalty of 1.01. For both modes, we set the max output length to 32,768 tokens. The evaluation results for TeleChat model series alongside comparisons with other models with comparable parameter sizes under similar settings, are presented in Tables 7 and 8.

The evaluation results demonstrate TeleChat series models' robust capabilities across both thinking and non-thinking modes. **T1-115B** achieves exceptional performance in thinking mode, surpassing OpenAI o1-mini by +4.0 points on MATH500 (94.0 vs. 90.0) and +0.31 on Alignbench (8.22 vs. 7.91). In non-thinking mode, **TeleChat2.5-115B** outperforms GPT-4o-1120 by +12.0 points on MATH500 (87.0 vs. 75.0) and demonstrates a +4.74 advantage in BFCL (83.39 vs. 78.65). The TeleChat2.5-35B variant also remains competitive against similarly sized alternatives. Compared to Deepseek-R1-Qwen32B-distill, **TeleChat2.5-35B** achieves +5.67 points on IFEval (78.26 vs. 73.33) and +3.97 points on BFCL (80.11 vs. 76.14), demonstrating stronger performance in thinking mode.

Table 7: Comparison among **T1-35B**, **TeleChat2-35B**,**TeleChat2.5-35B** and other models under thinking/non-thinking mode with comparable parameter sizes.

Benchmark	**MATH500**	**Alignbench**	**IFEval**	**BFCL**
		Thinking		
T1-35B	90.0	7.93	78.26	80.11
Deepseek-R1-Qwen32B-distill	94.3	7.42	73.33	76.14
QWQ-32B	96.0	7.97	80.09	83.10
Qwen3-32B	93.0	8.27	85.92	86.82
		Non-Thinking		
TeleChat2-35B	61.0	6.97	77.74	75.32
TeleChat2.5-35B	77.0	7.74	78.52	78.28
Qwen2.5-32B	82.0	7.39	79.44	82.11
Qwen3-32B(non-thinking)	83.0	8.23	84.07	81.84

Table 8: Comparison among **T1-115B**, **TeleChat2.5-115B**, **TeleChat2-115B** and other models under thinking/non-thinking mode.

Benchmark	#Params	MATH500	Alignbench	IFEval	BFCL
Thinking					
T1-115B	115B	94.0	8.22	80.15	83.39
OpenAI o1-mini	Unknown	90.0	7.91	79.07	-
Deepseek-R1	671B(A37B)	97.2	8.43	83.70	88.68
Non-Thinking					
TeleChat2-115B	115B	72.0	7.76	79.25	77.47
TeleChat2.5-115B	115B	87.0	7.94	80.93	83.39
Qwen2.5-72B	72B	82.0	7.62	83.70	79.15
GPT-4o-1120	Unknown	75.0	7.49	80.18	78.65
Deepseek-V3	671B(A37B)	90.2	8.06	86.10	77.66

8 Conclusion

The introduction of **TeleChat2**, **TeleChat2.5**, and **T1** series represents a significant advancement in large language model (LLM) development. Despite minimal architectural changes, these models achieve substantial performance improvements through systematic upgrades in both pre-training and post-training stages. By publicly releasing these models with scalable parameter configurations (35B and 115B), we empower researchers and developers to leverage cutting-edge LLMs for diverse applications, fostering innovation in natural language processing, code generation and reasoning. This work not only addresses critical gaps in prior research but also provides a robust foundation for future studies on large-scale model optimization and task-specific adaptation.

Acknowledgements

We are grateful to the MindSpore Team and SAC (Small & Agile Commando) Team at Huawei for their engineering support in developing and optimizing the model on the Ascend cluster infrastructure.

References

Hongjun An, Wenhan Hu, Sida Huang, Siqi Huang, Ruanjun Li, Yuanzhi Liang, Jiawei Shao, Yiliang Song, Zihan Wang, Cheng Yuan, Chi Zhang, Hongyuan Zhang, Wenhao Zhuang, and Xuelong Li. Ai flow: Perspectives, scenarios, and approaches, 2025. URL https://arxiv.org/abs/2506.12479.

Anthropic. Introducing the next generation of claude, 2024. URL https://www.anthropic.com/news/claude-3-family.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Wenhu Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. Theoremqa: A theorem-driven question answering dataset, 2023. URL https://arxiv.org/abs/2305.12524.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways, 2022. URL https://arxiv.org/abs/2204.02311.

DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, and et al. Dongjie Ji. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024a. URL https://arxiv.org/abs/2405.04434.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, and et al. Dongjie Ji. Deepseek-v3 technical report, 2024b. URL https://arxiv.org/abs/2412.19437.

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

Yao Fu, Rameswar Panda, Xinyao Niu, Xiang Yue, Hannaneh Hajishirzi, Yoon Kim, and Hao Peng. Data engineering for scaling language models to 128k context. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2025.

Google. Introducing gemini 2.0: our new ai model for the agentic era, 2024. URL https://blog.google/technology/google-deepmind/google-gemini-ai-update-december-2024/?utm_source=deepmind.google&utm_medium=referral&utm_campaign=gdm&utm_content=.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, and et al. Angela Fan. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Jujie He, Jiacai Liu, Chris Yuhao Liu, Rui Yan, Chaojie Wang, Peng Cheng, Xiaoyu Zhang, Fuxiang Zhang, Jiacheng Xu, Wei Shen, Siyuan Li, Liang Zeng, Tianwen Wei, Cheng Cheng, Bo An, Yang Liu, and Yahui Zhou. Skywork open reasoner 1 technical report, 2025. URL https://arxiv.org/abs/2505.22312.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021a. URL https://arxiv.org/abs/2009.03300.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021b. URL https://arxiv.org/abs/2103.03874.

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

Jian Hu, Jason Klein Liu, and Wei Shen. Reinforce++: An efficient rlhf algorithm with robustness to both prompt and reward models, 2025. URL https://arxiv.org/abs/2501.03262.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism, 2019. URL https://arxiv.org/abs/1811.06965.

Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Jiayi Lei, Yao Fu, Maosong Sun, and Junxian He. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. *arXiv preprint arXiv:2305.08322*, 2023.

"Teknium" "interstellarninja". Hermes-function-calling-dataset-v1. URL https://huggingface.co/NousResearch/hermes-function-calling-v1.

Hamish Ivison, Yizhong Wang, Jiacheng Liu, Zeqiu Wu, Valentina Pyatkin, Nathan Lambert, Noah A. Smith, Yejin Choi, and Hannaneh Hajishirzi. Unpacking dpo and ppo: Disentangling best practices for learning from preference feedback, 2024. URL https://arxiv.org/abs/2406.09279.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL https://arxiv.org/abs/2310.06825.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts, 2024. URL https://arxiv.org/abs/2401.04088.

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, 2023. URL https://arxiv.org/abs/2309.06180.

Chenliang Li, Hehong Chen, Mingshi Yan, Weizhou Shen, Haiyang Xu, Zhikai Wu, Zhicheng Zhang, Wenmeng Zhou, Yingda Chen, Chen Cheng, Hongzhu Shi, Ji Zhang, Fei Huang, and Jingren Zhou. Modelscope-agent: Building your customizable agent system with open-source large language models. *ArXiv*, abs/2309.00986, 2023a. URL https://api.semanticscholar.org/CorpusID:261531214.

Haonan Li, Yixuan Zhang, Fajri Koto, Yifei Yang, Hai Zhao, Yeyun Gong, Nan Duan, and Timothy Baldwin. Cmmlu: Measuring massive multitask language understanding in chinese, 2023b.

Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods, 2022. URL https://arxiv.org/abs/2109.07958.

Hao Liu, Matei Zaharia, and Pieter Abbeel. Ring attention with blockwise transformers for near-infinite context, 2023a. URL https://arxiv.org/abs/2310.01889.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation, 2023b. URL https://arxiv.org/abs/2305.01210.

Xiao Liu, Xuanyu Lei, Shengyuan Wang, Yue Huang, Zhuoer Feng, Bosi Wen, Jiale Cheng, Pei Ke, Yifan Xu, Weng Lam Tam, Xiaohan Zhang, Lichao Sun, Xiaotao Gu, Hongning Wang, Jing Zhang, Minlie Huang, Yuxiao Dong, and Jie Tang. Alignbench: Benchmarking chinese alignment of large language models, 2024a. URL https://arxiv.org/abs/2311.18743.

Xiaoran Liu, Hang Yan, Chenxin An, Xipeng Qiu, and Dahua Lin. Scaling laws of roPE-based extrapolation. In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=JO7k0SJ5V6.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.

MindSpore. Mindspore: Advanced ai framework. https://www.mindspore.cn/, 2025. Accessed: 5 Feb. 2025.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Anand Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on gpu clusters using megatron-lm, 2021. URL https://arxiv.org/abs/2104.04473.

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

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie

Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report, 2024b. URL https://arxiv.org/abs/2303.08774.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.

Richard Yuanzhe Pang, Weizhe Yuan, Kyunghyun Cho, He He, Sainbayar Sukhbaatar, and Jason Weston. Iterative reasoning preference optimization, 2024. URL https://arxiv.org/abs/2404.19733.

Shishir G. Patil, Huanzhi Mao, Charlie Cheng-Jie Ji, Fanjia Yan, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, et al. Tool learning with foundation models. *arXiv preprint arXiv:2304.08354*, 2023.

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

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*, 2023.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. *arXiv preprint arXiv:1910.02054*, 2020.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

Jiawei Shao and Xuelong Li. Ai flow at the network edge, 2024. URL https://arxiv.org/abs/2411.12469.

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. URL https://arxiv.org/abs/2402.03300.

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism, 2020. URL https://arxiv.org/abs/1909.08053.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2022.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. Challenging big-bench tasks and whether chain-of-thought can solve them, 2022. URL https://arxiv.org/abs/2210.09261.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge, 2019. URL https://arxiv.org/abs/1811.00937.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, Chuning Tang, Congcong Wang, Dehao Zhang, Enming Yuan, Enzhe Lu, Fengxiang Tang, Flood Sung, Guangda Wei, Guokun Lai, Haiqing Guo, Han Zhu, Hao Ding, Hao Hu, Hao Yang, Hao Zhang, Haotian Yao, Haotian Zhao, Haoyu Lu, Haoze Li, Haozhen Yu, Hongcheng Gao, Huabin Zheng, Huan Yuan, Jia Chen, Jianhang Guo, Jianlin Su, Jianzhou Wang, Jie Zhao, Jin Zhang, Jingyuan Liu, Junjie Yan, Junyan Wu, Lidong Shi, Ling Ye, Longhui Yu, Mengnan Dong, Neo Zhang, Ningchen Ma, Qiwei Pan, Qucheng Gong, Shaowei Liu, Shengling Ma, Shupeng Wei, Sihan Cao, Siying Huang, Tao Jiang, Weihao Gao, Weimin Xiong, Weiran He, Weixiao Huang, Weixin Xu, Wenhao Wu, Wenyang He, Xianghui Wei, Xianqing Jia, Xingzhe Wu, Xinran Xu, Xinxing Zu, Xinyu Zhou, Xuehai Pan, Y. Charles, Yang Li, Yangyang Hu, Yangyang Liu, Yanru Chen, Yejie Wang, Yibo Liu, Yidao Qin, Yifeng Liu, Ying Yang, Yiping Bao, Yulun Du, Yuxin Wu, Yuzhi Wang, Zaida Zhou, Zhaoji Wang, Zhaowei Li, Zhen Zhu, Zheng Zhang, Zhexu Wang, Zhilin Yang, Zhiqi Huang, Zihao Huang, Ziyao Xu, Zonghan Yang, and Zongyu Lin. Kimi k1.5: Scaling reinforcement learning with llms, 2025. URL https://arxiv.org/abs/2501.12599.

Shubham Toshniwal, Ivan Moshkov, Sean Narenthiran, Daria Gitman, Fei Jia, and Igor Gitman. Openmathinstruct-1: A 1.8 million math instruction tuning dataset. *ArXiv*, abs/2402.10176, 2024. URL https://api.semanticscholar.org/CorpusID:267681752.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark, 2024a. URL https://arxiv.org/abs/2406.01574.

Zihan Wang, Xinzhang Liu, Shixuan Liu, Yitong Yao, Yuyao Huang, Xuelong Li, Yongxiang Li, Zhonghao Che, Zhaoxi Zhang, Yan Wang, Xin Wang, Luwen Pu, Huinan Xu, Ruiyu Fang, Yu Zhao, Jie Zhang, Xiaomeng Huang, Zhilong Lu, Jiaxin Peng, Wenjun Zheng, Shiquan Wang, Bingkai Yang, Xuewei he, Zhuoru Jiang, Qiyi Xie, Yanhan Zhang, Zhongqiu Li, Lingling Shi, Weiwei Fu, Yin Zhang, Zilu Huang, Sishi Xiong, Yuxiang Zhang, Chao Wang, and Shuangyong Song. Telechat technical report, 2024b. URL https://arxiv.org/abs/2401.03804.

Mingyu Xu, Xin Men, Bingning Wang, Qingyu Zhang, Hongyu Lin, Xianpei Han, and weipeng chen. Base of roPE bounds context length. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=EiIelh2t7S.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. Qwen2 technical report, 2024. URL https://arxiv.org/abs/2407.10671.

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

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence?, 2019. URL https://arxiv.org/abs/1905.07830.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *arXiv preprint arXiv:1910.07467*, 2019.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, Haolin Chen, Zhiwei Liu, Yihao Feng, Tulika Manoj Awalgaonkar, Rithesh Murthy, Eric Hu, Zeyuan Chen, Ran Xu, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, Silvio Savarese, and Caiming Xiong. xlam: A family of large action models to empower ai agent systems. *ArXiv*, abs/2409.03215, 2024a. URL https://api.semanticscholar.org/CorpusID:272424184.

Xiaotian Zhang, Chunyang Li, Yi Zong, Zhengyu Ying, Liang He, and Xipeng Qiu. Evaluating the performance of large language models on gaokao benchmark, 2024b. URL https://arxiv.org/abs/2305.12474.

Wei Zhao, Mingyue Shang, Yang Liu, Liang Wang, and Jingming Liu. Ape210k: A large-scale and template-rich dataset of math word problems, 2020. URL https://arxiv.org/abs/2009.11506.

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. Pytorch fsdp: Experiences on scaling fully sharded data parallel, 2023. URL https://arxiv.org/abs/2304.11277.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023. URL https://arxiv.org/abs/2306.05685.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. Agieval: A human-centric benchmark for evaluating foundation models, 2023.

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. URL https://arxiv.org/abs/2311.07911.