JoyAI-LLM Flash: Advancing Mid-Scale LLMs with Token Efficiency

JD.com

Abstract

We introduce JoyAI-LLM Flash, an efficient Mixture-of-Experts (MoE) language model engineered to redefine the equilibrium between elite performance and token length within the sub-50B parameter regime. JoyAI-LLM Flash was pretrained on a massive corpus of 20 trillion tokens, followed by a rigorous post-training pipeline encompassing heavily supervised fine-tuning (SFT), Direct Preference Optimization (DPO), and large-scale Reinforcement Learning (RL) across multifaceted environments. To improve token efficiency, JoyAI-LLM Flash strategically balances "thinking" and "non-thinking" cognitive modes and proposes a novel reinforcement learning algorithm dubbed FiberPO, inspired by the concept of "fiber bundles" to formalize the optimization space. To elevate architectural sparsity, the model with 48B total parameters maintains a lean active parameter count of only 2.7B per forward pass, achieving a significantly higher sparsity ratio than contemporary industry leaders of comparable scale. To accelerate inference throughput, we employ a co-design of training and inference optimizations, including dense Multi-Token Prediction (MTP) and Quantization-Aware Training (QAT). We are releasing the checkpoints for both the JoyAI-LLM-48B-A3B Base and its post-trained counterparts on Hugging Face to support the open-source community.

1 Introduction

The pursuit of highly capable Large Language Models (LLMs) often faces the dual challenges of a token efficiency crisis and computational cost [1]. During inference, many models tend to process an excessive number of tokens to achieve accurate outputs. While scaling the test-time computation [2] has historically driven performance gains, there is a growing need to fundamentally rethink the efficiency of intelligence.

We introduce JoyAI-LLM Flash, an instruct language model [3] with agentic, short-chain-of-thought (sCoT), and chat capabilities. JoyAI-LLM Flash leverages a sparse Mixture-of-Experts (MoE) architecture to achieve significant improvements on the inference-throughput-to-accuracy frontier, activating only a fraction of its parameters during each forward pass. Concretely, JoyAI-LLM Flash is based on a pure attention-based architecture with learnt MLP router activates 8 out of 256 experts plus 1 shared expert. JoyAI-LLM Flash has a total of 48B parameters, out of which only 2.7B are activated per forward pass (3.2B including embeddings). JoyAI-LLM Flash achieves better or competitive token-efficiency performance compared to other state-of-the-art models with comparable scale, as shown in Figure 1. The accuracy and token consumption averaged across eighteen benchmarks used in post-training evaluation are illustrated, where models located in the upper-right region are more token-efficient. Furthermore, on the 8K input / 16K output token scenario, JoyAI-LLM Flash achieves 1.45x and 1.07x speedups over the pure attention-based architectures GLM-4.7-Flash [4] and Qwen3-30B-A3B [5]. In terms of multi-token prediction (MTP) efficiency–defined as the inference speedup of the MTP model compared to its non-MTP counterpart–JoyAI-LLM Flash achieves a 1.87x speedup. This outperforms both hybrid-attention architectures Qwen3.5-35B-A3B [6] (1.61x) and Step-3.5-Flash [7] (1.39x). We open-source the base model and chat model weights with different quantization.

The base model of JoyAI-LLM Flash was pretrained on an extensive text-only corpus of over 20 trillion tokens, employing a Warmup-Constant-Cosine-Decay learning rate schedule. To maximize token utilization and incrementally build model capabilities, we divide the pretraining curriculum into four stages:

- Foundational Phase: Exposing the model to diverse tokens to build general linguistic capabilities.

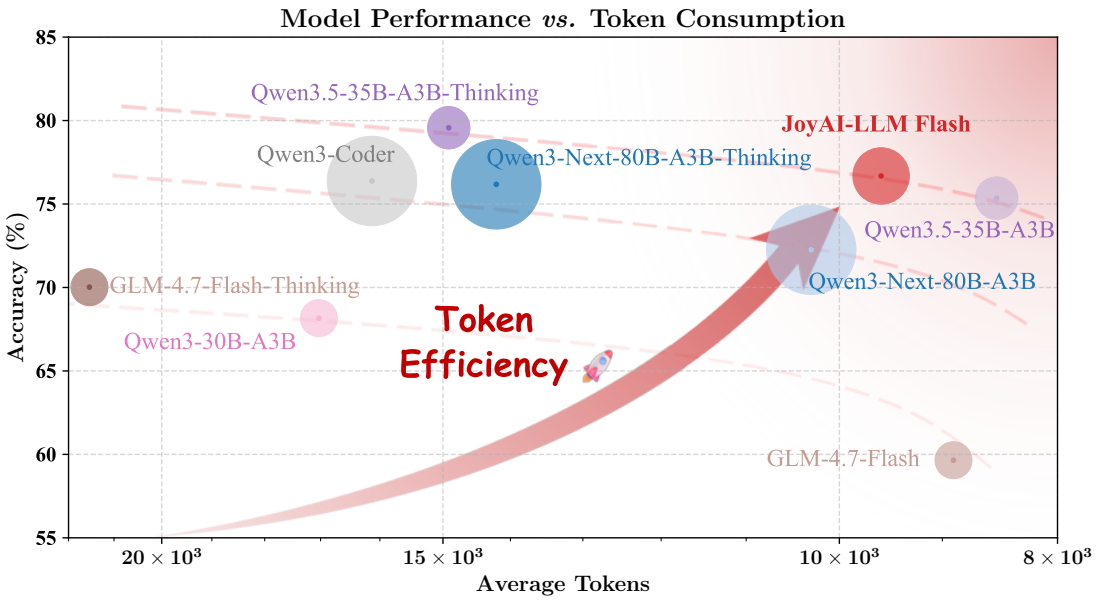


Figure 1: Model performance *vs.* token consumption. The accuracy and token consumption averaged across eighteen benchmarks used in post-training evaluation (Table 3) are illustrated, where the upper-right region indicates more token-efficient models. Bubble size represents the model parameter count.

- Code-Math-Enhancement Phase: Processing tokens with a significantly upweighted proportion of code and math data.
- Mid-Training Phase: Focusing on ultra-high-quality tokens to refine reasoning and alignment.
- Long-Context Phase: Utilizing nature long context tokens specifically engineered to extend the context window to 128K.

Empirically, JoyAI-LLM Flash Base achieves the competitive efficacy and efficiency for its size class across general knowledge, math, code, and comprehensive understanding evaluations.

Following pretraining, we implemented a rigorous post-training pipeline designed not only to align the model with human intent and enhance its autonomy, but also to fundamentally optimize token efficiency. This encompassing pipeline begins with heavily supervised fine-tuning (SFT) on a diverse set of high-quality traces, which strategically balance "thinking" and "non-thinking" cognitive modes, followed by Direct Preference Optimization (DPO) to refine responses and ensure robust human alignment. To further advance its reasoning and agentic problem-solving skills, we deployed large-scale Reinforcement Learning (RL) across multifaceted environments. Inspired by the algebraic concept of "fiber bundles", we introduce a novel RL algorithm named FiberPO to formalize the optimization space. Through these synergistic post-training innovations, JoyAI-LLM Flash emerges as a powerful foundation model, setting a new baseline for elite performance, token efficiency, and computational economy within the sub-50B parameter regime.

We also quantized JoyAI-LLM Flash from bfloat16 to FP8, INT8, FP4 and GGUF. Along with this report, we are releasing the model as follows:

Checkpoints

- JoyAI-LLM Flash Base 🤗: the pre-trained base model
- JoyAI-LLM Flash FP16 🤗: the post-trained model
- JoyAI-LLM Flash FP8 🤗: the post-trained model quantized with the FP8 format delivering an excellent trade-off between performance and efficiency
- JoyAI-LLM Flash INT8 🤗: the post-trained model quantized with the INT8 format achieving an optimal trade-off between performance and efficiency and compatible with some AI accelerators

Table 1: Detailed architectural configurations of JoyAI-LLM Flash.

Hyperparameter	JoyAI-LLM Flash 48B-A3B
General Settings	
Total Layers (N_{layers})	40
Dense Layers	1
Hidden Dimension (d_{model})	2048
Vocabulary Size ($\|V\|$)	129K
Max Context Length	128K
Activation Function	SwiGLU
Attention Mechanism	
Attention Type	MLA
Attention Heads (n_h)	32
QK Non-RoPE Dimension (d_{nope})	64
QK RoPE Dimension (d_{rope})	128
Value Dimension (d_v)	128
Mixture-of-Experts	
Total Routed Experts (N_r)	256
Activated Experts (K)	8
Shared Experts (N_s)	1
Expert Intermediate Size (d_e)	768

- JoyAI-LLM Flash INT4 🤗: the post-trained model quantized with the INT4 format serving as an ultra-compact variant tailored for environments with extremely restricted VRAM, such as consumer-level processors
- JoyAI-LLM Flash GGUF 🤗: the post-trained model quantized with the GGUF ultra-low-bit format delivering broad compatibility across personal computers

The rest of this report is arranged as follows. Section 2 describes the pretraining process of JoyAI-LLM Flash. Section 3 describes the post-training process, and Section 4 shows the inference technology used in JoyAI-LLM Flash. Section 5 concludes the report and presents future work.

2 Pretraining

In this section, we introduce the key features of JoyAI-LLM Flash Base, including its architecture, hyperparameters, and pretraining data. We also demonstrate that JoyAI-LLM Flash Base achieves competitive results to the state-of-the-art models.

2.1 Model Architecture

As summarized in Table 1, JoyAI-LLM Flash is a Mixture-of-Experts (MoE) model with 48.9B total parameters, of which 3.28B are activated per token. Its micro-architecture draws inspiration from DeepSeek-V3 [8] and Kimi-K2 [9], utilizing Multi-head Latent Attention (MLA) [3] with hidden dimensions of 2048 and 768, respectively. The model incorporates standard components such as RMSNorm [10] for layer normalization, RoPE [11] for positional encoding, and SwiGLU [12] activation within the feed-forward blocks.

In terms of macro-architecture, JoyAI-LLM Flash consists of 40 Transformer layers. The first layer utilizes a standard dense feed-forward network, while the remaining 39 layers are sparse MoE layers. The MoE module employs a fine-grained architecture with 256 total experts. For each input token, the model activates a total of nine experts: eight routed experts are dynamically selected via a Top-8 gating mechanism, and a single dedicated shared expert is always activated to capture common knowledge. To ensure numerical stability, the gating scores are computed using a sigmoid function and executed in FP32 precision. Additionally, we implement an auxiliary-loss-free load-balancing strategy [13] to maintain optimal utilization across the expert pool.

Muon Optimizer. To maximize sample efficiency and accelerate convergence, we employ the Muon optimizer [9, 14]. Unlike standard Adam, which relies on element-wise scaling, Muon optimizes parameters by leveraging matrix orthogonalization, effectively performing a form of Newton-style update on the spectral norm of the gradients. Previous studies [9, 14] have demonstrated that this approach significantly accelerates model convergence compared to Adam-based optimizers. Beyond these advantages, our empirical results reveal that Muon substantially enhances training robustness. During our experiments, training sessions utilizing Adam were frequently plagued by several loss spikes, which required manual intervention or careful adjustment of learning rates. In contrast, training with Muon remained exceptionally stable, with no significant numerical anomalies observed.

Multi-Token Prediction. We append a lightweight, single-layer dense Multi-Token Prediction (MTP) head to jointly optimize training and inference [8, 15]. During pre-training, this module enriches the learning signal, enabling the model to capture multi-step dependencies for improved data efficiency. During inference, it natively enables speculative decoding. By predicting multiple future tokens in parallel, this mechanism overcomes the latency bottleneck of standard autoregressive decoding and accelerates generation without requiring an external draft model.

2.2 Infrastructure

The JoyAI-LLM Flash training system is built upon a highly optimized extension of the Megatron-Core [16] framework. Along with foundational parallelization strategies—including Data Parallelism (DP), Tensor Parallelism (TP) [16], Sequence Parallelism (SP) [16], and the 1F1B Pipeline Parallelism (PP) [17, 18, 18, 19, 20, 21, 22, 23] schedule, we have introduced several architectural enhancements to maximize throughput and computational efficiency. Our specific configuration employs 2-way Pipeline Parallelism, 8-way Expert Parallelism (EP) [24] spanning two nodes, and ZeRO-1 Data Parallelism [25]. To further accelerate core operations, we integrate FlashAttention-3 [26] for high-performance attention kernels and utilize the DeepEP [8] library to minimize latency during token dispatch and combination within the MoE layers. Additionally, by leveraging distributed asynchronous checkpointing, we reduce loading times from 15 minutes to 30 seconds, ensuring the model can recover and resume training in less than a minute. We also implement a packing training strategy utilizing block-diagonal masks to isolate unrelated samples and preserve strict causal boundaries. Compared to the conventional full lower-triangular masking approach, this method accelerates the attention forward and backward passes by 50% and 20%, respectively.

2.3 Pretraining Data

In this section, we detail the composition and processing pipeline of our pretraining corpus. Our model was trained on a total of 20.7 trillion high-quality tokens. The dataset is curated from four main sources: diverse web crawls, reasoning-intensive code repositories, high-fidelity PDF documents, and large-scale synthetic data. The composition is designed to balance broad general knowledge with deep reasoning capabilities and domain-specific expertise.

2.3.1 Web Data Pipeline

We processed Common Crawl data up to October 2025 using a high-efficiency pipeline:

Text Extraction. To achieve high-quality content extraction, we process WARC files using the Trafilatura library, which more effectively removes boilerplate and menu text while filtering out HTML artifacts to extract the core semantic text.

Rule-based Cleaning. Our data refinement process utilizes the *Datatrove* framework [27], integrated with several customized modules for high-precision filtering:

- **Standard Filtering:** We employ URL filtering to block known malicious domains and a *fastText* classifier to retain only high-confidence English and Chinese documents. Content quality is further ensured through quality and heuristic repetition filters.
- **Privacy Preservation:** The PII (Personally Identifiable Information) detection logic was significantly expanded to cover a more comprehensive set of global identity markers, providing robust anonymization.
- **Optimized Decontamination:** To address the issue of excessive data removal during decontamination, we introduced an n-gram whitelisting mechanism. This refinement reduces the probability of "false deletions", ensuring that only true evaluation overlaps are removed.
- **Semantic Safety Classifier:** A dedicated BERT-based sensitive content classifier was added to our pipeline. This model performs deep semantic analysis to detect policy-prohibited data, ensuring the final dataset aligns with safety and ethical guidelines.

Deduplication. To mitigate redundancy, we developed a distributed deduplication pipeline based on MinHash-LSH[28, 29] on a Ray-based distributed cluster. Our approach involves decomposing documents into 7-gram shingles,

followed by the generation of compact 128-permutation MinHash signatures. We employ a Jaccard similarity threshold of 0.9 to identify near-duplicate candidates. These candidates are subsequently clustered using a parallelized Union-Find algorithm, ensuring that only a single canonical representative from each equivalence class is retained in the final corpus.

Model-based Filtering. To address the limitations of static rules in capturing nuanced quality, we fine-tuned Qwen [5] model series to create two specialized filtering models:

- **Line-level Noise Filter:** In filtered web crawled data, we observed persistent noise such as embedded advertisements, navigation bars, and templated boilerplate. We trained a lightweight classifier to evaluate each line, removing non-narrative content while preserving the semantic integrity of the document.
- **Multi-dimensional Scoring and Classification:** To ensure the highest data quality, we evaluates every document across several key metrics: factual accuracy, linguistic coherence, information density, grammatical correctness, thematic depth, web noise, safety and multi-topic identification. We only retained documents categorized as "high-quality" based on the integrated scores. This strict filtering significantly improved the model's learning efficiency.

2.3.2 Code Data Pipeline

Rule-Based Cleaning Pipeline. Our raw corpus primarily comes from The Stack v2[30] and large-scale GitHub code extraction. We follow a multi-stage cleaning workflow: rule-based filtering to remove obvious noise, model-based scoring for second-stage selection and stratification, and deduplication to reduce redundancy. Generic quality signals capture repetition (high-order n-gram duplication and duplicate-line ratios), length and scale (character/line counts, file size, extreme line lengths), character composition (ratios of alphabetic/digit/whitespace characters, hex-like fragments, hyperlinks/HTML tags), and suspicious patterns such as autogenerated or encoded content. Language-specific signals further characterize structural and semantic parseability (e.g., AST availability), function-to-line ratios, test-file patterns, preprocessing-directive density, and excessive trivial statements (e.g., `print`, `assert`, `pass`). Guided by downstream scoring, we relax overly aggressive heuristics by moving them from hard filtering to the scoring stage, improving the precision–coverage trade-off.

Model-Based Quality Scoring. To reduce the cost of per-sample LLM evaluation, we train a lightweight regression scorer to approximate a large-capacity judge model. We use Qwen2.5-3B-Instruct[31] as the backbone and target sequences shorter than 32k tokens. Labels are produced by Qwen2.5-Coder-32B-Instruct[32], which scores each sample 10 times; we take the minimum score to mitigate stochasticity. We define medium-quality data as scores in (2.5, 6) and high-quality data as scores greater than 6.

MinHash-LSH Deduplication. To remove exact and near-duplicate code, we reuse the MinHash-LSH based Ray deduplication scheme introduced in our data cleaning pipeline. In practice, most code duplicates we eliminate are exact file-level copies rather than merely similar variants.

Long-Context Code Construction. For long-context code data, we construct 64k and 128k token sequences by concatenating longer QA pairs and repository-level code that have already passed the preceding filtering and deduplication stages. For repositories, we build lightweight language-specific dependency graphs over modules and files, and within each connected component we apply a topological ordering routine to obtain file sequences that respect dependency directions from lower-level to higher-level modules—that is, an ordering in which every dependency appears before any module that depends on it; for repository-level data we explicitly split the mixture so that roughly half of the sequences follow this topological order and the other half use a random file ordering, exposing the model both to realistic project layouts and to more diverse context permutations.

Code Rewriting and Composition. For synthetic code data, we adopt a single-pass rewriting strategy inspired by SwallowCode[33] and OLMo3[34] (SGCR+SCOR). We start from the deduplicated and filtered GitHub corpus and a curated top-20-language subset as seeds, and ask an LLM to rewrite functions, files, and small multi-file snippets into more instruction- and documentation-like forms while preserving semantic intent; rewritten outputs are again deduplicated against both the seed pool and the original source files, routed through the same rule filtering and model scoring pipeline, and we retain samples with scores > 7, most of which empirically fall into the high-quality range (typically scoring above 8).

For QA-style code data, we primarily rely on Nemotron-Competitive-Programming-v1[35] as seed problems and solutions, and use the DeepSeek V3.2[36] model to produce paired "thinking" and "no-thinking" variants; these rewritten QA examples are then treated as code–QA compositions and subjected to the same deduplication and quality filtering stack as the rest of the corpus.

2.3.3 PDF Parsing and Knowledge Extraction

To mitigate the scarcity of high-quality, specialized content in open-web corpora, we curated a massive dataset comprising tens of millions of PDF documents. This collection prioritizes domain-specific knowledge often underrepresented in general web text, including STEM, Medicine, Social Sciences, Education, Humanities, and Law. The processing pipeline follows a rigorous workflow similar to our web text pipeline, with specialized enhancements for the PDF format:

- **Text Extraction:** We leverage MinerU [37] and DeepSeek-OCR [38] to perform high-fidelity document parsing. This ensures the precise recovery of complex mathematical formulas, tables, and hierarchical structures.
- **Filtering and Deduplication:** We implement a series of heuristic rules to rectify post-extraction artifacts, such as repetition or extraction noise. Furthermore, documents are unsuitable for linguistic modeling are systematically filtered.
- **Semantic Chunking:** Documents are partitioned into segments of approximately 4,096 tokens by utilizing double-newline delimiters as natural boundaries, we ensure that chunks respect semantic integrity, thereby avoiding arbitrary truncation during the training phase.
- **Quality Scoring:** Finally, the extracted text undergoes a scoring and filtering process consistent with our web-scale pipeline to ensure high data quality.

Unlike standard web crawls, these documents provide the structured, professional knowledge essential for the model to acquire advanced academic and technical expertise.

2.3.4 Large-Scale Data Synthesis

Synthetic data plays a critical role in our data pipeline, evolving from strengthening factual knowledge in early training to eliciting advanced, multi-step reasoning and agentic behavior in later stages.

Factual-knowledge reformulation. We synthesize factual pre-training data via two complementary transformations. First, we apply the MAGA reformulation method [39] to rewrite high-quality web passages into diversified, instruction-following styles while preserving the original semantics, thereby expanding stylistic coverage and reducing template bias. Second, we perform Nemotron-CC–style QA rewriting [40], converting curated Common Crawl text into question–answer pairs and short instructional exemplars that elicit explicit information retrieval and grounded responses.

Long-form reasoning QA synthesis. We construct reasoning-intensive supervision through two streams. First, we use DeepSeek V3.2 [36] to generate full solutions for real-world STEM problems, and retain only responses that are verified either by major voting or by matching available ground truth. Second, inspired by Nvidia's RQA method [35], we derive graduate-level "Thinking QA" from STEM papers by turning technical content into questions that require multi-step derivations, and we synthesize multiple independent reasoning paths per question to encourage robust, self-consistent reasoning rather than pattern imitation. Overall, during the mid-training stage, we increase the proportion of synthetic data to above 60% of total tokens to explicitly prioritize advanced reasoning.

Agentic trajectory synthesis. To further augment general agentic capabilities, we complement the reasoning-centric mixture with large-scale tool-use trajectories generated through a staged execution pipeline, as illustrated in Figure 2. We first sample diverse atomic tasks across domains, then compose them into more challenging multi-intent tasks while removing repeated goals and duplicate combinations. These tasks are compiled into executable scripts and instantiated via multi-turn simulations, where GLM-4.6 [4] serves as the primary agentic actor to simulate complex user–agent interactions under varied patterns. We subsequently employ an LLM-based evaluator to filter trajectories against a comprehensive set of rubrics, covering aspects such as role consistency, task completeness, and planning conciseness.

2.4 Training Strategy

We train the model using a rampup strategy, combined with a Warmup-Stable-Decay (WSD) learning rate schedule [41]. A sequence length of 4,096 tokens is used throughout training, except during the context extension stage.

Stage 1 (Foundational pretraining). During the warmup phase, we train on 100B tokens while linearly increasing the batch size from approximately 13M to 38M tokens and ramping the learning rate to a peak of 4.2×10^{-4}. In the Stable phase, we maintain a batch size of 38M tokens and a learning rate of 4.2×10^{-4}.

Stage 2 (Code-Math-Enhancement pretraining). This phase is also regarded as the decay phase, we train with the learning rate following a cosine schedule from 4.2×10^{-4} to 1.4×10^{-4}.

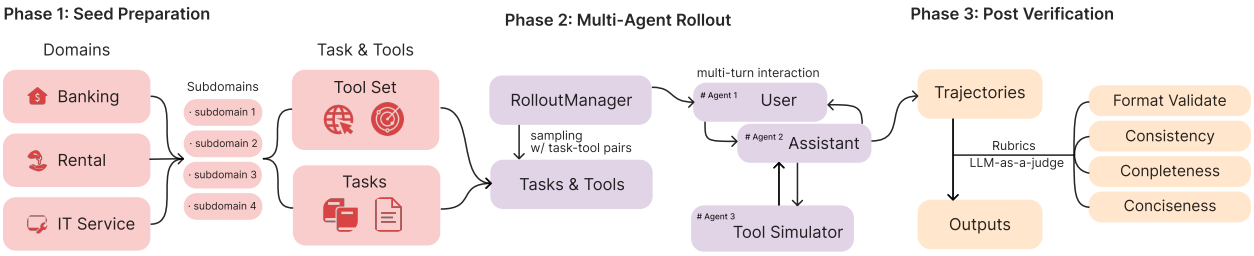
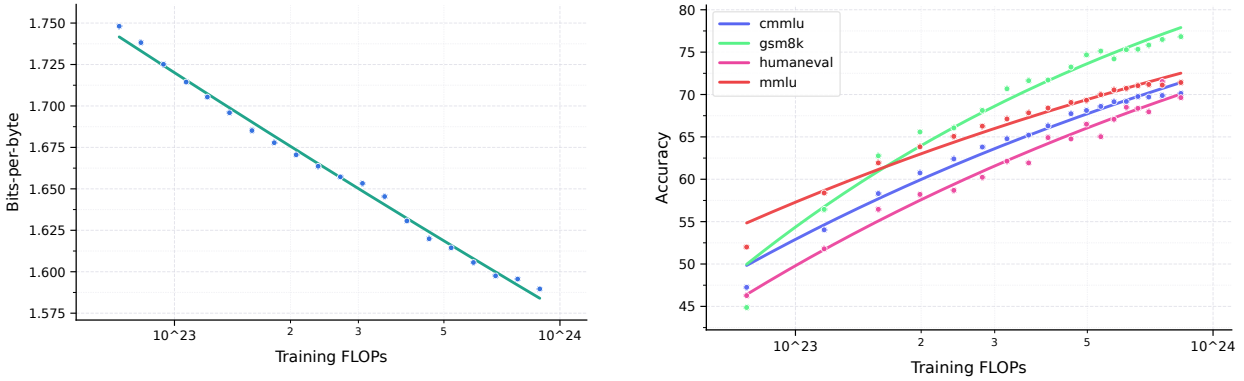


Figure 2: Agentic trajectory synthesis pipeline

Stage 3 (Mid-training). We continue training high-quality data to further refine the model. The learning rate decays from 1.4×10^{-4} to 4.2×10^{-5}. In this stage, we enable a single-layer dense Multi-Token Prediction (MTP) [8, 15] with a loss scaling factor of 0.1.

Stage 4 (Context Extension). Training proceeds in two steps, retaining the same Multi-Token Prediction configuration as Stage 2. First, we train with a 64K context window and a batch size of approximately 34M tokens, decaying the learning rate from 4.2×10^{-5} to 3.2×10^{-5}. We then extend the context window to 128K and train with the learning rate further decaying to 2.0×10^{-5}.

Scaling laws. During our model training process, we utilized the scaling law algorithm to guide and inform our approach. Building on previous work [42, 43, 44], we experimented with scaling both the model size and data volume. This strategy was crucial for anticipating the model's training needs due to the extensive resource requirements. The scaling law provided predictive insights throughout the training, particularly in terms of resource allocation, as well as adjustments to training hyperparameters and data compositions. As shown in the Figure 3, we compared the scaling law with model training loss and domain-specific benchmarks. The learning curve for training loss aligned perfectly with the scaling law's step predictions. Although downstream tasks displayed more variability, their performance was still commendable, generally fluctuating around the scaling law trend. A noteworthy discovery was the use of model merging to simulate a decay in the learning rate. This approach led to more stable improvements in the performance of sub-domain tasks, aligning with the scaling law trajectory. This finding underscores the potential of model merging to optimize learning dynamics within the framework of scaling laws.



(a) LOSS

(b) Benchmark

Figure 3: Scaling laws for JOYAI-LLM Flash. The plot illustrates the relationship between training compute and model performance. Data points represent empirical observations, while solid lines indicate the power-law fits.

2.5 Base Model Evaluation

To evaluate the comprehensive performance of JoyAI-LLM Flash, we select Qwen3-30B-A3B-Base [5] and the latest Qwen3.5-35B-A3B-Base [6] as our competitive baselines. Our evaluation framework is structured around four core domains: General Knowledge, Math, Coding, and Long-Context Processing. This multidimensional evaluation thoroughly validates the model's foundational capabilities with nine benchmarks.

- **General Knowledge**: MMLU [45] (5-shot, Cot), MMLU-Pro [46] (5-shot, Cot), CMMLU [47] (5-shot, Cot).

- **Math**: GSM8K [48] (4-shot, Cot), MATH [49] (4-shot), MATH-500 [49] (4-shot).
- **Coding**: HumanEval [50] (5-shot), LiveCodeBench [51] (v6, 2023.05-2025.04).
- **Long-Context**: RULER [52].

To ensure fair and reproducible comparisons, we adopt a standardized evaluation pipeline. Most benchmarks are executed with OpenCompass under greedy decoding for deterministic reporting. LiveCodeBench and RULER are evaluated using their official repositories to remain consistent with their native leaderboards. For LiveCodeBench, we use the default settings (Temperature=0.2, Top-p=0.95, Top-k=20, Repetition Penalty=1.05); for RULER, we follow the official execution protocol.

Table 2: Comparison of Base Model between Qwen3-30B-A3B, Qwen3.5-35B-A3B and JoyAI-LLM Flash. Best results are marked in bold.

Task	Qwen3-30B-A3B-Base	Qwen3.5-35B-A3B-Base	JoyAI-LLM Flash-Base
General Knowledge			
MMLU	82.1	**88.4**	84.7
MMLU-Pro	61.7	60.7	**73.1**
CMMLU	83.6	**86.1**	83.1
Math			
GSM8K	90.3	**90.5**	88.7
MATH	59.6	56.0	**78.1**
MATH-500	58.0	54.8	**77.0**
Coding			
HumanEval	**87.8**	79.8	85.3
LiveCodeBench	37.3	**42.6**	39.9
Long-Context			
RULER (128K)	61.8	**88.3**	77.0

Based on the results in Table 2, JoyAI-LLM Flash shows a competitive profile relative to the Qwen models. On broad general-knowledge benchmarks, it is slightly behind the strongest baseline, suggesting its factual coverage is comparable but not consistently better. On reasoning-intensive and math evaluations, it performs more strongly, indicating better robustness on harder multi-step problem solving under the reported setup. For coding, results are broadly on par with the baselines, with small differences depending on the benchmark.

3 Post-Training

In contrast to contemporary mid-scale models, JoyAI-LLM Flash dedicates a significantly larger proportion of its computational budget to the post-training phase. We structure this rigorous alignment pipeline into three sequential stages: Supervised Fine-Tuning (SFT), Direct Preference Optimization (DPO), and Reinforcement Learning (RL). During the SFT stage, we deliberately interleave "thinking" and "non-thinking" cognitive data mixtures. Empirical observations indicate that this hybrid training approach substantially enhances the performance of the instruct model (non-thinking model). Following SFT, we introduce a dedicated DPO phase to refine response quality and mitigate hallucinations. The inclusion of DPO before RL is strategically motivated by its rapid convergence, providing a highly efficient mechanism for penalizing negative or undesirable responses early in the alignment process. Finally, building upon the DPO-aligned foundation, we apply a novel, large-scale RL algorithm designed to maximize token efficiency and further elevate the model's agentic problem-solving capabilities.

3.1 Supervised Fine Tuning (SFT)

We establish that the SFT phase is fundamental to realizing the comprehensive capabilities of JoyAI-LLM Flash. Rather than merely aligning output formats or following instructions, this stage is instrumental in expanding the model's knowledge and amplifying its core cognitive capacities. To this end, we implement a heavily weighted SFT protocol comprising a diverse training mixture across three distinct categories: general SFT, environment and agent learning, and tool-integrated reasoning (TIR).

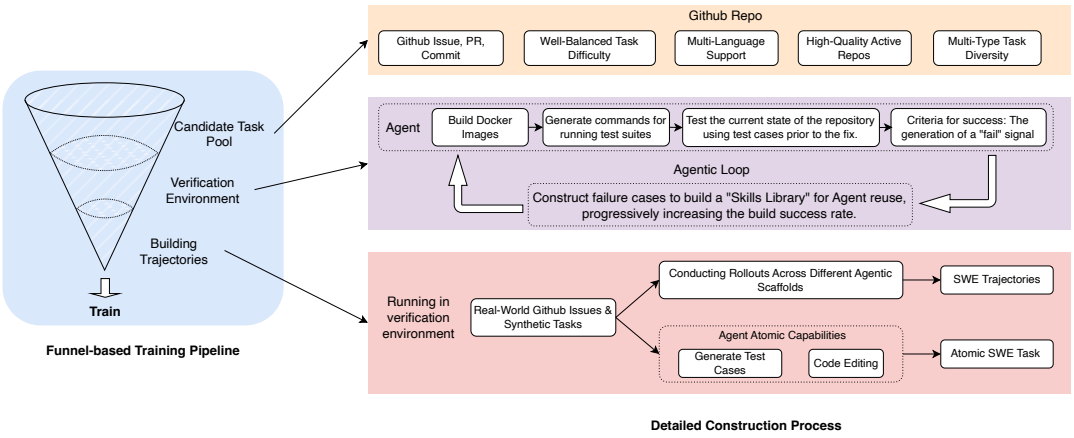


Figure 4: Verifiable Environment Pipeline

3.1.1 General SFT

Our general Supervised Fine-Tuning (SFT) corpus encompasses a comprehensive spectrum of domains, including mathematics, coding, tool utilization, instruction following, safety, science, Lean theorem proving, creative writing, role-playing, language and multilingual understanding. To construct this dataset, we aggregate substantial volumes of real-world and synthetic prompts, paired with high-quality responses derived from both human annotators and state-of-the-art open-source models, such as JoyAI-LLM Pro, DeepSeek-V3.2 [36], Qwen3-235B-A22B [5], and GPT-OSS 120B [53]. To maintain stringent quality standards, we employ Qwen3-30B-A3B [5] as a specialized filter to systematically remove low-quality queries. Notably, we eschew curriculum learning during this phase in favor of a unified training approach.

Recognizing the pivotal role that data mixture plays across both the mid-training and SFT stages, we apply a human-in-the-loop scheme [54] to dynamically optimize domain proportions. Specifically, we heavily weight coding and agent-centric data to constitute approximately 30% of the mixture, followed closely by general chat and STEM domains.

To maximize computational efficiency during training, we pack sequences to a context length of 128K using a best-fit packing algorithm. This technique reduces the padding ratio to a negligible 0.01%, ensuring that computational resources are exclusively allocated to effective tokens. To preserve strict causal boundaries and prevent cross-contamination between unrelated samples, we apply block-diagonal attention masks within the packed sequences. Overall, this sequence packing strategy yields a 1.5x improvement in training throughput compared to standard padding methodologies.

3.1.2 Environment and Agent Learning

To systematically optimize Software Engineering (SWE) tasks, as illustrated in Figure 4, we modeled the workflow as a conversion funnel, analogous to the traffic funnels used in internet marketing. From the initial mining of tasks on GitHub to the final rollout of agent trajectories, each successive stage involves a natural "conversion loss," where the pool of viable tasks diminishes. By monitoring each stage of this pipeline through the lens of a funnel model, we can precisely analyze bottlenecks and guide optimization efforts to improve the overall conversion rate of usable tasks. The pipeline is structured into three primary phases:

- **Candidate Task Mining** The initial stage focuses on excavating raw candidate tasks from GitHub. This involves filtering repositories and issues based on specific heuristics to ensure the tasks are substantive and representative of real-world engineering challenges.
- **Verifiable Environment Construction** Following mining, we transition to the environment setup phase. Here, we transform raw tasks into reproducible environments. The objective is to ensure that each task has a functional "ground truth" (e.g., passing/failing tests) that can be used for automated verification and subsequent Reinforcement Learning (RL).
- **Trajectory Generation and Cold Start** The final stage involves rolling out interaction trajectories using an agent framework. These successful trajectories (pass the test cases) serve as high-quality data for Supervised Fine-Tuning (SFT) to "cold start" the model's performance.

Candidate Task Mining. The candidate task pool is primarily derived from real-world GitHub issues, Pull Requests (PRs), and commit metadata. To ensure the data is suitable for model training, we apply a series of heuristic filters—such as the clarity of PR descriptions and the number of files modified—to select tasks with an appropriate level of complexity. Notably, our tasks are not restricted to Python. Instead, our pipeline encompasses 12 mainstream programming languages. To guarantee the quality and maturity of the source repositories, we enforce strict selection criteria: each repository must have over 10 stars and a history of at least two successfully merged Pull Requests. The tasks are categorized into several distinct engineering domains to ensure diverse functional coverage:

- **Bug Fix** Identifying, diagnosing, and resolving defects within the source code.
- **Feature Enhancement** Improving or expanding existing functionality to add value to the project.
- **Refactoring** Modifying source code to optimize internal structure without changing external behavior.
- **Test Case Generation** Automatically generating unit tests and utilizing test suites to verify code integrity.

Verifiable Environment Construction. To support the training requirements of SWE tasks, we have developed a large-scale infrastructure for execution sandboxes. In practice, we observed that simply constructing a Docker image capable of hosting the repository is insufficient; the execution of test suites often fails due to unresolved package dependencies and other environmental inconsistencies. Consequently, a repository-level image alone cannot adequately support the training workflow. To address this, we have decoupled the environment construction process into two distinct phases. One is the Docker Image Provisioning stage, which focuses on the baseline containerization of the repository. The other is the Test Case Validation stage, where we execute the test suites to verify whether the constructed environment is "verifiable" — ensuring that the dependencies are correctly configured and the environment is functional for downstream training and evaluation.

- **Phase 1: Build & Initialize**. Manually constructing executable sandbox environments is both time-consuming and labor-intensive. To address this, we employ an Agent-based approach, leveraging autonomous agents to build Docker images directly from existing GitHub repositories. The construction process utilizes an agentic automation workflow: the agent iteratively attempts to build the image, while human intervention is introduced only for execution failures. Through this process, we extract and refine successful patterns into a reusable Skill Library. By continuously repeating this agentic loop, the system's efficiency and success rate improve over time. Upon the completion of an image build, a mandatory "smoke test" is executed. This serves as a preliminary validation step to determine whether the environment possesses the foundational operational capabilities required for subsequent tasks.
- **Phase 2: Test Execution & Verification**. We employ an Agent-based approach to locate relevant test cases and generate executable test commands. By testing the project code in its states both before and after the Pull Request (PR), we identify Pass-to-Pass (P2P) and Pass-to-Fail (P2F) transitions, which serve as the primary success/failure signals. To provide more precise reward signals for a given patch, we developed a multilingual test log parser. This tool is utilized not only to extract test results but also to evaluate the test coverage of the aforementioned schemes. Based on these metrics, we systematically prune Docker images with insufficient coverage to ensure the quality.

Building Trajectories. To maximize the utility of the verifiable environments, we utilized the SWE-Smith [55] framework to synthesize a batch of tasks. To ensure sufficient task complexity, we filtered out overly simplistic cases by monitoring the number of modified lines and edited files. Moving forward, the pipeline bifurcates into two distinct branches:

- **Trajectory Rollout**. We utilize multiple agent frameworks (e.g., OpenHands [56], SWE-agent [57], mini-swe-agent [57]) to generate (rollout) interaction trajectories that successfully pass the predefined test cases.
- **Atomic Capability Task Generation**. We derive specialized tasks focused on the atomic capabilities of an agent, such as precise code editing and automated test case generation.

3.1.3 Tool-Integrated Reasoning

Tool-Integrated Reasoning (TIR) enhances Large Language Models (LLMs) by incorporating external tool use into their reasoning process. Unlike traditional models that rely exclusively on pre-trained parametric knowledge, TIR enables models to decide when to invoke tools and to generate specific instructions—such as Python code or search queries. The execution results are then fed back into the model to inform subsequent reasoning and response generation. This iterative approach improves computational accuracy and information freshness, effectively addressing the inherent limitations of LLMs like calculation errors and knowledge cutoffs.

In this section, we construct and analyze four specialized TIR datasets tailored to distinct functional domains. We develop an automated, scalable data synthesis pipeline to generate high-quality reasoning trajectories. This pipeline focuses on capturing the specific reasoning patterns needed for Code Interpretation, Agentic Search, and hybrid scenarios that require the coordinated use of both tools. Although our proposed JoyAI-LLM Flash is an instruct model, we find adding reasoning/thinking data in the SFT stage can improve the non-thinking capacity of the instruct model.

Code-Centric Trajectories. We extract complex mathematical problems and, where available, their corresponding ground truths from datasets such as OpenR1-Math-220k [58] and Nemotron-Math-v2 [59]. Using DeepSeek-V3.2 [36], we distill these into TIR trajectories.

We have developed an interactive environment between the model and the Python interpreter, allowing the LLM to dynamically utilize the Python interpreter for iterative and symbolic computations. Our Python setup includes essential libraries that enable robust mathematical operations and solving capabilities. For example, the setup employs the `math` library for basic arithmetic and `sympy` for symbolic mathematics, with commands demonstrating tasks like initializing symbols and solving equations using `from sympy import symbols, Eq, solve`.

During data distillation, the model is restricted to a maximum of 20 tool invocations per problem session. Occasionally, the model may produce erroneous code, which the interpreter catches and returns as tool responses, aiding the model in debugging and retrying. After distillation, incomplete trajectories, as well as those containing plotting code such as `matplotlib`, are rigorously filtered out.

This process culminates in a comprehensive dataset of multi-round TIR records, integrating Python tools and providing a substantial foundation to assess and enhance LLM tool-integrated reasoning capabilities.

Search-Centric Trajectories. To cultivate robust information-seeking, multi-hop reasoning, and agentic tool-use capabilities, we construct a search-centric trajectory corpus by distilling execution traces from DeepSeek-V3.2 [36] across three data sources: `Complex QA & Agentic Benchmarks` (6,601 queries aggregated from seven established benchmarks including 2WikiMultihopQA [60], MuSiQue [61], Bamboogle [62], SimpleQA [63], FRAMES [64], ScholarSearch [65], and GAIA [66], covering multi-hop reasoning, factuality, complex RAG, and real-world agentic tasks), `TaskCraft` [67] (17K search-relevant instances selected from a large pool of tool-intensive agentic tasks spanning single-step to expert-level multi-step executions), and `Nemotron-Science-v1` [68] (20K sampled instances from a multiple-choice scientific reasoning corpus). For sources with verifiable ground-truth labels, we retain only trajectories with correct final answers. The resulting corpus comprises trajectories with an average of 8.64 search invocations each.

Rather than exposing the model to raw search results, we introduce a summary agent as the sole interface to search results. Upon each search invocation, the model issues a structured call with explicit search keywords and an intent statement; the summary agent returns a concise, query-focused synthesis of the retrieved web pages. This design prevents overly long contexts, reducing computational overhead while preserving model performance during both training and inference.

Hybrid Tool-Integrated Trajectories. Beyond task-specific datasets, we further curate a sophisticated category of trajectories that necessitate the synergistic coordination of both code interpreters and search engines. In these complex scenarios, the model must exhibit high-level planning: typically utilizing Agentic Search to retrieve specialized domain knowledge or external constants, followed by Code Interpretation to perform rigorous algorithmic verification or numerical modeling based on the retrieved data.

Terminal-Centric Trajectories. To enhance the model's proficiency in terminal-based operations, we synthesize a diverse set of task scenarios within a standardized, constrained Docker environment. Despite the restricted scope of the environment, we achieve high task diversity by utilizing capability decomposition and evolutionary sampling to expand from initial seed data. We employ DeepSeek-V3.2 [36] to generate reasoning-action trajectories for each task. To ensure data quality, we implement an automated validation pipeline in which an LLM-based judge evaluates each trajectory across five key dimensions: completion, correctness, efficiency, safety, and overall quality. Trajectories with low scores are strictly excluded. This filtering mechanism ensures that only functionally viable and safe data remain for training, providing a stable yet diverse signal for subsequent SFT.

By constructing these four TIR datasets, we aim to comprehensively evaluate and improve models' abilities to integrate diverse external tools into their reasoning process, thereby pushing the frontier of automated, tool-augmented artificial intelligence.

3.2 Direct Preference Optimization (DPO)

During the Direct Preference Optimization (DPO) phase, we train the model for 1,000 steps utilizing a learning rate of 1e-6 with cosine-decay to 1e-7 and a global batch size of 256. To construct the DPO dataset, we curate STEM, general conversational, and safety queries distinct from those of the SFT stage. We form preference pairs by contrasting high-quality responses with negative examples derived from rejection sampling during SFT, specifically targeting prevalent failure cases such as hallucinations and instruction-following deviations. Ultimately, the DPO stage is crucial to the model's final performance; its rapid convergence provides a highly efficient mechanism to systematically penalize and eliminate undesirable outputs prior to RL.

3.3 Reinforcement Learning (RL)

Large language models are no longer single, monolithic policies: they are increasingly deployed and trained as heterogeneous systems—agentic pipelines spanning domains and tools, mixture-of-experts (MoE) architectures with conditional routing, and distributed/asynchronous training stacks where optimization noise and data nonstationarity are structural rather than incidental. In this regime, alignment via RLHF [69] must simultaneously handle multi-scale instability: token-level stochasticity, trajectory-level drift, and system-level heterogeneity (domains/experts/agents) interacting in the same update. Existing PPO-style "proximal" objectives [70, 71, 72] provide only coarse local controls (mostly per-token clipping) and limited diagnostics when failures arise from global structure (e.g., a drifting subset of trajectories, an expert partition, or a domain slice). This motivates importing more expressive mathematical structure, beyond new loss heuristics, to build controllers that can allocate stability budgets across the relevant global contexts. In our JoyAI-LLM, we develop Fiber Bundle Gating (FBG), a geometric framework grounded in fiber bundle theory, and derive FiberPO from it, a concrete policy optimization objective that decomposes trust-region maintenance into compositional global and local components, providing multi-scale stability control with first-order fidelity to the true RL objective near on-policy and a restorative gradient structure less explored in existing methods.

FiberPO rests on a principled theoretical foundation developed in [73]. Classical TRPO [74] trust regions collapse at the undiscounted horizon $\gamma{=}1$ required by LLM RL (the TRPO Vanishing Theorem in [73]). This does not preclude trust-region-style stabilization, but it shows that the classical radius cannot be used as-is, necessitating a decoupling of how trust regions are maintained (ratio clipping) from the specific radius prescribed by TRPO's monotonic improvement guarantee. An intermediate result, Aggregational Policy Censoring Objective (APC-Obj), achieves this decoupling by proving that the clipping-based surrogate can exactly reproduce trust-region updates (the APC-Obj-TRPO Equivalence Theorem in [73]), so that the clipping mechanism remains well-defined at any positive radius $\delta > 0$. APC-Obj also provides a unified Ratio Gating Formalism from which PPO [70], GRPO [71], and GSPO [75] are each derived as identified relaxations. This taxonomy reveals a structural gap: token-wise methods (PPO, GRPO) do not bound trajectory-level drift, while sequence-wise methods (GSPO) suppress within-trajectory variation. To compose the two scales, we introduce Fiber Bundle Gating (FBG), a geometric framework that organizes tokens as a fiber bundle over trajectory-level contexts and decomposes ratio gating into compositional base-level and fiber-level operations, with provable first-order agreement with the true RL objective near on-policy (the FBG First-Order Agreement Theorem in [73]).

FiberPO is the concrete instantiation of FBG derived from the APC-Obj objective through a sequence of controlled transformations [73]. The FiberPO objective factorizes each token's gated importance ratio into a trajectory-level base weight and a token-level gated residual. The base weight maintains a trust-region budget on trajectory-level drift through a piecewise-linear aggregate gate g^{agg}, while the gated residual clips each token's deviation from its trajectory mean via $\mathrm{logclip}$. This two-scale decomposition provides independent control at both levels, a structural property rarely explored in all prior methods. Because fibrations compose algebraically, the same gating mechanism extends to arbitrary hierarchical depth: [73] derives a Fibration Gating Hierarchy (FGH) and instantiates FiberPO-Domain, a four-level variant with independent trust-region budgets at the domain, prompt group, trajectory, and token levels. In this report we present the two-level trajectory-token case for conciseness. The full theoretical development is given in [73].

3.3.1 The FiberPO Objective

Let $r_i := \pi_\theta(a_i|s_i)/\pi_{\theta_{\mathrm{old}}}(a_i|s_i)$ denote the importance ratio for token i, $\hat{A}_i$ the estimated advantage, and T_τ the length of trajectory τ. The augmented token space $\bar{\mathcal{X}} := \{(s_t(\tau), a_t(\tau), \tau, t)\}$ indexes each token by its trajectory membership and timestep.

Definition 3.1 (FiberPO)**.** The FiberPO objective is:

$$\hat{J}^{\text{FiberPO}}(\theta|\theta_{\mathrm{old}}) = \sum_{(s,a,\tau,t)\in\bar{\mathcal{X}}} \frac{1}{|\mathrm{Tj}^{\theta_{\mathrm{old}}}|}\frac{1}{T_\tau} \cdot \mathcal{G}(r_\bullet)_{s,a,\tau,t} \ \cdot \hat{A}_{s,a}, \tag{1}$$

where $\mathrm{Tj}^{\theta_{\mathrm{old}}}$ is the set of sampled trajectories and the gating map $\mathcal{G}$ decomposes multiplicatively for each token $i \equiv (s, a, \tau, t)$:

$$\mathcal{G}(r_\bullet)_i \;=\; \underbrace{\frac{\exp \circ\, g^{\mathrm{agg}}(\log s_\tau^+,\; C^+,\; T_\tau)}{\exp \circ\, g^{\mathrm{agg}}(\log s_\tau^-,\; C^-,\; T_\tau)}}_{w_\tau^{\mathrm{base}}\ :\ \text{base weight}} \cdot \underbrace{\frac{\mathrm{logclip}\Big((s_\tau^{(l_i)})^{-l_i}\, r_i,\; \epsilon\Big)}{\mathrm{logclip}\Big((s_\tau^{(-l_i)})^{-l_i},\; \epsilon\Big)}}_{\tilde{r}_i^{\mathrm{fiber}}\ :\ \text{gated residual}}. \tag{2}$$

Each constituent is defined in detail below. At a high level, the decomposition reflects the fiber bundle structure of sampled RLHF data [73]: each token's log-ratio is split into a trajectory-level component (how much the trajectory as a whole has drifted) and a token-level residual (how much that token deviates from its trajectory's drift). The base weight w_τ^{base} corresponds to the base gate in the Fiber Bundle Gating (FBG). It depends only on trajectory-level aggregate log-ratios s_τ^+, s_τ^-, which separately track positive and negative drift within each trajectory based on the sign label $l_i := \mathrm{sign}(\log r_i) \in \{+1, -1\}$ (with $s_\tau^{(l_i)}$ selecting the same-sign channel and $s_\tau^{(-l_i)}$ the opposite), and is shared by all tokens in trajectory τ, controlling how much gradient signal the trajectory as a whole is permitted to contribute through the piecewise-linear gate g^{agg}. The gated residual $\tilde{r}_i^{\mathrm{fiber}}$ corresponds to the fiber gate. It captures each token's deviation from the trajectory aggregate, gated by $\mathrm{logclip}$ to prevent individual token spikes. Together, the two components provide compositional multi-scale control: the base weight maintains a trust-region budget at the trajectory level, while the gated residual regulates per-token outliers within each trajectory.

Intuitive example. To illustrate the practical significance of this decomposition, consider two trajectories answering "Name a famous landmark":

"I love Paris and the Eiffel Tower" vs. *"I love Rome and the Colosseum."*

Globally (trajectory-level), the policy may strongly prefer the Paris response, perhaps it scores higher overall, so the aggregate ratio s_τ^+ for that trajectory is large. Without decoupling, this global preference bias leaks into every token's gradient: the token *Colosseum* in the Rome trajectory receives a weaker learning signal not because the token-level association "Rome $\to$ Colosseum" is poor, but simply because its trajectory is globally less preferred. The residual decomposition in Eq. 2 prevents this contamination. By subtracting the trajectory aggregate from each token's log-ratio, the fiber gate $\tilde{r}_i^{\mathrm{fiber}}$ isolates the pure local association, how much "Colosseum" co-varies with "Rome" relative to what the trajectory drift alone would predict, and gates it independently via $\mathrm{logclip}$. Within each trajectory, token-level learning thus operates at a uniform, unbiased scale: $P(\text{Colosseum} \mid \text{Rome})$ and $P(\text{Eiffel Tower} \mid \text{Paris})$ are each refined on their own statistical merits, free from the global preference $P(\text{Paris trajectory}) \gg P(\text{Rome trajectory})$. The base weight w_τ^{base} then re-couples the trajectory-level preference when the two scales are composed, so that global significance is preserved without polluting local precision. This is the orthogonal, non-interfering decomposition guaranteed by the reflecting condition $\pi_{E*} \circ K = \mathrm{id}_{\mathbf{B}}$ [73].

The gating map $\mathcal{G}$ satisfies three structural properties established in [73]: (i) trajectory independence, the Jacobian of $\mathcal{G}$ is block-diagonal over trajectories, fully decoupling each trajectory's gradient, (ii) first-order agreement, at the on-policy point ($r_\bullet = \mathbf{1}$) the Jacobian reduces to identity, recovering the true RL objective to first order near on-policy, and (iii) scale separation, the local self-gating term has $O(1)$ magnitude while trajectory-mediated coupling is weighted by $1/T_\tau$, so that local gradients dominate near on-policy and trajectory-level corrections engage only as aggregate drift grows.

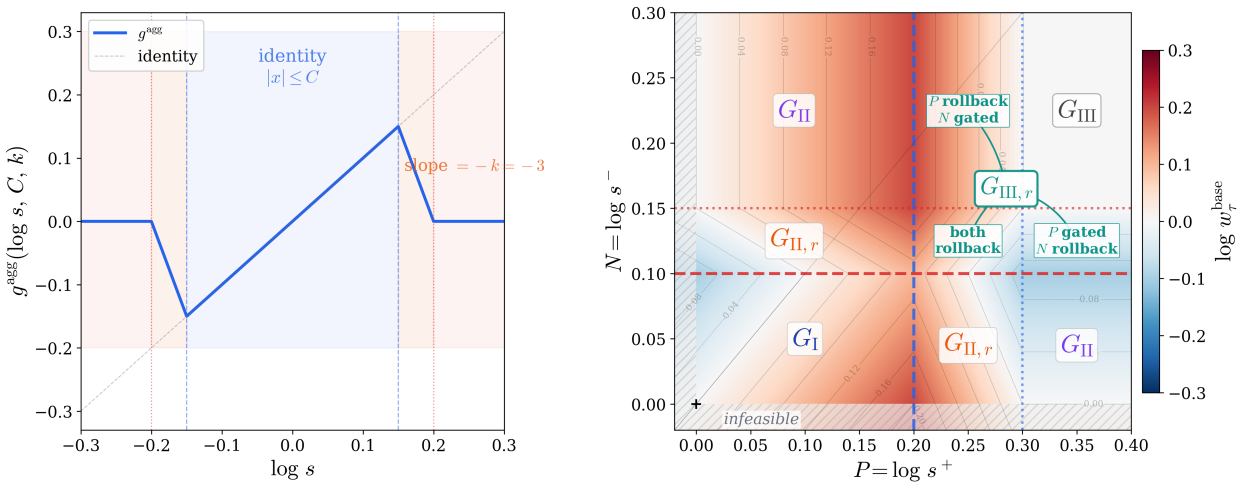
We now introduce each constituent of Eq. 2. For the base weight w_τ^{base} term, the positive and negative aggregate ratios decompose the trajectory-level drift by sign:

$$\log s_\tau^+ := \frac{1}{T_\tau} \sum_{t=0}^{T_\tau - 1} \max(\log r_{s_t(\tau), a_t(\tau)},\, 0), \qquad \log s_\tau^- := \frac{1}{T_\tau} \sum_{t=0}^{T_\tau - 1} \max(-\log r_{s_t(\tau), a_t(\tau)},\, 0). \tag{3}$$

The aggregate gating function g^{agg} is a piecewise-linear gate on each sign channel:

$$g^{\mathrm{agg}}(x, C, T_\tau) := \begin{cases} x & \text{if } |x| \le C \\ \mathrm{sign}(x)(T_\tau + 1)C - T_\tau x & \text{if } C < |x| < (1 + T_\tau^{-1})C \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

where $C \in \{C^+, C^-\}$ is the per-channel trust-region budget satisfying $C^+ + C^- = \delta$, with $C^- < C^+$ recommended to compensate the intrinsic KL bias $\log s_\tau^- \ge \log s_\tau^+$. The three regimes are: pass-through ($|x| \le C$, gate outputs x unchanged), rollback ($C < |x| < (1 + T_\tau^{-1})C$, slope reverses to $-T_\tau$ producing a restorative gradient), and zeroed



(a) g^{agg}**: 3-regime piecewise gating** (k**=3**) **(b)** $\log w_\tau^{\text{base}}$ (k**=2,** C^+**=0.2,** C^-**=0.1)**

Figure 5: (a) Aggregate gate g^{agg} (Eq. 4) with three regimes: pass-through ($|x| \le C$, slope 1), rollback ($C < |x| < C^* := (1 + T_\tau^{-1})C$, slope $-T_\tau$), and zeroed ($|x| \ge C^*$, output 0). As T_τ increases, the rollback zone narrows (width C/T_τ) and g^{agg} approaches a hard clip at $\pm C$. (b) Base weight $\log w_\tau^{\text{base}}$ (Eq. 2) in $(\log s^+, \log s^-)$-space with asymmetric thresholds. Dashed lines mark the budget boundaries $C^\pm$ (onset of rollback), and dotted lines mark the full-gating thresholds $C^{*\pm}$ (onset of zeroing). The five global regimes follow a non-monotonic pattern: $|\log w|$ rises through the rollback onset (G-II,r), peaks when one channel is fully gated (G-II), declines under mutual rollback (G-III,r), and collapses to zero when both channels are fully gated (G-III, $w_\tau^{\text{base}} = 1$).

($|x| \ge (1+T_\tau^{-1})C$, output 0, fully blocking gradient signal). Since w_τ^{base} is a ratio of two independently gated channels (Eq. 2), the combined behavior produces five global regimes (G-I through G-III) depending on which sign channel occupies. These range from nominal pass-through (G-I: both channels transparent, base weight equals the unmodified importance-sampling ratio) through one-channel rollback (G-II,r: restorative gradient actively opposes the drifting channel), one-channel fully gated (G-II: the drifting channel is zeroed, delivering maximum one-sided correction), mutual rollback (G-III,r), and extinction (G-III: both channels fully gated, $w_\tau^{\text{base}} = 1$, trajectory-level gradient vanishes). The restorative rollback property is absent in PPO, GRPO, and GSPO. Figure 5 visualizes g^{agg} and the five global regimes; see [73] for detailed regime definitions.

For the gated residual $\tilde{r}_i^{\text{fiber}}$ term in Eq. 2, the sign label l_i partitions the tokens within each trajectory into two channels, reflecting the FBG fiber bundle structure [73] where the base space $B = \text{Traj}^{\theta_{\text{old}}} \times \{-1, +1\}$ indexes each trajectory by a sign channel: the fiber over $(\tau, +1)$ collects all tokens whose likelihood increased, and the fiber over $(\tau, -1)$ collects those whose likelihood decreased. Splitting by sign rather than averaging all log-ratios is essential because the total trajectory drift $\overline{\log r}_\tau = \log s_\tau^+ - \log s_\tau^-$ can be small even when both $\log s_\tau^+$ and $\log s_\tau^-$ are individually large. In this case, the trajectory contains many tokens that have shifted substantially in both directions, and the trajectory-level total variation distance ($\approx (1/T_\tau) \sum_t |\log r_{s_t,a_t}|$) is large and may require control. Averaging all log-ratios into a single mean would mask this need for regulation. By tracking each sign channel independently, g^{agg} detects high total variation in the importance weights even when the signed average nearly cancels, and can apply rollback on the offending channel without suppressing the well-behaved one.

The log-clipping function is $\text{logclip}(x, \epsilon) := \exp(\text{clip}(\log x, \pm\epsilon))$. In ratio space, this clamps the argument to $[e^{-\epsilon}, e^{+\epsilon}]$. The fiber residual $u_i := l_i \log r_i - \log s_\tau^{(l_i)}$ measures each token's deviation from its same-sign trajectory mean. Define also the opposite-sign aggregate $v_i := -\log s_\tau^{(-l_i)}$. In terms of u_i and v_i, the gated residual Eq. 2 can be written equivalently as:

$$\tilde{r}_i^{\text{fiber}} = \exp\big(\text{clip}(l_i\, u_i,\ \pm\epsilon) - \text{clip}(l_i\, v_i,\ \pm\epsilon)\big). \tag{5}$$

The numerator's logclip acts on $e^{l_i u_i}$, which involves only the same-sign channel aggregate $s_\tau^{(l_i)}$, preventing opposite-channel contamination. The denominator's logclip incorporates the opposite-sign aggregate $s_\tau^{(-l_i)}$ to complete the subtraction by the trajectory-mean log-ratio $\overline{\log r}_\tau$. The ϵ-clip on the fiber residual u_i induces three local regimes: L-I (unclipped, all tokens retain full gradient), L-II (selective clipping of outlier tokens), and L-III (all saturated, gradient

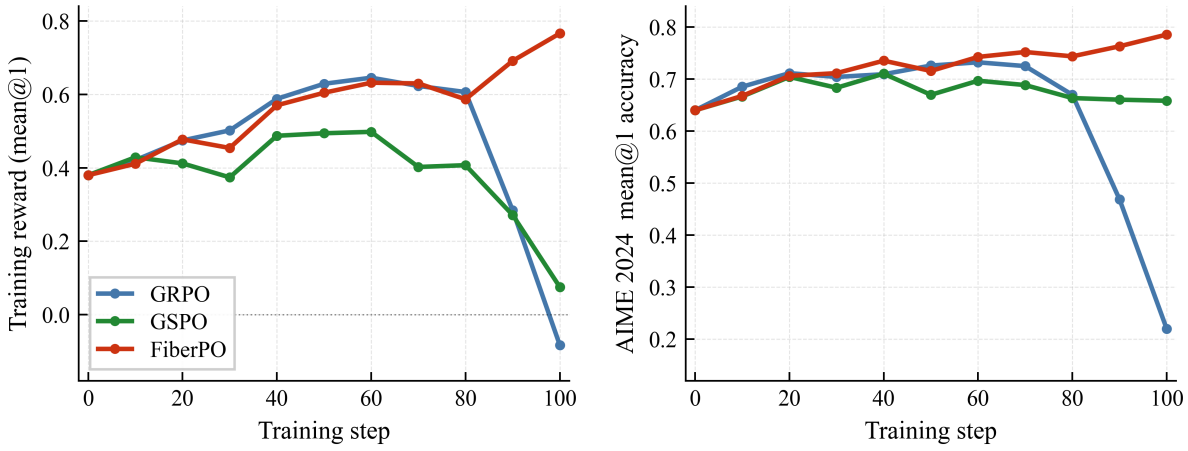
governed entirely by the base weight). The recommended hyper-parameter relationship $\epsilon \ll \delta$ ensures that local regulation engages before global regulation: the fiber gate clips outlier tokens (L-I to L-II) well before trajectory-level aggregates reach the g^{agg} budget threshold (G-I to G-II). See [73] for the joint local–global regime visualization on the probability simplex.

When neither logclip saturates (i.e., $|u_i| \le \epsilon$ and $|v_i| \le \epsilon$), the clips are inactive and we obtain $\log \tilde{r}_i^{\text{fiber}} = \log r_i - \overline{\log r}_\tau$, the trajectory-mean-centered log-ratio, recovering the true linear surrogate after multiplication by the base weight in G-I.

This fiber residual formulation yields a concrete *token-efficiency* advantage (with empirical evidence in Section 3.3.2). Because the logclip acts on $u_i = l_i \log r_i - \log s_\tau^{(l_i)}$ rather than on $\log r_i$ directly, a token is clipped only when it deviates from the same-sign trajectory mean by more than ϵ, regardless of the magnitude of the trajectory aggregate $\log s_\tau^{(l_i)}$ itself. Tokens that shift in concert with the trajectory-level drift always pass through the FiberPO gating map $\mathcal{G}$ unattenuated, retaining their full gradient signal and contributing *finer, discriminative per-token update directions* even when the signed trajectory-level drift is large ($|\log s_\tau^{\pm}| > \epsilon$). By contrast, methods such as PPO and GRPO that clip $\log r_i$ directly tie the clip threshold to the absolute log-ratio. Once the trajectory-level drift exceeds the clip bound, the majority of tokens saturate simultaneously, destroying token-level discrimination and collapsing the gradient to a coarse trajectory-level signal.

3.3.2 Single-Domain Evaluation

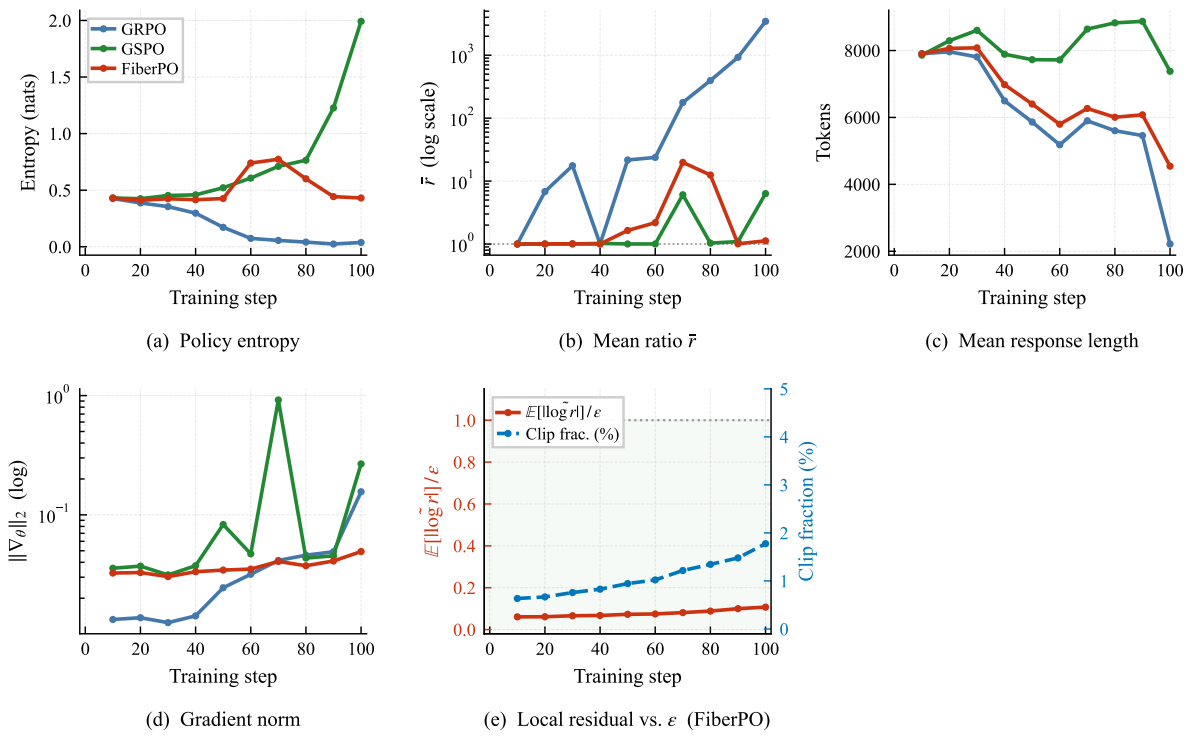
We evaluate FiberPO in a single-domain math RLVR setting and perform a pure algorithmic comparison against GRPO and GSPO, with no additional stabilizers (no curriculum learning, no overlong reward shaping or filters, etc.). We train on DAPO-Math-17k and evaluate on AIME 2024 following the default evaluation protocol in DAPO [72]. We initialize our reinforcement learning phase using the checkpoint produced by the aforementioned rigorous SFT and DPO stages. This presents a deliberately challenging scenario for RL optimization, as the extensive prior alignment significantly diminishes the policy's entropy. Figure 6 shows that FiberPO's training and validation curves both rise monotonically in the latter half of training. All methods are trained in verl [76] with matched infrastructure. We use a learning rate of 10^{-6} for the RL stage.



(a) Training reward (b) Validation accuracy

Figure 6: Single-domain RLVR on DAPO-Math-17k [72]: (a) training reward and (b) validation accuracy (AIME 2024 mean@1) vs. training step. GRPO collapses after step 60. GSPO stagnates. FiberPO improves steadily on both metrics.

Figure 7 provides training diagnostics that can be interpreted through FiberPO's theoretical framework. GRPO's entropy collapses to 0.038 nats (a 91% reduction from initialization) and its mean importance ratio exceeds 10^3. These observations are consistent with the structural gap identified in Section 3.3.1: since GRPO gates each token's ratio r_i independently without bounding trajectory-level aggregate drift, once the trajectory-level drift exceeds the per-token clip bound, the majority of tokens in a trajectory can saturate simultaneously, destroying token-level discrimination. This plausibly triggers a feedback loop in which imprecise updates accelerate further drift. A contributing factor is that GRPO clips the absolute log-ratio $\log r_i$ rather than a fiber residual: the clip threshold is shared between trajectory-level drift and token-level variation, so trajectory drift consumes the clip budget and forces tokens into saturation even when their *within-trajectory* deviations are small. Additionally, GRPO lacks a restorative gradient: when a token



(a) Policy entropy (b) Mean ratio $\bar{r}$ (c) Mean response length

(d) Gradient norm (e) Local residual vs. ε (FiberPO)

Figure 7: Training diagnostics for the single-domain DAPO math run. Top row (comparative, all three methods): (a) policy entropy, (b) mean importance ratio on log scale, (c) mean response length. Bottom row (FiberPO-specific): (d) gradient norm on log scale, (e) fiber residual and token-level clip fraction (both in the safe zone throughout training).

exceeds the clip, its gradient is zeroed rather than reversed, providing no mechanism to oppose drift. GSPO exhibits the complementary failure mode. Its entropy diverges to 1.99 nats with response lengths remaining between 7,380 and 8,870 tokens: by collapsing each trajectory to a single aggregate ratio, GSPO suppresses within-trajectory variation and prevents the optimizer from distinguishing token-level quality differences within the same trajectory. The elevated entropy suggests diffuse probability mass rather than concentration on efficient solution paths.

FiberPO addresses both failure modes through its two-scale decomposition. It preserves entropy at 0.43 nats throughout training while improving validation accuracy from 0.668 to 0.786. Its mean importance ratio mostly remains at 1.13, near the on-policy value of 1, consistent with the first-order agreement property (first-order agreement, property 2 in [73]): the FiberPO Jacobian reduces to the identity at on-policy, so each update step provides an accurate gradient direction when the policy has not drifted far. The scale separation property (property 3 in [73]) further suggests that near on-policy, the per-token local gradient dominates with $O(1)$ magnitude while trajectory-level corrections contribute at most $O(1/T_\tau)$ per token, engaging primarily only as drift grows. Unlike GRPO, the logclip acts on the fiber residual $u_i = l_i \log r_i - \log s_\tau^{(l_i)}$ (Eq. 5), separating the clip budget from trajectory-level drift so that tokens with small within-trajectory deviations retain their full gradient signal. The fiber residual and token-level clip fraction (Figure 7e) remain in the safe zone throughout training, confirming that most tokens satisfy $|u_i| < \epsilon$ and stay in the unsaturated L-I regime. Unlike GSPO, the per-token gated residual $\tilde{r}_i^{\text{fiber}}$ preserves within-trajectory discrimination, allowing the optimizer to directly receive and update according to individual token contributions. The gradient norm supports this interpretation: FiberPO's norm increases only $1.5\times$ over 100 steps (0.033 to 0.049), compared to $12\times$ for GRPO and $7.5\times$ for GSPO.

The *token-efficiency property* (Section 3.3.1) offers a plausible explanation for the joint behavior of response length, entropy, and validation accuracy. FiberPO decreases mean response length from 7,902 to 4,543 tokens while simultaneously increasing validation accuracy and preserving entropy. Because the logclip acts on the fiber residual $u_i = l_i \log r_i - \log s_\tau^{(l_i)}$ rather than on $\log r_i$ directly, tokens that shift in concert with trajectory-level drift pass through unattenuated, retaining their full gradient signal. The optimizer therefore receives discriminative per-token directions even under moderate trajectory-level drift, which may favor concise, correct reasoning paths over verbose ones. GRPO also shortens responses (7,904 to 2,216 tokens at step 100), but the accompanying entropy collapse and accuracy degradation suggest degenerate compression rather than efficiency: once trajectory-level drift exceeds the clip bound

and token-level discrimination is lost, the model can no longer distinguish valid from invalid tokens. GSPO maintains high response lengths with high variance, consistent with its suppressed per-token signal preventing concentration of probability on efficient solution paths.

3.3.3 Multi-Domain Extension

Single-domain RL training commonly degrades capabilities outside the trained domain: a model fine-tuned exclusively on mathematics may lose instruction-following or coding ability. Multi-domain training addresses this by optimizing across diverse environments simultaneously, preserving existing capabilities while gaining on the trained domains. However, mixing heterogeneous reward distributions intensifies the stability challenge, since trajectory-level drift statistics vary across domains. FiberPO's compositional two-scale gating is well suited to this setting: the trajectory-level gate maintains per-trajectory trust regions regardless of which domain the trajectory belongs to, while the token-level gate preserves fine-grained credit assignment within each domain's distinct reward structure.

We compose our training data from several domains, such as coding agent, math, knowledge, instruction following, language, and so on. All domains supply verifiable reward signals, making the full blend suitable for RLVR without learned reward models.

Following the Gaussian curriculum strategy introduced in [35], we progressively shift training from easier to harder prompts. Prior to training, every prompt is profiled with $K{=}10$ rollout samples from the DPO checkpoint to obtain an empirical pass rate $p_i \in [0, 1]$. Prompts with $p_i = 1$ (already solved) are filtered out. At training step t the target difficulty is parameterized by a Gaussian mean

$$\mu_t = \mu_0 + (\mu_T - \mu_0)\,\frac{t}{T}, \tag{6}$$

which decays linearly from $\mu_0 = 0.8$ (easy) to $\mu_T = 0.2$ (hard) over T total steps Each prompt receives a sampling weight

$$w_i^{(t)} = \exp\Big(-\tfrac{1}{2}\big(\tfrac{p_i-\mu_t}{\sigma}\big)^2\Big), \qquad \sigma = 0.15, \tag{7}$$

concentrating the batch around the current target difficulty.

To preserve domain balance across all heterogeneous environments, we extend the flat Gaussian sampling with a two-level domain-balanced scheme. At each draw, a domain group g is first selected with probability α_g, then a prompt is drawn within g with probability proportional to $w_i^{(t)}$. Unspecified group weights default to the square-root-proportional heuristic $\alpha_g \propto \sqrt{N_g}$, where N_g is the number of valid prompts in group g, with manual overrides for selected domains.

We reuse the training protocol and FiberPO hyperparameters from our single-domain baseline without introducing any multi-domain-specific tuning. Despite this lack of targeted adjustment, FiberPO achieves stable gains across all domains without exhibiting catastrophic degradation in any single area, confirming that the per-trajectory trust regions employed by FiberPO inherently generalize across heterogeneous reward distributions

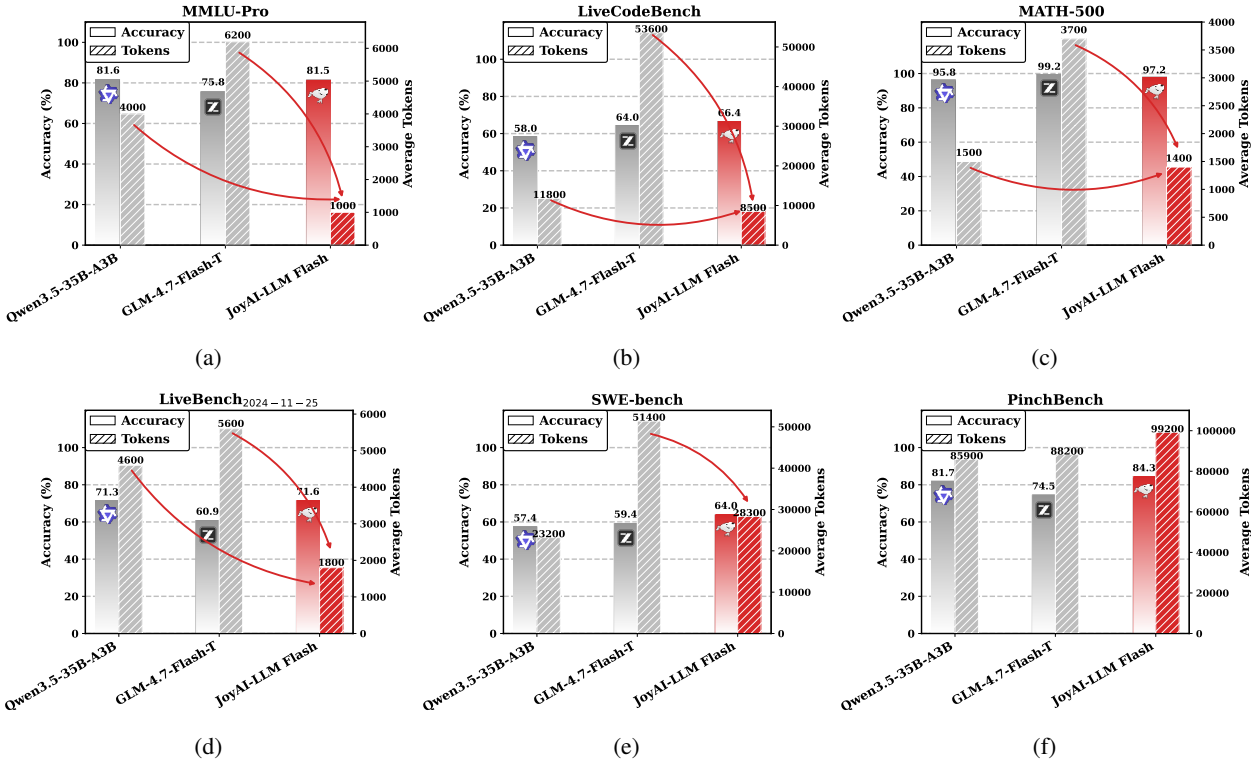
3.4 Instruct Model Evaluation

To provide a comprehensive assessment of JoyAI-LLM Flash, we evaluate the model on a diverse set of widely used LLM benchmarks covering multiple capabilities, including

- **General Knowledge**: MMLU [45], MMLU-Pro [46], HellaSwag [77], CMMLU [47], C-Eval [78], GPQA-Diamond [79], SuperGPQA [80].
- **Math Reasoning**: MATH-500 [49], AIME′25.
- **Coding Ability**: HumanEval [50], LiveCodeBench v6 [51], SWE-bench Verified [81].
- **Instruction Following**: AlignBench [82], IFEval [83].
- **Long-context Ability**: RULER [52].
- **General Tasks**: $\text{LiveBench}_{\text{2024-11-25}}$ [84].
- **Agent & OpenClaw**: τ^2-Bench [85], PinchBench.

The experimental results are shown in Table 3, where Qwen3-Next-80B-A3B, Qwen3.5-35B-A3B, and Qwen3-30B-A3B are the baseline instruct models. We also include GLM-4.7-Flash-Thinking for comparison, as its reasoning capabilities provide a more direct alignment with our JoyAI-LLM Flash than its standard instruct counterpart.

As can be observed, our JoyAI-LLM Flash achieves remarkable token efficiency. Specifically, on the LiveCodeBench, JoyAI-LLM Flash surpasses GLM-4.7-Flash-Thinking by 2.4% accuracy with a 85% reduction in token usage. Beyond efficiency, JoyAI-LLM Flash also demonstrates strong competitive performance across diverse tasks, including mathematics and long-context understanding. The token efficiency of JoyAI-LLM Flash is illustrated in Figure 8. Specifically, the accuracy (left bar) and token usage (right bar) are compared across different models, highlighting our efficiency gains. Notably, although JoyAI-LLM Flash consumes more tokens on PinchBench, it achieves the best accuracy compared to other models.



(a) (b) (c)

(d) (e) (f)

Figure 8: Comparison of model performance (left bars) and token consumption (right bars) across six benchmarks. Qwen3.5-35B-A3B and JoyAI-LLM Flash are instruct models, "GLM-4.7-Flash-T" refers to GLM-4.7-Flash-Thinking, which is included due to its comparable performance.

Table 3: Comparison with baseline models. Qwen3-Next-80B-A3B, Qwen3.5-35B-A3B, and Qwen3-30B-A3B are instruct models; "GLM-4.7-Flash-T" refers to GLM-4.7-Flash-Thinking, which is additionally included due to its more comparable performance. Results marked with asterisk* are directly cited from original papers and differ substantially from our reproduced results.

Task	Qwen3-Next-80B-A3B		Qwen3.5-35B-A3B		Qwen3-30B-A3B		GLM-4.7-Flash-T		JoyAI-LLM Flash	
	Acc	#Token	Acc	#Token	Acc	#Token	Acc	#Token	Acc	#Token
General Knowledge										
MMLU	86.4	400	**91.2**	900	88.0	**200**	88.2	2000	88.8	**200**
MMLU-Pro	76.7	1500	**81.6**	4000	78.7	1600	75.8	6200	81.5	**1000**
HellaSwag	88.1	**<100**	89.1	**<100**	86.2	**<100**	71.5	3000	**91.7**	**<100**
CMMLU	89.1	500	**89.5**	600	87.1	500	80.2	4900	86.7	**400**
C-Eval	89.2	700	**91.5**	800	87.8	800	77.0	12200	88.8	**600**
GPQA-Diamond	73.0	4000	74.2	4300	66.2	4100	**76.7**	19900	72.7	**2500**
SuperGPQA	60.0	1900	**62.0**	3700	53.0	2000	41.0	8400	55.0	**1500**
Math										
MATH-500	97.4	1700	95.8	1500	97.6	**1400**	**99.2**	3700	97.2	**1400**
AIME′25	70.4	6900	56.7	7000	63.8	6100	**92.1**	26000	71.6	**5300**
Coding										
HumanEval	**95.1**	600	93.9	**300**	92.1	400	93.9	7200	94.5	800
LiveCodeBench v6	58.8	**8500**	58.0	11800	48.1	15900	64.0*	53600	**66.4**	**8500**
SWE-bench Verified	31.2	25400	57.4	23200	29.0	**16400**	59.4*	51400	**64.0**	28300
Instruction Following										
AlignBench	**8.3**	800	8.1	1000	7.9	1200	6.9	3600	8.1	**700**
IFEval	84.7	600	81.8	1000	80.8	**500**	**85.4**	1900	79.1	**500**
Long-Context										
RULER	94.2	100	**96.0**	**<100**	93.7	**<100**	74.7	8300	95.7	**<100**
General Tasks										
LiveBench$_{2024\text{-}11\text{-}25}$	**75.9**	2300	71.3	4600	68.5	2700	60.9	5600	71.6	**1800**
Agent & OpenClaw										
τ^2-Bench	38.4	2200	**76.6**	2800	31.6	**1900**	69.9	3400	72.2	3700
PinchBench	83.7	107700	81.7	**85900**	67.0	210400	74.5	88200	**84.3**	99200

4 Inference

The architecture of JoyAI-LLM Flash deliberately adopts a compact parameter scale of 48 billion together with a highly sparse Mixture-of-Experts (MoE) structure. Meanwhile, we employ a co-design of training and inference optimizations, including Quantization-Aware Training (QAT) and dense Multi-Token Prediction (MTP). Furthermore, we evaluate the inference throughput performance across various context lengths under the prefill–decode disaggregation setting.

4.1 Quantization

JoyAI-LLM Flash adopts both Quantization-Aware Training (QAT) and Post-Training Quantization (PTQ) to achieve the optimal trade-off between model accuracy and throughput. All quantized models are open-sourced on HuggingFace.

During the QAT phase, JoyAI-LLM Flash simulates INT4 quantization during training by inserting fake-quantization operators (Quantize $\rightarrow$ DeQuantize $\rightarrow$ Quantize) and keeping weights in high precision for stable optimization. To cope with non-differentiable operations such as rounding and clamping, we adopt Straight-Through Estimators (STE) [86] in backpropagation, allowing gradients to flow to the master weights. This aligns with the practice of mainstream LLMs such as Kimi-K2-Thinking [87] and GLM-5 [88]. A beneficial side effect is the stabilization of the reinforcement learning stage: INT4 quantization fosters more robust model rollouts and reduces noise diversity by narrowing the numerical space. JoyAI-LLM Flash thus maintains high accuracy even when applying the simple Round-To-Nearest quantization scheme at lower bit widths.

During the PTQ phase, we compare JoyAI-LLM Flash with Qwen3-30B-A3B BF16 baseline under BF16, FP8, and W4AFP8 [89] quantization in Figure 9. All the experiments were conducted using vLLM [90] and TRT-LLM [91]. We reported the best out of the two for each model. Model accuracy is evaluated by the mean accuracy across three distinct domain datasets: MATH-500[49], GPQA [79], and MBPP [92]. The results indicate that JoyAI-LLM Flash architecture consistently outperforms the Qwen baseline in terms of accuracy and throughput. Qwen3-30B-A3B FP8 model yields a throughput gain of 10%, while suffering noticeable accuracy degradation. In contrast, JoyAI-LLM Flash FP8 model improves throughput by 17% with nearly no accuracy degradation. Moreover, the W4AFP8 model maximizes the throughput gain of nearly 28% over the baseline with a slight accuracy drop of 1.2%. These findings demonstrate that JoyAI-LLM Flash achieves an optimal trade-off between accuracy and efficiency, even with 1.63× larger model weights than the Qwen baseline.

Furthermore, to accommodate model usage on edge devices, we also released the effective low-bit GGUF [93] variants of JoyAI-LLM Flash. Inspired by the practices of NVIDIA NVFP4 quantization and GGUF's K-Quants, we propose a "DoubleQuant" strategy tailored for less sensitive weights: partition the weight matrix into super blocks, apply the aforementioned quantization method within each block, perform a global-like quantization across blocks and store the double-quantized scales at lower precision (e.g., 6-bit or 8-bit). Experimental results show that this DoubleQuant strategy achieves comparable accuracy to the BF16 baseline, validating the effectiveness of the proposed approach.

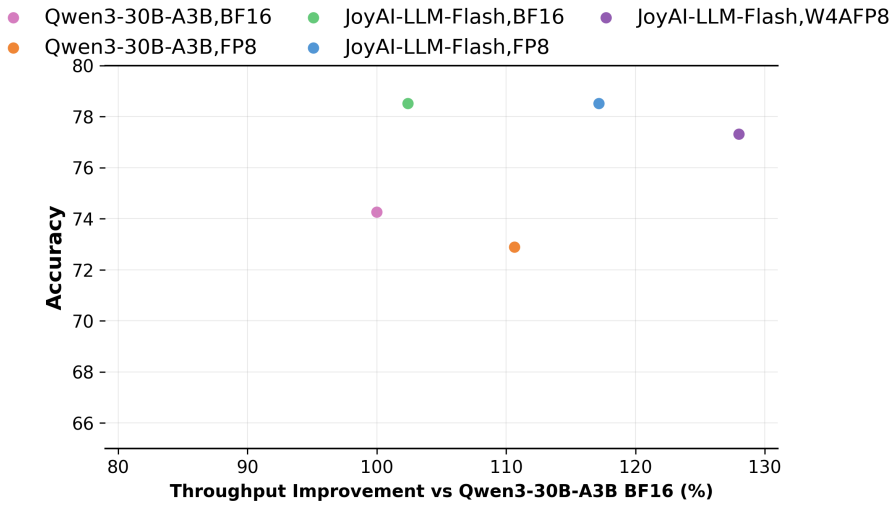


Figure 9: Comparison of Accuracy and Throughput for Quantized Models: JoyAI-LLM Flash vs. Qwen3-30B-A3B. The accuracy is measured by the mean accuracy across three distinct domain datasets: MATH-500, GPQA, and MBPP.

4.2 Multi-Token Prediction

JoyAI-LLM Flash features a lightweight dense MTP architecture, achieving state-of-the-art speedup despite a medium acceptance length. Table 4 evaluates the MTP performance of JoyAI-LLM Flash on SpechBench [94] under the MTP 3 layers and concurrency 64 configuration. The performance is evaluated by the acceptance length, ratio and speedup over the non-MTP counterpart. We compare JoyAI-LLM Flash with a suite of MTP-optimized LLMs, including Qwen3.5-35B-A3B [6], Step-3.5-Flash [7], MiMo-V2-Flash [15], GLM-5 [95], GLM-4.7-Flash [4], DeepSeek-V3.2 [36], and DeepSeek-V3 [8]. JoyAI-LLM Flash achieves the highest speedup of 1.87×, representing a 3% improvement over the closest competitor, GLM-5 (1.82×), and a 72% improvement over the slowest model, GLM-4.7-Flash (1.09×).

Table 4: SpecBench MTP-3 Speculative Decoding Performance. Best results are marked in bold.

Model	GPU	Speedup	Acceptance		TPS	
			length	ratio	user	server
JoyAI-LLM Flash	1	**1.87×**	2.20	40.35	66	4241
GLM-5 [95]	8	1.82×	3.03	**75.84**	31	1969
DeepSeek-V3.2 [36]	8	1.79×	2.55	63.72	31	1958
Qwen3.5-35B-A3B [6]	1	1.61×	**3.18**	72.56	**85**	**5428**
MiMo-V2-Flash [15]	4	1.61×	2.69	67.22	47	3033
Step-3.5-Flash [7]	4	1.39×	2.21	55.23	48	3048
DeepSeek-V3 [8]	8	1.21×	2.71	56.86	25	1608
GLM-4.7-Flash [4]	1	1.09×	2.11	36.84	26	1695

To better align with user habits, we integrate MTP with quantization and conduct joint optimization. The overall performance of MTP is evaluated by the *next-n* configuration. We evaluate inference throughput across three quantization formats—BF16, FP8, and W4AFP8—on a randomly sampled dataset with ISL=1K and OSL=2K. All quantization settings were deployed with TensorRT-LLM [91], integrated with CUDA Graph, Torch Compile, and Overlap Scheduling techniques. As shown in Figure 10, both BF16 and FP8 model achieved optimal throughput under the MTP setting of *next-n*=3. Compared to the baseline configuration (BF16 with *next-n*=0), these settings delivered a throughput acceleration of 1.57× and 1.81×, respectively. The overall best performance was observed with W4AFP8 quantization at *next-n*=2, yielding a 1.96× speedup over the same baseline. The experiments indicate that excessively high MTP layers (e.g., *next-n*>3) could introduce diminishing or even negative returns, which is attributed to the additional computational overhead incurred in predicting multiple future tokens.

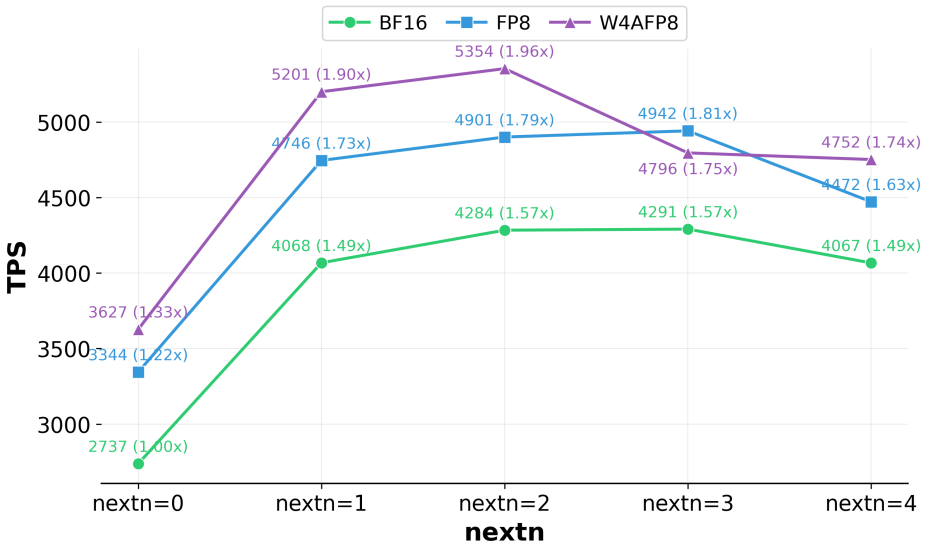


Figure 10: Joint Optimization of MTP and Quantization (ISL/OSL = 1K/2K, Concurrency = 64)

4.3 Serving and Scheduling

Modern LLM inference workloads exhibit highly variable sequence lengths, bursty arrival patterns, and drastically distinct compute–memory profiles between prefill and decode stages. And the serving efficiency is usually workload-dependent: short-context requests are governed by the trade-off of Time To First Token (TTFT) and Time Per Output Token (TPOT), whereas long-context requests are prefill-dominated and benefit more from cross-request prefix reuse. We therefore evaluate JoyAI-LLM Flash under these two distinct regimes.

Short-context workloads such as interactive chat typically prioritize user interaction experience and use TTFT and TPOT as the key performance metrics. We build a workload grid with input lengths ranging from 128 to 2048 tokens and output lengths of 128 and 512 tokens. Request rates were varied from 1 to 8 requests per second (RPS). JoyAI-LLM Flash maintains excellent responsiveness across all tested scales. To achieve comparable or better performance gains in real-world environments, we provide the following deployment insights based on our practical experience:

- **Intra-node PD Improves overall Performance:** For smaller models like JoyAI-LLM Flash, we recommend colocating prefill and decode instances on the same node to minimize communication overhead.
- **Dynamic PD Adapts well to Real-time Workloads Variations:** Performance simulation tools such as AIConfigurator [96] enable runtime dynamic scaling of PD instances according to SLA targets.

Long-context workloads such as Retrieval-Augmented Generation (RAG) and multi-turn agent tasks typically require large KV cache capacities. However, limited memory frequently triggers KV cache eviction, which induces redundant recomputation and restricts cross-request KV reuse. This constraint poses severe challenges for latency-sensitive metrics such as TTFT and degrades user experience. To simulate this workload, we randomly generated data with no prefix reuse, featuring input lengths of 20,000 tokens and output lengths of 100 tokens. Request rates were varied from 0.25 to 3.0 requests per second (RPS). Experiments were conducted with a Prefill-Decode (PD) disaggregated deployment. And Mooncake [97] was employed as a centralized KV cache store to manage cache across requests. Also, we provide the following deployment insights under this scenario:

- **PD Disaggregation Delivers Better Flexibility:** Compared with aggregated deployment, PD disaggregation supports independent scaling of prefill and decode, and centralized KV caching lets us tune their instance ratio to better match workload demands.
- **Choose KV Cache Management Wisely:** Two mainstream centralized KV cache management schemes exist: remote KV pooling and peer-to-peer (P2P) CPU sharing. Selection depends on hardware infrastructure, with the goal of minimizing data transfer overhead. Across both approaches, we strongly discourage TCP as the data-plane transport due to its high latency and overhead.
- **Choose the Appropriate Cache Write Strategy:** The write strategy for KV cache should account for available bandwidth. When bandwidth is sufficient, we recommend writing to the distributed cache layer immediately upon each cache hit to maximize hit rates. While in scenarios with constrained I/O bandwidth, we recommend reducing write frequency and using an eviction policy to preserve hot data.
- **Trade-off Between Recomputation and Transfer:** Smaller models exhibit higher sensitivity to data transfer overhead, even when KV caches are exchanged via P2P RDMA between instances. Transfer latency can outweigh recomputation cost, making centralized KV cache management a net-negative optimization. Careful evaluation is required to identify the crossover point for a given model and hardware setup.

5 Conclusion and Future Work

We present JoyAI-LLM Flash, a state-of-the-art medium-sized instruct language model with 3 billion activated parameters and 48 billion total parameters. JoyAI-LLM Flash was pretrained on 20 trillion text tokens using Muon optimizer, followed by large-scale supervised fine-tuning (SFT), direct preference optimization (DPO), and reinforcement learning (RL) across diverse environments. JoyAI-LLM Flash achieves strong performance across frontier knowledge, reasoning, coding tasks, and agentic capabilities. Moving forward, we aim to extend the model's paradigm by integrating continual learning and persistent memory, enabling the LLM to dynamically adapt and retain knowledge over time.

6 Contribution

Project Leaders Chao Xue, Xiaodong He†

Core Contributors Bo Zhang, Bohua Cai, Chang Li, Chao Xue, Dongkai Liu, Guoqiang Huang, Jialong Shi, Liang Huang, Ming Ke, Panfeng Shi, Qi Wang, Qiaoqiao Yuan, Qiong Cao, Qixiang Wang, Rongcheng Bian, Shi Suo, Shijie Ren, Shijin Zhang, Shiying Fan, Shuai Xie, Tianyi Zhang, Wei Liu, Wentao Tan, Xiaodong He, Xuyang Peng, Ya Zhang, Yifei Liu, Yinhao Bai, Yuqi Zhang, Yuesong Zhang, Zhenfang Wang

Contributors Aichen Cai, Anmeng Zhang, Anyu Li, Changjian Jiang, Changkai Lu, Chaocai Liang, Cheng Zhang, Fei Wang, Haijian Ke, Han Lin, Hao Wang, Ji Miao, Jiacheng Zhang, Jifeng Zhu, Jingjing Qian, Junhui Luo, Junwu Xiong, Lam So, Mingyang Li, Peng Hao, Qian Lai, Qingyu Yin, Rongduo Han, Shaoqiang Zheng, Shi Hu, Xianghan Meng, Xing Pan, Xiran Wang, Yanxu Chen, Yang Liu, Yangyang Duan, Yicheng Gong, Yidan Huang, Yongqiang Liu, Zerui Xie, Zhennan Shen, Zheyuan Liu, Zhuwei Zeng

†Corresponding Author

References

[1] Zheng Du, Hao Kang, Song Han, Tushar Krishna, and Ligeng Zhu. Ockbench: Measuring the efficiency of llm reasoning. *arXiv:2511.05722*, 2026.

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

[3] Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

[4] GLM Team. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models, 2025.

[5] Qwen Team. Qwen3-30b-a3b-instruct-2507, July 2026.

[6] Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026.

[7] Ailin Huang, Ang Li, et al. Step 3.5 flash: Open frontier-level intelligence with 11b active parameters, 2026.

[8] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv:2412.19437*, 2024.

[9] Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

[10] Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.

[11] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[12] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941. PMLR, 2017.

[13] Lean Wang, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. Auxiliary-loss-free load balancing strategy for mixture-of-experts. *arXiv preprint arXiv:2408.15664*, 2024.

[14] Keller Jordan, Yuchen Jin, Vlado Boza, You Jiacheng, Franz Cecista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. 6.

[15] Bangjun Xiao, Bingquan Xia, Bo Yang, Bofei Gao, Bowen Shen, Chen Zhang, Chenhong He, Chiheng Lou, Fuli Luo, Gang Wang, et al. Mimo-v2-flash technical report. *arXiv preprint arXiv:2601.02780*, 2026.

[16] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[17] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

[18] Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, and Phil Gibbons. Pipedream: Fast and efficient pipeline parallel dnn training, 2018. *URL https://arxiv. org/abs*, 1806.

[19] Joel Lamy-Poirier. Breadth-first pipeline parallelism. *Proceedings of Machine Learning and Systems*, 5:48–67, 2023.

[20] Ziming Liu, Shenggan Cheng, Haotian Zhou, and Yang You. Hanayo: Harnessing wave-like pipeline parallelism for enhanced large model training efficiency. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13, 2023.

[21] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pages 1–15, 2021.

[22] Penghui Qi, Xinyi Wan, Guangxing Huang, and Min Lin. Zero bubble pipeline parallelism. *arXiv preprint arXiv:2401.10241*, 2023.

[23] NVIDIA. Moe a2a interleaved 1f1b based computation and communication overlap. *https://developer.nvidia.com/zh- cn/blog/1f1b- moe- a2a- computing- overlap/*, 2025.

[24] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

[25] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

[26] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 68658–68685. Curran Associates, Inc., 2024.

[27] Guilherme Penedo, Hynek Kydlíček, Alessandro Cappelli, Mario Sasko, and Thomas Wolf. Datatrove: large scale data processing, 2024.

[28] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, page 604–613, New York, NY, USA, 1998. Association for Computing Machinery.

[29] A.Z. Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*, pages 21–29, 1997.

[30] Anton Lozhkov and Raymond Liand others. Starcoder 2 and the stack v2: The next generation, 2024.

[31] Qwen Team. Qwen2.5 technical report, 2025.

[32] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. Qwen2.5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

[33] Kazuki Fujii, Yukito Tajima, Sakae Mizuki, Hinari Shimada, Taihei Shiotani, Koshiro Saito, Masanari Ohi, Masaki Kawamura, Taishi Nakamura, Takumi Okamoto, Shigeki Ishida, Kakeru Hattori, Youmi Ma, Hiroya Takamura, Rio Yokota, and Naoaki Okazaki. Rewriting pre-training data boosts llm performance in math and code, 2025.

[34] Olmo Team. Olmo 3, 2025.

[35] NVIDIA Team. Nemotron 3 nano: Open, efficient mixture-of-experts hybrid mamba-transformer model for agentic reasoning, 2025.

[36] DeepSeek-AI. Deepseek-v3.2: Pushing the frontier of open large language models, 2025.

[37] Junbo Niu, Zheng Liu, et al. Mineru2.5: A decoupled vision-language model for efficient high-resolution document parsing, 2025.

[38] Haoran Wei, Yaofeng Sun, and Yukun Li. Deepseek-ocr: Contexts optical compression. *arXiv preprint arXiv:2510.18234*, 2025.

[39] Xintong Hao, Ruijie Zhu, Ge Zhang, Ke Shen, and Chenggang Li. Reformulation for pretraining data augmentation, 2025.

[40] Dan Su, Kezhi Kong, Ying Lin, Joseph Jennings, Brandon Norick, Markus Kliegl, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-cc: Transforming common crawl into a refined long-horizon pretraining dataset. *ArXiv*, abs/2412.02595, 2024.

[41] Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, et al. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024.

[42] Jack W. Rae, Sebastian Borgeaud, et al. Scaling language models: Methods, analysis & insights from training gopher. *CoRR*, abs/2112.11446, 2021.

[43] Jordan Hoffmann, Sebastian Borgeaud, et al. Training compute-optimal large language models. *CoRR*, abs/2203.15556, 2022.

[44] Jared Kaplan, Sam McCandlish, et al. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020.

[45] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

[46] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290, 2024.

[47] 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. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11260–11285, 2024.

[48] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[49] 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. *arXiv preprint arXiv:2103.03874*, 2021.

[50] Mark Chen. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[51] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

[52] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

[53] OpenAI. gpt-oss-120b & gpt-oss-20b model card, 2025.

[54] Chao Xue, Wei Liu, et al. Omniforce: On human-centered, large model empowered and cloud-edge collaborative automl system. *nature npj-ai*, 2023.

[55] John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents. In *Proceedings of the 39th Annual Conference on Neural Information Processing Systems (NeurIPS 2025 D&B Spotlight)*, 2025. arXiv:2504.21798, accepted at NeurIPS 2025 (Spotlight).

[56] Xingyao Wang, Boxuan Li, et al. OpenHands: An Open Platform for AI Software Developers as Generalist Agents, 2024.

[57] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[58] Open-R1 Team. Openr1-math-220k dataset, 2025. Accessed: 2025-03-06.

[59] Wei Du, Shubham Toshniwal, Branislav Kisacanin, Sadegh Mahdavi, Ivan Moshkov, George Armstrong, Stephen Ge, Edgar Minasyan, Feng Chen, and Igor Gitman. Nemotron-math: Efficient long-context distillation of mathematical reasoning from multi-mode supervision. *arXiv preprint arXiv:2512.15489*, 2025.

[60] Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625, 2020.

[61] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

[62] Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5687–5711, 2023.

[63] Jason Wei, Nguyen Karina, Hyung Won Chung, Yunxin Joy Jiao, Spencer Papay, Amelia Glaese, John Schulman, and William Fedus. Measuring short-form factuality in large language models. *arXiv preprint arXiv:2411.04368*, 2024.

[64] Satyapriya Krishna, Kalpesh Krishna, Anhad Mohananey, Steven Schwarcz, Adam Stambler, Shyam Upadhyay, and Manaal Faruqui. Fact, fetch, and reason: A unified evaluation of retrieval-augmented generation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4745–4759, 2025.

[65] Junting Zhou, Wang Li, Yiyan Liao, Nengyuan Zhang, Tingjia Miao, Zhihui Qi, Yuhan Wu, and Tong Yang. Scholarsearch: Benchmarking scholar searching ability of llms. *arXiv preprint arXiv:2506.13784*, 2025.

[66] Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*, 2023.

[67] Dingfeng Shi, Jingyi Cao, Qianben Chen, Weichen Sun, Weizhen Li, Hongxuan Lu, Fangchen Dong, Tianrui Qin, King Zhu, Minghao Liu, et al. Taskcraft: Automated generation of agentic tasks. *arXiv preprint arXiv:2506.10055*, 2025.

[68] Dhruv Nathawani, Igor Gitman, Somshubra Majumdar, Evelina Bakhturina, Ameya Sunil Mahabaleshwarkar, , Jian Zhang, and Jane Polak Scowcroft. Nemotron-Post-Training-Dataset-v1, 2025.

[69] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

[70] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[71] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[72] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

[73] Chang Li, Tshihao Tsu, Yaren Zhang, Chao Xue, and Xiaodong He. Fibration policy optimization. *arXiv preprint arXiv:2603.08239*, 2026.

[74] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897. PMLR, 2015.

[75] Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, et al. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

[76] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv:2409.19256*, 2024.

[77] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

[78] Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Yao Fu, et al. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. *Advances in Neural Information Processing Systems*, 36:62991–63010, 2023.

[79] 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. In *First Conference on Language Modeling*, 2024.

[80] M-A-P Team. Supergpqa: Scaling llm evaluation across 285 graduate disciplines, 2025.

[81] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024.

[82] Xiao Liu, Xuanyu Lei, et al. AlignBench: Benchmarking Chinese alignment of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pages 11621–11640, 2024.

[83] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

[84] Colin White, Samuel Dooley, et al. Livebench: A challenging, contamination-free LLM benchmark. In *The Thirteenth International Conference on Learning Representations*, 2025.

[85] Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment, 2025.

[86] Penghang Yin, Jiancheng Lyu, Shuai Zhang, Stanley Osher, Yingyong Qi, and Jack Xin. Understanding straight-through estimator in training activation quantized neural nets. *arXiv preprint arXiv:1903.05662*, 2019.

[87] Moonshot AI. Kimi-k2-thinking. Hugging Face model card, 2025. Accessed: 2026-03-04.

[88] GLM-5-Team, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, et al. Glm-5: from vibe coding to agentic engineering, 2026. Accessed: 2026-03-04.

[89] NVIDIA. Model optimizer quantization support matrix, 2025. GitHub repository.

[90] vLLM Project. vllm: Easy, fast, and cheap llm serving, 2023. Accessed: 2026-02-11.

[91] NVIDIA. Tensorrt-llm, 2024. GitHub repository, tag: v0.xx.x, accessed: 2026-02-11.

[92] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[93] StyMaar. Readme: Gguf, 9 2025. GitHub documentation.

[94] Heming Xia, Zhe Yang, et al. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics ACL 2024*, pages 7655–7671, Bangkok, Thailand and virtual meeting, August 2024. Association for Computational Linguistics.

[95] GLM-5-Team. Glm-5: from vibe coding to agentic engineering, 2026.

[96] Dynamo Project. Offline optimization of your disaggregated dynamo graph, 2023. Accessed: 2026-02-11.

[97] Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: A kvcache-centric disaggregated architecture for llm serving. *arXiv preprint arXiv:2407.00079*, 2024.