GigaChat

GigaChat 4: Technical Report

GigaChat Team

Abstract

We present GigaChat 4, a hybrid mixture-of-experts language model with 708B total parameters (41B activated) and a native context length of one million tokens. The model combines Gated DeltaNet layers with multi-head latent attention and uses DeepSeek Sparse Attention for long-context processing. We use smaller proxy models to select training settings and stability controls. Our data pipeline combines filtered web text, documents, and code with synthetic tasks for reasoning and tool use. After supervised fine-tuning, we train specialist models with reinforcement learning. Multi-teacher on-policy distillation then combines their capabilities in a single model with Reasoning and Instant modes. Under our evaluation protocol, GigaChat 4 scores 71.26% on Terminal-Bench 2.0 in Reasoning mode and 60.47% in Instant mode. The corresponding GigaChat 3.5 scores are 30.30% and 12.94%. We also describe FP8 training, asynchronous post-training, optimized attention kernels, and speculative decoding. The model checkpoints are available at https://huggingface.co/ai-sage/GigaChat4-708B-A41B.

Contents

1 Introduction and Model Overview **4**
- 1.1 Scope and Main Contributions . . . 4
- 1.2 Architecture and Model Configuration . . . 4

2 Pretraining **6**
- 2.1 Data . . . 6
- 2.2 Training Recipe . . . 7
- 2.3 Architectural Stability Controls . . . 7
- 2.4 Monitoring and Interventions . . . 9
- 2.5 Infrastructure . . . 9

3 Midtraining **10**
- 3.1 Data . . . 10
- 3.2 Training Recipe . . . 11

4 Context Extension **11**
- 4.1 Data . . . 11
- 4.2 Training Recipe . . . 12
- 4.3 Long-Context Architecture . . . 12
- 4.4 Selected Configuration . . . 12
- 4.5 Monitoring and Checkpoint Selection . . . 13
- 4.6 Infrastructure . . . 13

5 Alignment **14**
- 5.1 Cold Start . . . 14
- 5.2 Online Reinforcement Learning . . . 15
- 5.3 Multi-Teacher On-Policy Distillation . . . 19
- 5.4 Infrastructure . . . 21

6 Inference **23**
- 6.1 Serving Stack . . . 23
- 6.2 Inference Optimizations . . . 23
- 6.3 Speculative Decoding . . . 24
- 6.4 Performance Evaluation . . . 25

7 Evaluation **26**
- 7.1 Evaluation Protocol . . . 26
- 7.2 Reasoning Results . . . 26
- 7.3 Instant Results . . . 27

8 Conclusion **29**

A Pretraining Data **29**
- A.1 Sources and Mixture . . . 29
- A.2 Organic and Synthetic Data . . . 29
- A.3 Web Data . . . 30
- A.4 Multilingual Data . . . 32
- A.5 PDF Documents . . . 32
- A.6 Reference Sources . . . 33

A.7 Source Code . . . 33
A.8 Deduplication and Decontamination . . . 33
A.9 Data Validation Experiments . . . 34
A.10 Infrastructure . . . 35

B Midtraining Data **36**
B.1 Data Preparation . . . 36
B.2 Data Synthesis Pipelines . . . 36
B.3 Midtraining Data Recipe Development . . . 37

C Long-Context Data **37**
C.1 Organic Long Documents . . . 37
C.2 Repository-Level Code . . . 38
C.3 Synthetic Question Answering . . . 38
C.4 Long-Context Data Recipe Development . . . 39

D Architecture and Context-Extension Ablations **39**
D.1 Pretraining Stability and Architecture . . . 39
D.2 Context Extension and DeepSeek Sparse Attention . . . 42

E Cold-Start Data Sources and Domain-Specific Pipelines **44**
E.1 Data Source Categories . . . 44
E.2 Data Composition by Domain . . . 44
E.3 Validation Tools . . . 45
E.4 Data Generation Pipelines . . . 45
E.5 Data Selection and 37B Baselines . . . 46
E.6 SWE and Terminal Data . . . 47

F Online RL and Distillation **48**
F.1 Online-RL Environment Construction . . . 48
F.2 RL Grading and Training Protocols . . . 49
F.3 RL Stability . . . 55
F.4 MoE Load Imbalance during RL . . . 55
F.5 MOPD Configuration and Experiments . . . 57

G Alignment Infrastructure **62**
G.1 Asynchronous Runtime . . . 62
G.2 Sparse-Attention Training . . . 63
G.3 Gated DeltaNet Training . . . 63
G.4 Rollout Cache Policy and Additional Optimizations . . . 64
G.5 SWE Execution Environments . . . 65

H Inference Implementation and Benchmarks **65**
H.1 Kernel Implementations . . . 65
H.2 Recurrent MTP Background . . . 66
H.3 DFlash2 Training . . . 66
H.4 Component Benchmarks . . . 67

References **69**

1 Introduction and Model Overview

1.1 Scope and Main Contributions

GigaChat 4 is a 708B hybrid mixture-of-experts language model that succeeds the 432B GigaChat 3.5 generation. It supports a native one-million-token context and provides Reasoning and Instant modes in a single model.

Model and training. We scale the hybrid backbone while introducing stability controls for deep sparse training and a native one-million-token extension based on DeepSeek Sparse Attention. Sections 2 and 4 describe the resulting recipes. Their supporting experiments are collected in Appendix D.

Data. We extend the data pipeline with document parsing and filtering for web text, mathematics, multilingual STEM content, and source code. Midtraining adds synthetic tasks for reasoning and tool use. Sections 2.1 and 3.1 describe these pipelines. Appendices A, B, and C provide data-construction details.

Post-training. After supervised fine-tuning, we train specialist models with reinforcement learning and transfer their capabilities to a single DSA model with Reasoning and Instant modes. Section 5 describes the post-training pipeline, with reward design in Section 5.2.3 and multi-teacher on-policy distillation (MOPD) in Section 5.3.

Infrastructure. We use FP8 computation, sharded parallelism, and overlap between computation and communication for large-scale training. Section 2.5 describes these components. Sections 5.4.1 and 5.4.3 cover asynchronous post-training and context parallelism for Gated DeltaNet, respectively. Appendix G.3.2 provides supporting block-level benchmarks for context parallelism.

Inference optimization. We combine low-precision expert execution, fused kernels, and speculative decoding in the serving stack. Section 6 describes the implementations and performance measurements. We compare recurrent MTP with DFlash2 in Section 6.3 and report prefill and decode throughput across context lengths in Figure 5.

Evaluation. We evaluate both Reasoning and Instant modes on coding, STEM, instruction following, long-context understanding, and tool-use benchmarks. Our evaluation framework combines public benchmark runners and scoring code with our own implementations. Under our evaluation setup, GigaChat 4 Reasoning performs comparably to DeepSeek V4 Flash 0731 and Mistral 4 Large on coding and agentic benchmarks. Section 7 describes the evaluation protocols, and Tables 7 and 8 provide detailed comparisons.

1.2 Architecture and Model Configuration

Compared with GigaChat 3.5, GigaChat 4 increases depth from 40 to 64 blocks while retaining the model width and 3:1 GDN–MLA schedule (Table 1). For long-context processing, DSA replaces dense attention in the MLA blocks, while the GDN blocks remain unchanged.

The stability controls, including block-local DLR and projection-specific $\mathcal{Z}$-loss, are described in Section 2 and Appendix D.1. Figure 1 shows the decoder architecture.

Table 1. Model specifications and the transition from GigaChat 3.5 to GigaChat 4. GigaChat 3.5 specifications refer to the released instruction-tuned checkpoint [1].

Property	**GigaChat 3.5**	**GigaChat 4**
Nominal total / Active size	432B/$\sim$28B	708B/$\sim$41B
Decoder layers / hidden width	40/7,168	64/7,168
GDN / MLA layers	30/10	48/16
MoE experts (routed / active / shared)	256 / 8 / 1	256 / 8 / 1
Initial layers with dense FFNs	3	3
Vocabulary size	128,256	128,256
Residual normalization	Zero-Centered GatedNorm	GatedNorm
Normalization gate	2σ	2σ
Progressive residual warm-up	Not used	Used
SwiGLU clipping threshold	10	10
Logit soft-cap	Not used	30
Long-context MLA computation	Dense MLA	DSA, top-K 2,048
Maximum context length	262,144	1,048,576
Pre-training data target	$\sim$11T	$\sim$20T

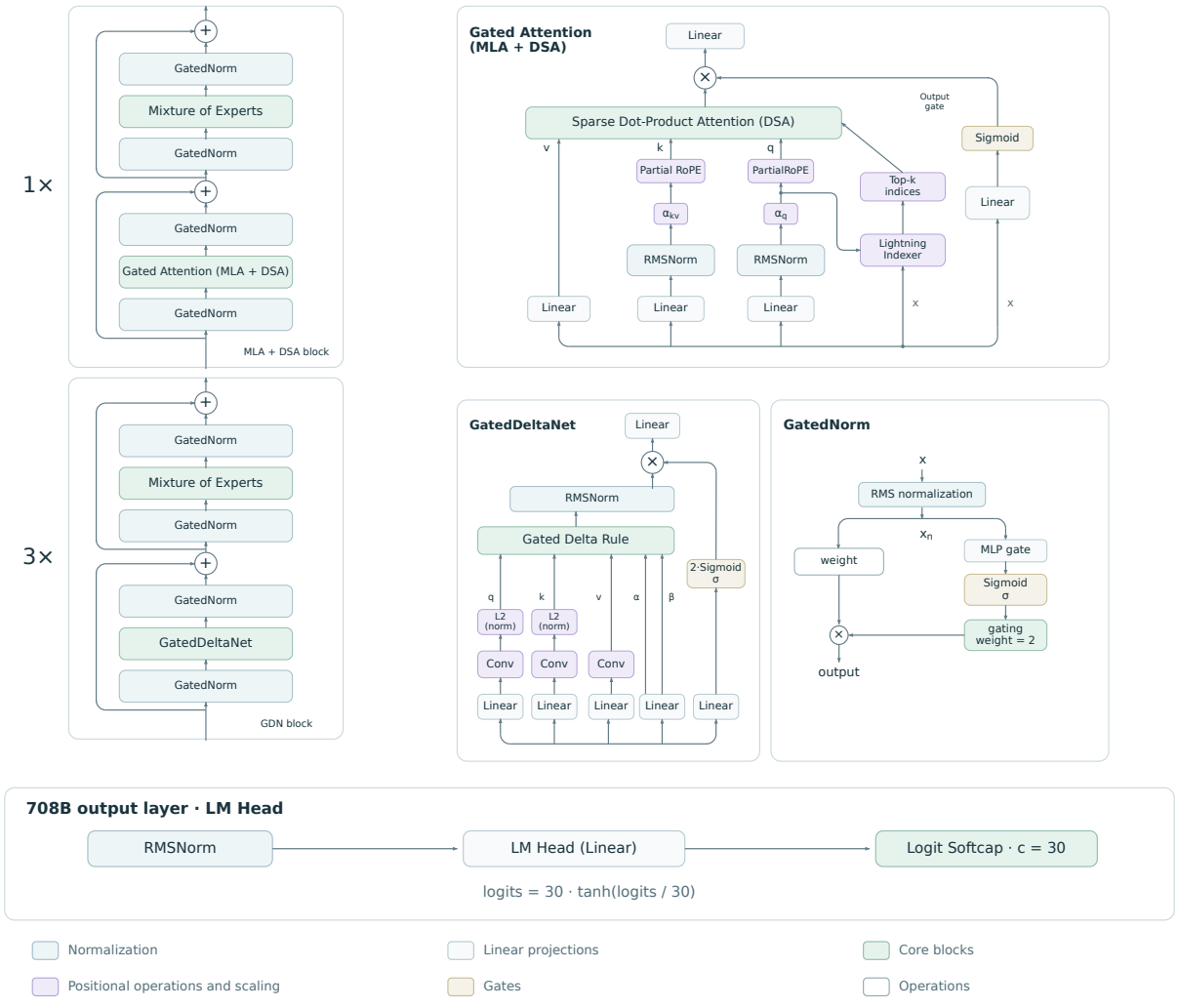

Figure 1. GigaChat 4 architecture.

2 Pretraining

2.1 Data

The GigaChat 4 pretraining corpus contains approximately 15.35 trillion tokens, with a pretraining target of 20T tokens (Table 1). We set the mixture weights through source-specific repetition factors: higher-quality and more complex datasets receive more passes through their available data. Each source contributes in proportion to its size multiplied by its repetition factor. The resulting domain sampling weights are 57% web, 17% code, 13% OCR/PDF documents, 6% mathematics, 2.95% multilingual web, and 4.05% reference and domain-specific sources. Pretraining relies primarily on naturally occurring text and source code, with a small share of synthetic examples in the web, code, and mathematics domains. We use synthetic data more extensively during midtraining (Section 3). Appendix A.1 gives the corpus sizes and sampling weights by domain.

The data pipeline builds on the earlier GigaChat family [2] and GigaChat 3.5 [3]. We collect web documents from Common Crawl [4] and targeted crawls, using extraction and filtering stages comparable to those in published web-data curation pipelines [5–10]. English and Russian form the core of the web corpus. Additional streams cover Arabic, Chinese, French, German, Japanese, Portuguese, and Spanish, together with languages of the post-Soviet states and minority languages of Russia. Documents are routed by language before language- and domain-specific selection. We complement web data with text extracted from PDF documents and additional general-purpose and domain-specific collections.

We convert HTML to Markdown with an LLM-based parser [11, 12], preserving the main text, equations, tables, and code. A rule-based extractor provides a fallback when the primary parser cannot process a document. We evaluate extraction quality against manually annotated Russian and English pages and an external parsing benchmark. PDF documents undergo OCR to recover text and mathematical content. The HTML parser and its evaluation are described in Appendices A.3.2 and A.3.3.

Mathematics- and programming-related web documents are selected in two stages. A lightweight fastText [13] classifier first retrieves candidates at high recall, reducing the volume passed to a more selective LLM-based classifier. For code-related pages, the candidate selector is developed iteratively from known programming sites: promising hosts and URL prefixes are inspected and added to the seed collection before retraining. This stream includes documentation, tutorials, forum discussions, and technical blog posts. To improve Russian-language mathematics coverage, we also include Russian translations of selected English mathematical documents.

Relative to GigaChat 3.5, we expand mathematics collection through targeted crawls, extraction of text from images, a dedicated multilingual mathematics classifier, and separate classifiers for parsed PDF documents (Appendix A.3.7). We separately collect broader STEM content from multilingual Common Crawl: a high-recall fastText filter retrieves candidates, and an MLP over document embeddings assigns the final labels (Appendix A.4). Programming-related web pages receive an additional quality filter implemented as an MLP over document embeddings (Appendix A.3.6).

Source-code collection combines public repositories with additional mathematical code resources. We reconstruct repository file sets, remove near-duplicates, and select files by extension and programming language, following the repository-processing approach of StarCoder2 [14]. Rule-based filters derived from OpenCoder [15] check file length, character diversity, and language-specific properties. Classifiers trained on LLM annotations further assess educational value, complexity, correctness, and robustness. We downsample the most

common languages to preserve coverage of less frequent ones. The resulting corpus spans more than 400 programming languages. We describe collection and filtering in detail in Appendix A.7.

Quality classifiers trained on LLM annotations assess educational value and the presence of useful explanations or reasoning. We remove exact and near-duplicates using normalized-text matching and MinHash-based similarity detection [16]. Benchmark decontamination retrieves candidate overlaps with evaluation items and checks whether a document exposes the task or its solution. The source-specific filtering and decontamination procedures are described in Appendices A.3.5 and A.8.

2.1.1 Pretraining Data Recipe Development

We evaluate data choices through proxy training before using them at production scale. The proxy suite developed for GigaChat 3.5 spans dense 3B models trained on 200B, 400B, and 1T tokens, dense 7B models trained on 400B tokens, and 120B MoE models trained on 2T tokens for final mixture comparisons. The GigaChat 3.5 mathematics experiments also used staged training: a comparison started from a 7B model trained on 200B web tokens and continued for another 200B tokens on a mixture of 70% web and 30% mathematics. For GigaChat 4, we extend this suite with 10B MoE models trained on approximately 200B tokens and 37B MoE models trained on approximately 370B tokens. We give the proxy training recipes and hyperparameters in Appendix A.9.

Pretraining data experiments use classification accuracy for multiple-choice tasks, choosing the answer with the highest model probability, and mean@8 over eight sampled responses for generative tasks. We compare recipes using a paired bootstrap over benchmark items. Benchmark suite and evaluation details are given in Appendix A.9.1.

2.2 Training Recipe

We train the model with Decoupled AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-20}$, wd $= 1.4 \times 10^{-5}$) and a WSD scheduler (the peak learning rate is 1.4×10^{-4}, decay is cosine). Gradients are clipped to a global norm of 1.0. The sequence length increases from 768 to 4,096 tokens during the first 13,000 steps, while ProRes progressively exposes the residual branches over a 15,000-step schedule starting from a residual coefficient of 0.1. The global batch contains 18,432 sequences, corresponding to approximately 75.5 million tokens per update at a sequence length of 4,096.

The routed feed-forward layers contain 256 experts and use top-8 sigmoid routing with score scale 2.5. The model is initialized with standard deviation 0.006. The optimizer, initialization, and stability controls were selected jointly on a hierarchy of proxy models. We report the scale-transfer methodology and supporting results in Appendix D.1.

2.3 Architectural Stability Controls

2.3.1 GatedNorm and Progressive Residual Warm-up

GatedNorm explicitly controls feature-wise rescaling through a bounded gate [17]. We use

$$y = \frac{\gamma \odot x}{\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2 + \epsilon_{\text{norm}}}}, \tag{1}$$

$$g(y) = W_2\,\text{SiLU}(W_1 y), \tag{2}$$

$$\text{GN}(x) = 2y \odot \sigma(g(y))\,. \tag{3}$$

We use the factor of two in Eq. (3), introduced in GigaChat 3.5 [3]. It makes the gate approximately scale-preserving for logits near zero, since $2\sigma(0) = 1$. Proxy stability ablations favored a directly learned scale $\gamma = w$ over the zero-centered parameterization $\gamma = 1 + w$, with the bounded gate unchanged (Appendix D.1.2).

ProRes [18] complements GatedNorm by gradually increasing residual-branch contributions. Writing F_l for a normalized sublayer, the update is

$$x_{l+1} = x_l + \alpha_l(t) F_l(x_l). \tag{4}$$

For decoder layer $l \in \{1, \ldots, N\}$ and training step $t \geq 0$, our linear schedule is

$$\alpha_l(t) = \alpha_0 + (1 - \alpha_0)\min\left(\frac{t}{T_l}, 1\right), \qquad T_l = T\frac{l}{N}, \tag{5}$$

with $\alpha_0 = 0.1$ and $T = 15{,}000$ in the production recipe. The coefficient scales the normalized attention and feed-forward outputs before residual addition, while the identity path is unchanged. Shallower layers reach unit scale earlier. We also zero-initialize the post-attention GatedNorm scale, adapting the zero-gain output RMSNorm initialization in MAI-Thinking-1 [19]. We report diagnostic comparison on the auxiliary 27B-parameter model in Appendix D.1.2.

2.3.2 *SwiGLU Clipping*

SwiGLU clipping bounds the two multiplicative factors before they are combined. With clipping threshold $L = 10$, the transformation is

$$\text{SwiGLU}_L(x) = \text{SiLU}\big(\min(W_g x, L)\big) \odot \text{clip}(W_u x, -L, L). \tag{6}$$

The threshold leaves the central activation distribution almost unchanged while preventing rare coordinate pairs from producing extreme expert outputs.

2.3.3 *Logit Soft-Capping*

To stabilize optimization of the output projection (`lm_head`) and control its gradient norms, we apply a smooth soft-cap to the logits before the token softmax,

$$\widetilde{z} = c\tanh(z/c), \qquad c = 30, \tag{7}$$

which bounds logit magnitudes and attenuates gradients associated with large logits without introducing a hard discontinuity. We discuss the retained SwiGLU clipping threshold and the soft-capping experiments in Appendix D.1.3.

2.3.4 DLR and Projection-Specific $\mathcal{Z}$-Loss

Dynamic learning-rate rescaling (DLR) responds to sustained growth in block-level maximal violation (MaxVio), the monitored expert-load imbalance statistic. For each block l, it estimates a trend over a recent window,

$$m_l(t) = \text{slope}\big(\{\text{MaxVio}_l(u)\}_{u=t-W+1}^{t}\big)\,, \tag{8}$$

and reduces the local learning-rate multiplier when this trend exceeds a threshold. This controller has warm-up, cooldown, and a multiplier floor limit. It combines our observation that rising MaxVio can precede loss degradation with evidence that elevated learning rates reproduce large-scale instabilities in small proxies [20]. Local rescaling reduces optimization pressure in the affected blocks while preserving the global learning-rate schedule.

We add a $\mathcal{Z}$-loss penalty to the activations of selected projections:

$$\mathcal{L} = \mathcal{L}_{\text{base}} + \sum_{p\in\mathcal{P}} \lambda_p\, \mathcal{Z}\big(P_p(h)\big)\,, \tag{9}$$

$$\mathcal{Z}(a) = \left[\log\left(\sum_{j=1}^{d}\exp(|a_j|)\right)\right]^2,\ a\in\mathbb{R}^d. \tag{10}$$

Here $\mathcal{L}_{\text{base}}$ includes language modeling, MTP, routing auxiliaries, and the existing block-output penalties. The additional terms regularize activations at the projections $\mathcal{P}$ identified by activation and spectral monitoring, with coefficients assigned separately by layer and projection. Production interventions included query, key, value and MLP up projections, including the MTP branch. We report the effects of these interventions in Appendix D.1.4.

2.4 Monitoring and Interventions

We monitored cross-entropy and MTP losses, gradient clipping, activation maxima, expert utilization, and MaxVio. At checkpoints, we also tracked the largest singular value and stable rank of query, key, value and router matrices. A simultaneous increase in the largest singular value and decrease in stable rank indicated that a projection was concentrating capacity into a small number of directions.

When routing imbalance grew in an isolated set of blocks, DLR reduced only their optimizer multipliers. When the anomaly was localized to an internal projection, the corresponding $\mathcal{Z}$-loss coefficient was adjusted instead. This separation allowed the training run to correct local instability without reducing the learning rate of the full model.

2.5 Infrastructure

Our pretraining setup combines expert parallelism (EP) with GigaFSDP in a hybrid sharded data parallel (HSDP) configuration. EP distributes routed experts across ranks, while GigaFSDP flattens parameters into contiguous buffers and shards them across the corresponding groups. Shard boundaries can therefore split weight matrices and their FP8 quantization blocks.

FP8 computation. We use blockwise FP8 quantization following DeepSeek-V3 [21], with 128×128 blocks for weights and 1×128 blocks for activations. Grouped expert matrix multiplications use an adapted DeepGEMM [22] implementation, while dense projections use NVIDIA cuBLAS [23]. Selected communication paths also use FP8, while numerically sensitive operations retain BF16 or FP32 precision.

FP8 parameter all-gather. At the start of each optimizer step, ranks compute FP32 absolute maxima over 128-element segments of their BF16 routed-expert weight shards and exchange the packed statistics across the sharding group. A fused Triton kernel reconstructs common scales for global 128×128 blocks and quantizes each local shard to E4M3, including blocks that cross shard boundaries. FP8 shards and scales are reused across gradient-accumulation microbatches. Forward and backward gathers supply them directly to the expert kernels, avoiding a BF16 gather followed by quantization. This halves the weight payload relative to BF16, while the statistics exchange adds four bytes per 128 weights once per optimizer step. Optimizer master weights remain in FP32, and gradient communication retains its configured precision.

Activation checkpointing and offload. We checkpoint attention, MoE, and post-MoE normalization separately. Keeping post-MoE normalization outside the expert checkpoint region avoids replaying the forward output-combine operation during recomputation. Selected region inputs are offloaded to CPU during forward execution and restored for backward.

In a profiled 40-layer configuration, inputs to each region type occupied approximately 6.4 GB per GPU across all layers. FP8 storage reduced the MoE-input footprint from 6.4 GB to 3.2 GB. In one profiled run, CPU offload removed approximately 16 GB of accumulated activation memory per GPU.

Measurements of FP8 checkpoint storage showed an approximately 20% reduction in peak GPU memory. This storage mode is not combined with the two-chunk execution schedule described below.

Profiling showed that GPU-to-CPU copies prolonged concurrent DeepEP operations and, to a lesser extent, NCCL all-gather, even on a dedicated CUDA stream. We therefore defer these copies to execution windows between communication operations.

Overlapping expert computation and communication. We split routed tokens into two nearly equal chunks to overlap expert computation with communication, using a schedule similar to the two-chunk forward schedule described in LongCat-Flash [24]. Routing and the shared expert remain outside this schedule. Expert computation for the first chunk overlaps dispatch of the second, and the first chunk's combine overlaps computation of the second. Asynchronous DeepEP operations[1] and CUDA events enforce dependencies. Both chunks reuse the same quantized expert weights, including weights supplied by FP8 all-gather.

We recompute expert activations within the backward schedule, avoiding replay of the complete forward communication sequence. Saved inputs are redispatched, and the first chunk's input and routing-weight gradients are combined while the second chunk is recomputed and differentiated. Optionally, retaining the first chunk's received activations and routing metadata avoids its redispatch at the cost of additional memory.

3 Midtraining

3.1 Data

Midtraining combines selected web documents, books, and source code with a larger emphasis on synthetic data than in pretraining. We focus on difficult tasks in mathematics, programming, STEM, medicine, finance, and law, presenting them as questions with solutions, explanations, or sequences of actions. We extract questions from source documents

[1] Fix for fused combine after non-blocking dispatch: https://github.com/deepseek-ai/DeepEP/pull/766.

and generate new problems using Magpie [25] and PromptCoT [26]. Document rewrites provide additional educational material.

We check generated examples using model assessments, program execution, tests, or task-completion checkers, depending on the task. Mathematical documents receive additional formula normalization, image-content extraction, and relevance ranking. The mixture also retains high-quality organic text and code. We describe data preparation and the synthesis pipelines in Appendix B.

3.1.1 Midtraining Data Recipe Development

We compare candidate sources through continuation runs from a common auxiliary 37B-parameter model checkpoint, using a 32,768-token context and approximately 40B training tokens. Separate experiments evaluate mathematics, code, and reference and domain-specific data by changing their allocation within the mixture. We give the baseline and candidate proportions in Appendix B.3. We use the pretraining benchmark suite and evaluation protocol (Appendix A.9.1), supplemented by zero-shot RULER [27], HELMET [28], and SCROLLS [29] evaluations.

3.2 Training Recipe

The final 1T training tokens are used as a midtraining stage. We apply cosine decay from the starting checkpoint's learning rate of 5.5×10^{-5} to 1.0×10^{-5}, without warmup. The context length is extended from 4,096 to 32,768 tokens with the global batch size updated to 2,048 sequences corresponding to approximately 67.1 million tokens. Documents are packed with variable-length attention masks, so tokens do not attend across document boundaries. Other training hyper-parameters remain the same as in the pretraining stage.

4 Context Extension

4.1 Data

Long-context training combines long documents with synthetic questions that require retrieving and connecting information across distant passages. The organic data include OCR-extracted scientific and technical PDFs, long web pages, academic documents, complete code repositories, and concatenated books. Using the OpenAlex [30] citation graph, we also assemble scientific articles with the full text of cited papers.

We filter documents for quality, remove benchmark contamination, and adjust sampling to increase exposure to longer texts. Code repositories are selected as a whole, preserving their files and cross-file dependencies. Synthetic data include questions over PDFs with answers that cite supporting passages, questions over shuffled web passages, and paragraph-based questions placed away from their supporting text. Appendix C describes the source pools, length selection, and construction procedures.

4.1.1 Long-Context Data Recipe Development

Repository comparisons use 37B proxies at 128k context to select quality thresholds, decontamination rules, and length sampling. Additional experiments evaluate long-document sources and synthetic question answering on retrieval and reading benchmarks. These comparisons guide the final data recipe. Setups and results are given in Appendices C.4 and D.2.4.

4.2 Training Recipe

The 32k midtraining checkpoint is first extended to 128k with dense MLA attention. The full trunk is trained for 5,000 updates at a learning rate of 2×10^{-5}, corresponding to approximately 336 billion tokens. This stage adapts positional representations independently of sparse token selection.

DSA is then introduced in two stages. In the first, the trunk is frozen and the indexer is trained at 128k with a learning rate of 5×10^{-4}. The checkpoint at 2,000 updates is used for the next stage. In the second, the trunk and indexer are trained at the native one-million-token length for 6,000 updates with a learning rate of 2×10^{-5}, corresponding to approximately 400 billion tokens. The indexer learns to match the attention distribution through an auxiliary KL objective, while the trunk is optimized for language modeling [31]. The curriculum moves directly from 128k to one million tokens. Our experiments indicate that sparse adaptation requires a longer optimization horizon to recover long-context quality than dense continuation (see Appendices D.2.2, D.2.3, and D.2.5 for details).

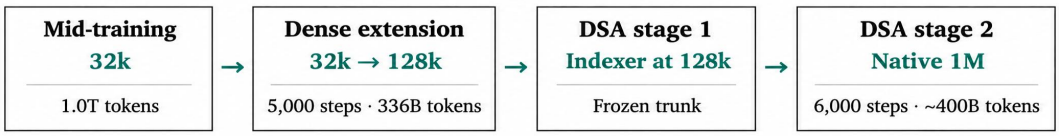

Figure 2. Selected context-extension curriculum. The dense trunk first learns the 128k positional geometry, the DSA indexer is then fitted with the trunk frozen, and both components are finally trained jointly at the native one-million-token length.

4.3 Long-Context Architecture

The long-context model retains the 64-block hybrid backbone described in Section 1. YaRN [32] rescales rotary position embeddings, while DeepSeek Sparse Attention (DSA) [31] replaces dense attention only in the 16 MLA blocks, and the 48 gated linear-attention blocks remain unchanged. The resulting backbone combines content-based sparse retrieval with recurrent linear mixing.

For a query at position t, the DSA lightning indexer assigns a relevance score to each preceding token s by aggregating H_I indexer heads,

$$I_{t,s} = \sum_{j=1}^{H_I} w_{t,j}^I \, \mathrm{ReLU}\Big(\big\langle q_{t,j}^I, k_s^I \big\rangle\Big), \qquad \mathcal{S}_t = \mathrm{TopKIndices}_{s \le t}(I_{t,s}, K), \tag{11}$$

and the MLA operation is evaluated only on the selected key-value entries $\{c_s : s \in \mathcal{S}_t\}$. The production configuration uses $K = 2{,}048$. The indexer therefore learns which tokens should be retained, whereas the pretrained MLA projections determine how information from those tokens is aggregated.

4.4 Selected Configuration

The final model uses YaRN with reference length 32,768 and scaling factor 32, DSA top-K 2,048 in every MLA block, and a native maximum sequence length of 1,048,576 tokens. The indexer has 64 heads of dimension 128. The final YaRN configuration is already enabled during dense adaptation at 128k and remains fixed thereafter, avoiding another change of positional scaling when sequence length increases to 1M. We report supporting experiments in Appendices D.2.1, D.2.3, and D.2.4.

4.5 Monitoring and Checkpoint Selection

Checkpoint selection combines the language-model objective with retrieval and aggregation evaluations at 128k, 512k, and one million tokens. Single-key retrieval often saturates before multi-key retrieval or question answering. Consequently, neither a low training loss nor a strong retrieval endpoint alone establishes convergence. We monitor task-level trajectories, indexer KL, expert-load balance, and short-context quality, interpreting small score changes in light of evaluation sample size. The final checkpoint preserves the dense 128k reference's quality while supporting native 1M inference. Quantitative comparisons and the indexer-removal experiment are reported in Appendices D.2.6 and D.2.7.

4.6 Infrastructure

4.6.1 Fused indexer and top-k selection

The full matrix of index scores for a sequence of 131,072 tokens takes 64 GiB in FP32 for a single DSA layer. Computing it in chunks and merging partial top-k results avoids the allocation, but every score is still written to global memory and read back. We instead fuse scoring and selection into a single kernel, written in TileLang. The kernel accumulates candidates in per-query shared-memory buffers. Before overflow, radix selection retains the best k candidates and sets the pruning threshold to the k-th highest score. Only selected indices and per-query counts are written to global memory, while index scores remain on chip. Causal masking and sequence boundaries are passed as a per-query key range, so a single compiled kernel serves packed batches of variable-length sequences as well as the shards of context parallelism. Figure 3 illustrates the data flow. We release an implementation of this kernel as a TileLang pull request [2].

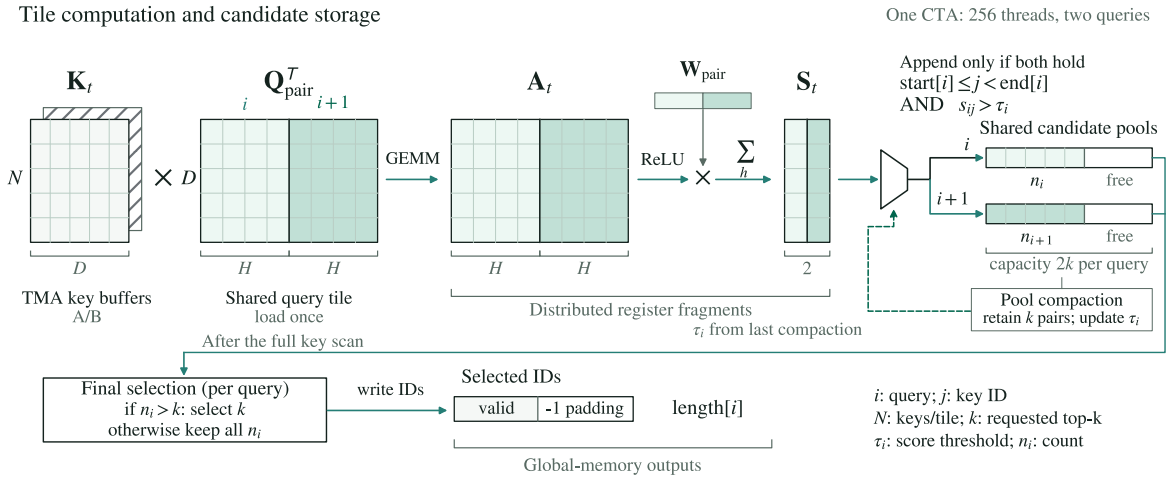

Figure 3. Fused DSA index scoring and top-k selection. Each thread block processes two queries, computes scores tile by tile in registers, and stores candidates in shared-memory pools. Before a pool can overflow, a top-k selection compacts it and updates its threshold. Only selected key indices (with -1 padding) and per-query counts are written to global memory. The full score matrix is never materialized.

[2] https://github.com/tile-ai/tilelang/pull/4382

5 Alignment

5.1 Cold Start

5.1.1 Data

The cold-start mixture covers mathematics, STEM, programming, agent tasks and software engineering, long-context tasks, general chat and instruction following, and finance and law. Training examples include verified solutions, complete agent trajectories, filtered public datasets, human annotations, and de-identified user interactions. The mixture supports Russian and English. Agent tasks, mathematics, and programming together account for more than 77% of its allocation (Table 2).

We select examples using reasoning-quality and human-preference reward models, answer-consistency checks, and programmatic verification. Additional filtering removes benchmark contamination, language and formatting errors, and invalid characters. Appendix E describes the sources, generation pipelines, and validation tools.

Table 2. Domain composition of the GigaChat 4 cold-start training mixture

Domain	Share of training mixture
Agents & SWE	34.3%
Mathematics (MATH)	21.8%
Programming (Code)	21.0%
General tasks, instruction following, and chat (General, IF, Chat)	11.9%
STEM	5.4%
LongContext	4.2%
Finance & Law	1.4%

5.1.2 Cold-Start Data Recipe Development

We compare candidate datasets against domain-specific baselines using auxiliary 37B-parameter models and fixed benchmark suites. These experiments guide source selection and mixture weights. We then combine the selected domain datasets into the final cold-start mixture. Appendix E.5 gives the experimental setup, source-weight comparisons, and baseline results.

5.1.3 Supervised Fine-Tuning

Supervised fine-tuning needs about 10K optimizer steps to converge. With a 512K-token context and our sequence packing, the global batch size is counted in packed sequences, and the learning rate is tuned to the global batch so that training converges within 10K steps. Learning rates above 2×10^{-4} diverge regardless of the other settings, while useful values lie between 5×10^{-5} and 2×10^{-4}, depending on the training length. Attempts to shorten training to about 3K steps with a higher learning rate or a larger global batch were unsuccessful.

Training was frequently unstable: max-vio, the expert-load imbalance statistic that we also monitor during pretraining (Section 2.3.4), grew rapidly. In a sweep over the global batch size and the learning rate, the largest stable settings were a global batch of 64 and a learning rate of 2×10^{-4}, and larger global batches gave substantially worse metrics. Disabling MoE load balancing did not help. The final model was trained with a learning rate of 5×10^{-5} and a global batch of 32 packed sequences for 10K steps.

The routing balance inherited from the cold-start checkpoint also matters for subsequent RL. We examine this effect in Section 5.2.4.

5.2 Online Reinforcement Learning

5.2.1 Data

STEM. The STEM data consist of mathematical problems with reference answers, olympiad-style proofs, and STEM questions. Our mathematics data comprise 123,250 problems with verifiable final answers, including problems collected from online mathematics forums, and 27,740 proof problems for the early phase of training. Our STEM data comprise 39,735 questions with verifiable final answers. Each prompt is either plain or carries the answer-format instruction in the system prompt or as a prefix of the user message.

Competitive programming. The competitive-programming data consist of competitive-programming problems with hidden test cases, posed in twelve programming languages: Python, C++, C#, Java, Go, Rust, Ruby, PHP, JavaScript, TypeScript, Kotlin, and Scala.

Agent harnesses. Preparation included adapting execution environments, making dependencies available without external network access, and checking evaluation correctness. We verified the expected test outcomes before and after applying reference solutions, checked for regressions, and assessed consistency across repeated runs. We removed Git objects that could expose reference solutions from repositories accessible to the agent.

We also evaluated tasks through multiple attempts by our model, excluded overly easy tasks, and used only valid evaluation outcomes. We retried a subset of unsolved tasks with other strong models, compared them with public solutions, and used additional model-based assessments to check the correctness of task descriptions, tests, and resulting patches. Based on these runs and reviews, we updated a subset of tests and task descriptions and excluded a substantial portion of the reviewed tasks because of contradictory requirements or unreliable evaluation. During training, we also dynamically excluded tasks with a high success rate from further sampling.

General-chat. The general-chat training dataset contains 177,000 tasks spanning arena-style open-ended chat, safety, instruction following, structured output, long-context retrieval and question answering, planning, B2C assistant tasks, and web search. For all domains except arena-style chat and B2C assistant tasks, responses are evaluated using reference answers, either programmatically or by an LLM judge.

During data preparation, we filtered examples and adjusted the proportions of task domains, as well as the mix of topics and task types within each domain. We used the initial model to identify and filter out easy examples in every domain except arena-style chat.

Tool use. We describe the environments and tasks for tool use in Appendices F.1.1–F.1.4.

5.2.2 Training Recipe

We remove the indexers from the post-SFT cold-start checkpoint and restore full MLA in the corresponding layers to initialize the RL experts. A preliminary comparison of the original and converted checkpoints showed no noticeable degradation on the evaluated suite. MOPD transfers their capabilities to a student initialized from the original DSA-enabled checkpoint (Section 5.3).

DSA selections can differ between rollout and training and affect RL stability [33]. For our 708B model, replaying 32-bit indices for 16 DSA layers with top-$k = 2048$ over 131,072 tokens

requires 16 GiB without compression, versus 64 MiB for MoE R3 (64 layer slots, eight experts per token, one-byte indices): a 256-fold difference. We describe DSA training optimizations in Section 5.4.2.

5.2.3 *Reward Design*

Persistent-memory rewards are described in Appendix F.1.4.

The STEM, competitive-programming, tool-use, and general-chat experts are trained in reasoning mode from the cold-start checkpoint. Each uses 32 rollouts per prompt, GRPO advantages [34], and CISPO [35] with M2PO token masking [36], without a KL penalty. Adam uses $(\beta_1, \beta_2) = (0.9, 0.95)$. The MoE router is frozen, online load balancing is disabled, and rollout expert selections are replayed with R3 [37]. We give per-expert settings in Table 3 and complete protocols in Appendix F.2.

Table 3. RL settings by expert. CP denotes competitive programming. All experts generate training rollouts at temperature 1.0. STEM, CP, and general chat use Adam $\epsilon = 10^{-15}$. General chat also uses weight decay 0.01. Tool-use and general-chat runs clip the gradient norm at 1.0 and use a constant learning rate after warm-up.

Setting	STEM	CP	Agent harnesses	SWE	Tool use	Chat
Prompts per update	128	128	128	128	125	128
Prompt token limit	8,192	4,096	98,304	12,288	32,768	131,074
Response token limit	65,568	32,768	131,072	118,784	65,536	32,768
Learning rate (10^{-6})	1	2	2	3	2	2
Warm-up steps	25	35	10	25	25	25
Maximum assistant turns	1	1	160	160	80	5

Tool use and general chat use token-mean loss aggregation. External harnesses use prompt-mean aggregation.

STEM expert. Final answers receive correctness rewards of 1 or 0 from symbolic equivalence or an LLM judge. Five-verdict majority voting reduces judge noise by 53% compared with a single verdict. Proofs are used only during the first 80 steps and graded by a dedicated reference-free LLM judge. Appendix F.2.1 specifies grading and format adjustments.

From the resulting task reward R_{task}, we subtract a pass-rate-scaled length penalty [38],

$$R = R_{\text{task}} - w_{\text{len}}\,\rho_q\,\frac{|y|}{l_{\max}}, \tag{12}$$

where R_{task} is the graded task reward, w_{len} the penalty weight, ρ_q the successful-rollout fraction for prompt q, $|y|$ the response length, and $l_{\max}$ its limit. Appendix F.2.11 specifies domain-specific weights, success criteria, and token counting.

Competitive-programming expert. Test execution alone determines correctness: 1 if all tests pass and 0 for failures. We apply Equation (12) using the test-pass rate. Appendix F.2.3 specifies extraction and compliance adjustments.

Agent harnesses. For online reinforcement learning of our coding agent, we prepared verifiable software-engineering tasks in repository environments. Preparation included adapting execution environments, making dependencies available without external network access, and checking evaluation correctness.

For external agent harnesses (Section 5.4.5), the base score is

$$r_{\text{base}} = \begin{cases} 1, & \text{the solution passes the check and the agent ends normally,} \\ 0.5, & \text{the solution passes the check, but a context or turn limit stops the rollout,} \\ 0, & \text{the solution fails the check or a special stop is triggered;} \end{cases} \tag{13}$$

where only a confirmed limit yields $r_{\text{base}} = 0.5$, and a successful submission on the last allowed turn keeps $r_{\text{base}} = 1$. The final reward is

$$R = r_{\text{base}} \left(1 - 0.2\, p_{\text{success}}\, C_{\text{traj}} - 0.2\, C_{\text{content}}\right), \tag{14}$$

where p_{success} is the fraction of rollouts of the same task with a positive base score, and both costs are normalized to $[0, 1]$. Here C_{traj} measures total session length, including tool responses and summarizations, and C_{content} measures average visible assistant text per turn, excluding reasoning, tool-call arguments, and responses to summarization requests.

We average the loss over trainable generated assistant tokens across each task's rollouts, then over tasks (prompt-mean aggregation). Appendix F.2.5 gives cost thresholds, termination, and loss-masking rules.

Tool-use expert. We train the tool-use expert on the task mixture described in Table 4. Environment rewards combine correctness and format scores. Other tasks use LLM judges (Appendix F.2.6). For web search, the judge evaluates the whole trajectory and returns a binary verdict $b_i \in \{0, 1\}$ for each criterion. Here D contains mandatory criteria and W the quality criteria with weights w_i. The reward is

$$R_{\text{web}} = \underbrace{\prod_{j \in D} b_j}_{\text{dealbreakers}} \times \underbrace{\frac{\sum_{i \in W} w_i b_i}{\sum_{i \in W} w_i}}_{\text{weighted quality}} \times \underbrace{\frac{1}{1 + \max(0,\, h - H)}}_{\text{excess-turn penalty}}, \tag{15}$$

where h is the number of assistant turns with at least one successful tool call (parallel calls within a turn count once, and turns with only failed calls are not counted), and H is the number of such turns after which the penalty starts. The base score satisfies $R_{\text{web}} \in [0, 1]$.

After zero-flooring and completion checks, we apply the length penalty in Equation (12) to both successful and failed rollouts. Here $|y|$ is the whole multi-turn response length, including tool outputs and simulated-user turns. Success means a positive environment reward or a judge score of 1.

General-chat expert. For open-ended chat, the n non-empty, byte-distinct answers play $M = 2$ independent single-elimination brackets with random seeding, and receive

$$R_{\text{arena}}(i) = \frac{w_i}{M\lceil \log_2 n \rceil - y_i} \in [0, 1], \tag{16}$$

where w_i counts the matches won by answer i across both brackets and y_i counts its byes. For $n = 32$, the two brackets require 62 judge calls, compared with 496 for a round robin. An answer that wins both brackets receives reward 1.

Safety grading requires refusal of dangerous requests and assistance on benign requests. We list the task domains and reward sources in Table 18 and give the complete grading protocols in Appendix F.2.8.

In Equation (12), chat uses $\rho_q = 0.5$. Other domains use the group pass rate. B2C and web search instead retain their turn-based penalties.

Table 4. Task mixture and reward sources of the tool-use expert.

Task group	Share	Reward
Customer-service environments with a simulated user	20%	Environment
Harder customer-service and synthetic service environments	5%	Environment
Banking and retrieval-based search environments	5%	Environment
Instruction-following tool-use environments	5%	Environment
MCP environments with synthetic tasks	28%	Environment
Persistent-memory writing	7.5%	Environment
Memory question answering	7.5%	LLM judge (golden answer)
Tool-use tasks adapted for RL[1]	7%	LLM judge (trajectory quality)
Agentic web search	7.5%	LLM judge (rubric)
Simple web search	7.5%	LLM judge (rubric)

[1]Adapted from the Toucan-1.5M dataset.

Instant experts. In instant mode, let m be the fraction of scored assistant turns in a rollout that fail the format check. We apply

$$P_{\text{fmt}} = \begin{cases} 1.2, & \text{if the parser detects a reasoning leak in any assistant turn,} \\ 1.2\,m, & \text{otherwise.} \end{cases} \tag{17}$$

An LLM judge flags reasoning in mathematics and selected tool-use sources. Let k count flagged responses in the rollout and R_{pre} be its tool-use reward immediately before the judge penalty:

$$P_{\text{judge}} = \begin{cases} 0.5\,\mathbb{1}[k > 0]\,\mathbb{1}[\text{task solved}], & \text{mathematics (single turn),} \\ \min(0.1\,k,\, 0.5)\,\mathbb{1}[R_{\text{pre}} > 0], & \text{tool use (multi-turn).} \end{cases} \tag{18}$$

For mathematics, P_{judge} precedes P_{fmt}. For tool use, R_{pre} includes the format adjustment. If a tool-use rollout reaches the generation limit and is marked incomplete, we reset any positive reward to zero before applying P_{judge}.

5.2.4 MoE Load Balancing

With the 37B model, router freezing preserved cold-start imbalance. Online auxiliary-loss-free bias updates at 10^{-3} reduced mean per-layer imbalance from 7.07 to 2.50, but lowered mean training reward from 0.439 to 0.374 over steps 182–200. This concerns the tested bias-update rule. A 708B math-expert run showed rising localized imbalance while validation improved. We report both studies in Appendix F.4.

5.2.5 Training Stability

Diagnostic model. We separate actor–rollout divergence into engine mismatch and accumulated policy drift. For token a at history h, let $p_r^{(v)}$ be the rollout policy and $p_a^{(k)}$ the current actor, with $\delta = \log p_r^{(v)}(a \mid h) - \log p_a^{(k)}(a \mid h)$. For nearby normalized policies with common support, linearizing policy changes and neglecting weight decay gives

$$D_{\text{KL}}(p_r^{(v)} \| p_a^{(k)}) \simeq \tfrac{1}{2}\mathbb{E}_r\Bigg[\Bigg(\underbrace{\varepsilon_{\text{eng}}}_{\text{engine mismatch}} + \underbrace{\sum_{j=1}^{N} \eta_j g^\top u_j}_{\text{accumulated policy drift}}\Bigg)^2\Bigg], \tag{19}$$

where $\varepsilon_{\text{eng}} = \log p_r^{(v)} - \log p_a^{(v)}$ measures same-weight engine mismatch, $g = \nabla_\theta \log p_\theta(a \mid h)|_{\theta_v}$ is the token log-probability gradient at the rollout weights θ_v, and $\mathbb{E}_r$ averages over rollout tokens at fixed h. Across the N intervening updates, η_j is the learning rate and $u_j = \widehat{m}_j/(\sqrt{\widehat{v}_j} + \epsilon)$ is Adam's preconditioned direction. Here $\widehat{m}_j$ and $\widehat{v}_j$ are the bias-corrected first and second moments, ϵ is the numerical stabilizer, and division is elementwise. Appendix F.3 gives the local KL expansion and its scaling with update count.

Monitoring and stabilization. M2PO masking, adjusted Adam adaptation and learning rates, and corrected checkpoint resumption extended affected runs from approximately 150 to 600 steps while the mean training score improved. Appendix F.3.2 gives the observations, settings, and interventions.

5.3 Multi-Teacher On-Policy Distillation

We use MOPD [39, 40] to distill eight 708B domain experts, trained independently from the shared cold-start SFT checkpoint (Section 5.1), into a 708B student initialized from the same checkpoint. STEM, competitive programming, and general chat each supply a Reasoning teacher and an Instant teacher. Tool use and SWE each supply one Reasoning teacher. Each prompt's teacher label determines its scoring teacher and the student's generation mode. We jointly distill both modes across 26 mostly multi-turn and agentic domains, including SWE in multiple scaffolds, using FP8 teachers and an online token-budget scheduler.

5.3.1 Training Recipe

We train the student in FP8 for 120 policy-version steps (240 optimizer updates) with Muon, using 256 prompts with four responses each per optimizer update. We use a constant learning rate of 3×10^{-6} after a 20-update linear warmup. Rollouts use temperature 1.0 and are generated asynchronously with partial rollouts. Policy weights are synchronized every two optimizer updates. Full optimizer, parallelism, and rollout-pool settings are given in Appendix F.5.1.

5.3.2 Objective

We sample student trajectories for on-policy distillation [41] and use the top-k reverse-KL objective of Ma et al. [39], $\mathcal{L}_{\text{KD}}^{\text{TopK}}$, over the teacher's $k = 64$ most likely tokens at student-generated positions. We mask tool observations and clamp log-probabilities from below at -10 and the per-token loss from above at 10. Our fused vocabulary-parallel kernel computes the loss without gathering full student logits. We back-propagate this directly as a supervised loss [41], without task-correctness rewards. The only scalar signal is a turn-local format penalty: tokens in malformed turns receive $\hat{A}_{\text{fmt}} = -2$ and all others receive zero. We optimize it with CISPO [35]:

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{CISPO}}(\theta; \hat{A}_{\text{fmt}}) + \lambda\, \mathcal{L}_{\text{KD}}^{\text{TopK}}(\theta), \qquad \lambda = 0.3. \tag{20}$$

The penalty is neither normalized within response groups nor propagated to other turns. We give the format checks and length-limit exemption in Appendix F.5.4.

5.3.3 Dynamic Domain Scheduling

We adapt D^3-MOPD [42] from four prompt-weighted domains to 26 domains with mean responses ranging from under 500 to 25K tokens. The scheduler reuses each domain's token-mean distillation loss and scores its smoothed loss relative to its initial value, multiplied

Table 5. Comparison of the cold-start initialization (CS), the best domain expert, and the MOPD student (step 120) in reasoning and instant modes. All scores come from the same training-time evaluation pipeline. The independent final evaluation (Section 7) uses partly different settings, so scores reported there may differ. **Bold**/underlined: MOPD significantly better / worse than the expert (95% CI). —: not measured.

		Reasoning		Instant	
Benchmark (metric)	**CS**	**Expert**	**MOPD**	**Expert**	**MOPD**
Agentic and tool use					
τ^3-Bench (mean@3)	0.565	0.686	0.692	0.634	0.589[a]
airline	0.727	0.813	0.840	0.793	0.647[a]
retail	0.696	0.795	0.766	0.728	0.693[a]
telecom	0.722	0.936	0.950	0.868	0.868[a]
banking (knowledge)	0.117	0.199	0.210	0.148	0.148[a]
FCL V4 overall (mean@3)	0.712	0.778	0.781	—	—
SWE-bench Verified (pass@1)	0.766	0.805	0.798	—	—
Terminal-Bench 2.1 (mean@2)*	0.624	0.740	0.747	—	—
CoworkBench (mean@2)*	0.413	0.702	0.688	—	—
Code, mathematics, and science					
LiveCodeBench (pass@1)*	0.874	0.953	0.948	0.785	0.786
AIME 2025 (mean@8)	0.800	0.917	**0.950**	0.754	0.688
AIME 2026 (mean@8)	0.829	0.963	0.942	0.829	0.808
HMMT Feb 2026 (pass@32)[b]	0.833	0.967	0.983	0.850	0.867
IMO-AnswerBench	0.686	0.836	0.842	0.595	0.601
GPQA Diamond*	0.813	0.854	0.869	0.737	0.702
HLE (text-only)	0.164	0.241	0.223	0.097	0.082
Instruction following and structured output					
IFBench (loose)	0.593	0.810	0.773	0.550	0.513
IFBench (strict)	0.553	0.787	0.750	0.503	0.457
JSONSchemaBench	0.317 (0.938)[c]	0.994	0.997	0.957	0.948
StructEval (text)	0.839	0.881	0.886	0.882	0.891
Long context and planning					
GraphWalks	0.445	0.622	**0.745**	0.377	**0.480**
Needle in a Haystack (complex)	0.839	1.000	1.000	1.000	0.957
Natural Plan	0.921	0.939	0.944	0.489	0.509

*Training-time and final settings differ most for these benchmarks. Their scores are not directly comparable with Tables 7–10. [a]No instant tool-use teacher in the run. The expert is a separate instant tool-use model. [b]Instant: pass@8. [c]Strict parser. In parentheses, a lenient parser that extracts the JSON from the Markdown block the cold-start model wraps it in.

by its recent nonnegative relative descent velocity. It converts normalized scores to floored softmax token shares, then divides by running mean response length to obtain prompt counts. Both token and minimum prompt shares have a floor of 0.015. The token floor reserves 39% of the budget. We stabilize the small loss signal with a 0.005 denominator floor and a 10-update initial-loss estimate. The mixture is uniform until sufficient velocity history is available or when no domain descends. We report the complete algorithm and hyperparameters we use in Appendix F.5.5.

5.3.4 *Results and Analysis*

We compare the models in Table 5. Scores there may differ from the final evaluation because of settings and run-to-run variance. In Reasoning mode, no benchmark shows a significant deficit against its expert (95% CI), while AIME 2025 and GraphWalks improve, indicating positive transfer. Instant-mode deficits are significant on τ^3-Bench (almost entirely airline), AIME 2025, and needle-in-a-haystack retrieval. GraphWalks improves. The instant tool-use reference was not a teacher in this run. We report convergence in Appendix F.5.6 and compare Muon with AdamW in Appendix F.5.7.

5.4 Infrastructure

5.4.1 *Asynchronous Training*

Our RL infrastructure extends verl [43] with asynchronous execution across separate resource pools for rollout generation, policy optimization, and scoring. Completed, scored sample groups enter a queue from which the trainer assembles batches while rollout workers continue generating trajectories. This overlaps generation with optimization and allows each pool to be sized for its workload. Queue capacity and a configurable staleness budget bound how far sample production can advance ahead of the trainer. With partial rollouts enabled, interrupted generation resumes from the returned prefix, retaining its log probabilities and expert-routing records within the original response-length budget.

We synchronize policy weights[3] through a P2P checkpoint engine adapted from Miles [44], writing trainer shards directly into rollout-engine buffers with Mooncake RDMA.[4] Synchronization takes approximately 8 seconds in our measured 708B deployment. A replica returns to service only after receiving the updated policy.

Training and rollout engines use coordinated FP8 execution with higher precision for sensitive operations. Rollout Routing Replay (R3) [37] reuses the expert selections recorded during generation for policy updates. It does not eliminate other numerical differences or asynchronous policy lag. Our stack supports a TransferQueue [45] delivery path in which the scheduling queue carries row keys and metadata. With worker prefetch enabled, training workers fetch routing records, teacher top-k outputs, and sampling-replay tensors directly. Appendix G.1 describes weight transfer, tensor delivery, prefetching, and replica recovery.

5.4.2 *DSA Training Optimizations*

For the DSA-enabled MOPD student, our Megatron-based stack uses the fused indexer and top-k selection kernel in Section 4.6.1. Under full activation recomputation, we reuse the first pass's selected indices (k per token) rather than repeat $O(L^2)$ selection. We offload the indices to pinned host memory between passes. Both passes attend to the same tokens.

[3] Fix for multi-owner pipeline-parallel tensors during weight export: https://github.com/NVIDIA-NeMo/Megatron-Bridge/pull/3428.

[4] https://github.com/verl-project/verl/pull/7108

Inside sparse layers, we partition queries rather than heads so each query's entries are selected and gathered once per tensor-parallel group. Context parallelism retains global query positions and Megatron's zigzag token ownership, shared with GDN layers. Appendix G.2 gives the kernel choices, indexer-loss handling, and tensor- and context-parallel exchanges.

5.4.3 Context Parallelism for Gated DeltaNet

Context parallelism (CP) distributes long sequences across GPUs to reduce local activation memory. Our State CP implementation preserves Megatron's zigzag partition shared by attention and Gated DeltaNet (GDN). Head-parallel CP redistributes projected token activations through all-to-all [46]. State CP instead keeps token activations on their original GPUs and communicates FP32 recurrent states and short convolution-boundary windows. States propagate between chronologically adjacent fragments. Backward propagates boundary gradients in reverse order. Packed documents maintain independent histories. The main trade-offs are sequential dependencies between fragment owners and padding to align documents to the Flash Linear Attention (FLA) chunk grid.

In an isolated Transformer-block benchmark using a trained auxiliary 37B model at approximately 128K tokens, CP=2, TP=DP=1, blockwise FP8, and matched selective recomputation, State CP improves block throughput by 30.1% and reduces peak allocated memory by 17.1% relative to head-parallel all-to-all CP. We report the complete protocol and CP sweep in Appendix G.3.2 and describe the checkpoint integration in Appendix G.3.1.

5.4.4 Rollout Inference

Rollouts are generated with SGLang in FP8, using workload-specific tensor parallelism (TP), expert parallelism (EP), and data-parallel attention (DPA). Separate parallel groups for attention and expert weights accommodate their different memory requirements.

MTP and recurrent-state replay. MTP speculative decoding uses replay-based state recovery for GDN layers, following the input-caching principle of ReplaySSM [47]: compact per-step updates reconstruct the accepted recurrent state without retaining a full state matrix at every draft position.

Domain-specific rollout topologies. In our expert-RL workloads, code and mathematics are predominantly decode-heavy, with short inputs (typically below 1K tokens) and generations that can extend to tens of thousands of tokens. DPA with wide expert-parallel groups (TP $=$ EP), together with compact recurrent-state replay buffers, supports higher per-node concurrency for these long generations. In the tested general-domain workloads, compute-bound prefill saturates available compute at lower concurrency, so DPA provides less benefit. Multi-turn SWE and tool-use workloads are less sensitive to parallelism topology and benefit primarily from higher prefix-cache hit rates. We pin requests to a specific data-parallel (DP) rank to reuse cached prefixes. We describe the cache policy and the evaluated alternatives that remain disabled in Appendix G.4.

5.4.5 SWE Training Infrastructure

For SWE expert RL, we integrate Polar with our asynchronous pipeline to execute stateful agent sessions and convert their interactions into training trajectories (Figure 4, Appendix G.5). The infrastructure separates reusable task environments, agent harnesses, model serving and trajectory ingestion.

Harness integration and learning-signal integrity. A common rollout interface supports different agent harnesses, with a consistent harness assignment within each GRPO group.

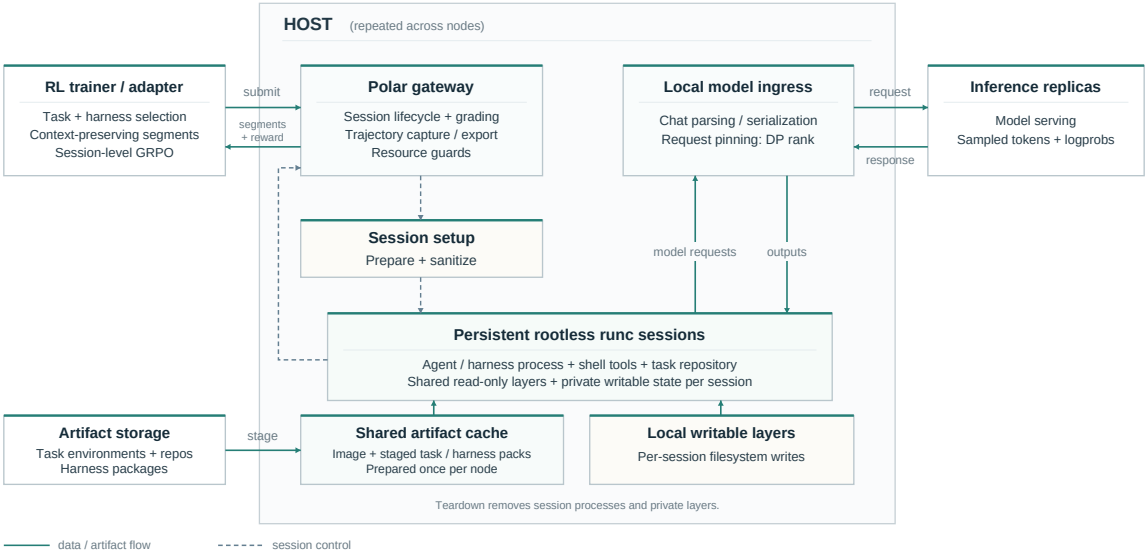

Figure 4. Infrastructure for SWE expert RL. Host-local orchestration connects persistent agent sessions to model serving and returns trajectories and rewards to the trainer. Shared environment artifacts are combined with private session state.

When an episode compacts its context, we retain the prompt token IDs, sampled responses, log probabilities and loss masks associated with the actual model calls. History rewrites form separate training segments, preserving the conditioning used during sampling. Segments share the session outcome, while group-relative reward statistics count each session once, so additional compactions do not multiply its contribution to the baseline. Infrastructure failures are distinguished from graded task failures when constructing the learning signal.

6 Inference

6.1 Serving Stack

Our custom inference stack serves the 708B hybrid backbone, combining sparse MLA attention, recurrent GDN layers, and routed MoE experts. It builds on TensorRT-LLM [48], SGLang [49], and GigaInfer, our in-house inference engine, and supports disaggregated prefill and decode. We contributed native GigaChat 4 support to SGLang.[5] We use NCCL for inter-GPU communication.[6]

Linear layers outside the routed experts use MXFP8 weights and activations with DeepGEMM [22], and the KV cache is stored in FP8.

Each experiment evaluates a separate serving configuration. Comparisons are made within each setup. We describe RL rollout inference in Section 5.4.4.

6.2 Inference Optimizations

We support separate dense/sparse choices for prefill and decode because indexer and top-K overhead can outweigh attention savings at short contexts or small batches. For sparse prefill, we construct FlashMLA inputs directly in their required layout. A K-only indexer prepares the cache for dense-prefill/sparse-decode handoff without prefill scoring or top-K. In eligible

[5] https://docs.sglang.io/cookbook/autoregressive/GigaChat/GigaChat4

[6] Fix for receive-proxy hangs in multi-node inference: https://github.com/NVIDIA/nccl/pull/2390.

SGLang prefill steps, query-token partitioning assigns all heads of each query to one rank, avoiding duplicated indexer work while preserving GDN token order.

For SGLang decode, we fuse query/key/gate projections and FP8 quantization, and fuse NeoX RoPE, Hadamard transformation, gating, and cache writes. During CUDA-graph decode, independent MLA preparation runs on a second CUDA stream, overlapping indexer scoring and top-K on the main stream. The TensorRT-LLM path fuses paged FP8 scoring with exact top-K without materializing logits, alongside fused key/query preparation, with overlap where supported.

Our GatedNorm kernels factor the root-mean-square (RMS) normalization scale out of the first gate projection and fuse subsequent normalization, gating, and residual operations. Specialized prefill and decode kernels can emit MXFP8 output directly for the following projection.

We give implementation details and applicability conditions in Appendix H.1. Approximate indexer routing is a quality-affecting option, and projection-weight replication is not used in the evaluated configurations. Component experiments are in Appendix H.4.

6.3 Speculative Decoding

For GigaChat 4, we replace three sequential, independently parameterized multi-token prediction (MTP) blocks that we used in GigaChat 3.5 (Appendix H.2) with a single recurrent MTP drafter that shares weights and the key-value cache across draft steps, following EAGLE-3 [50]. Unlike EAGLE-3, however, we omit multi-layer feature fusion and condition the drafter only on the final hidden states of the target model. We train it on target-generated responses using SFT data. Our experiments show that this architecture reduces the number of draft parameters by a factor of three while maintaining comparable acceptance length across benchmarks and even improving it on some.

We also explore DFlash2 [51], a parallel drafting approach based on block diffusion [52]. Unlike recurrent MTP, which generates draft tokens sequentially, DFlash2 predicts token candidates for an entire block in a single forward pass. Our DFlash2 drafter consists of five transformer layers with dense multi-head latent attention (MLA), operating over captured target hidden states and masked draft positions. Unlike the recurrent MTP block, which uses causal sparse latent attention, DFlash2 employs a different attention mask and augments its transformer layers with dynamic causal convolutions to mix representations across adjacent positions. We use a block size of 8, consisting of one anchor token and seven draft positions. A lightweight selector then constructs the final proposal sequentially by choosing among the top-16 candidates at each position, conditioned on the previously selected token. The target model verifies the proposed block in a single forward pass, accepting tokens up to the first rejection.

We train DFlash2 on the same target-generated responses used for the recurrent MTP drafter. Minimizing train–inference mismatch is particularly important for the diffusion drafter, as its conditioning on preceding context relies heavily on target hidden states. To this end, we extract these states using the same SGLang implementation as at inference. Appendix H.3 details the training procedure and hyperparameters.

We compare both drafters across 11 benchmarks, with at most seven proposed tokens (Table 6). We also consider the native MTP block without additional fine-tuning as a baseline. DFlash2 increases mean acceptance length by 2.75% in Instant mode but reduces it by 0.83% in Reasoning mode. Recurrent MTP performs better on MT-Bench in both modes and on mathematics in Reasoning mode. The remaining domain averages favor DFlash2.

Table 6. Comparison of recurrent MTP and DFlash2 acceptance lengths across 11 benchmarks for GigaChat 4. Both drafters propose up to seven tokens, with decoding at temperature 0. The average is computed with equal weight assigned to each benchmark.

	Instant			Reasoning		
Benchmark	**Recurrent MTP**	**DFlash2**	**Δ (%)**	**Recurrent MTP**	**DFlash2**	**Δ (%)**
Math & reasoning						
MATH-500	5.211	5.708	+9.54	4.838	4.754	−1.74
GSM8K	5.087	5.259	+3.38	4.642	4.503	−2.99
AIME25	4.325	4.810	+11.21	4.982	4.950	−0.64
Domain avg.	4.874	5.259	+7.89	4.821	4.736	−1.76
Code						
HumanEval	4.992	5.150	+3.17	4.755	4.839	+1.77
SciCode	4.404	4.449	+1.02	4.487	4.535	+1.07
MBPP	4.722	4.858	+2.88	4.514	4.532	+0.40
LiveCodeBench	4.537	4.760	+4.92	4.750	4.758	+0.17
WebDev	4.381	4.163	−4.98	4.081	3.944	−3.36
Domain avg.	4.607	4.676	+1.49	4.517	4.522	+0.09
Knowledge QA						
GPQA	4.200	4.243	+1.02	4.540	4.514	−0.57
MMLU-Pro	4.237	4.254	+0.40	4.350	4.394	+1.01
Domain avg.	4.219	4.249	+0.71	4.445	4.454	+0.20
Open chat						
MT-Bench	3.283	3.081	-6.15	3.619	3.426	-5.33
Overall avg.	**4.489**	**4.612**	**+2.75**	**4.505**	**4.468**	**-0.83**

6.4 Performance Evaluation

We compare prefill-only and decode-only throughput with GigaChat 3.5 (432B) over 10K–200K input tokens (Figure 5). The GigaChat 4 configuration runs on TensorRT-LLM with natural expert routing. Over 50K–200K contexts, prefill throughput remains approximately 79K–82K input tokens/s, while decode throughput stays around 3.0K–3.4K output tokens/s. Decode throughput is lower than GigaChat 3.5 at 10K–40K input lengths. At 200K, both phases deliver approximately 1.7× its throughput. This comparison measures the complete configurations within each phase. It does not isolate the contribution of an individual optimization.

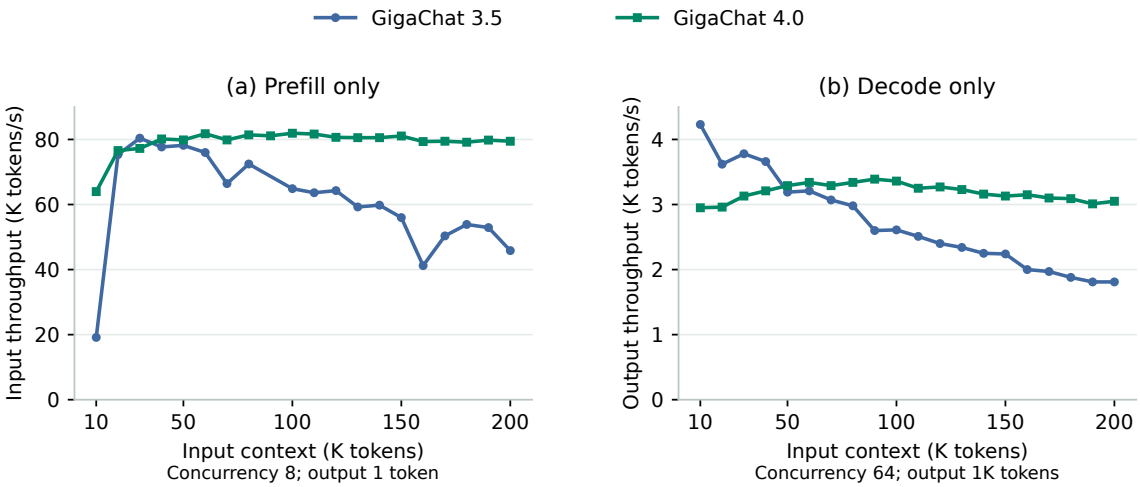

Figure 5. Prefill-only and decode-only throughput across input lengths. Both panels use four GPUs with TP4/EP4 and four-way attention data parallelism (DP4). Prefill uses concurrency 8 and one output token, decode uses concurrency 64 and 1,000 output tokens. The GigaChat 4 decode configuration uses a maximum batch and CUDA-graph size of 16 per attention data-parallel rank. Markers show digitized measurements. Lines connect available points. The two phases are benchmarked separately.

7 Evaluation

7.1 Evaluation Protocol

We adapted the public runners for FCL v4, τ^2-Bench, and τ^3-Bench [53–55] to our model APIs and execution environment. We reused the public IFBench [56] verification code and the LM Evaluation Harness implementation of JSONSchemaBench [57]. We also adapted the published Natural Plan [58] scorer for our setup.

For SWE-bench Verified [59, 60], SWE-bench Multilingual [61], Terminal-Bench 2.0 and 2.1 [62, 63], NL2Repo [64], SkillsBench [65], HealthAdminBench [66], CoworkBench [67], and JobBench [68] we used Harbor-based runners [69]. We used a Pier-based runner for DeepSWE [70]. In the results tables, we mark the benchmarks for which we increased the timeout to seven hours.

We also implemented our own evaluators for LiveCodeBench v6 [71], GraphWalks [72], AA-LCR [73], GPQA Diamond [74], IMO-AnswerBench [75], and Humanity's Last Exam [76], and scored the published AIME 2025 [77], AIME 2026 [78], and HMMT [79, 80] contest problems with our own evaluation pipeline. For StructEval [81] and Humanity's Last Exam, we evaluated only the text subsets. Our StructEval evaluator uses rule-based scoring and does not reproduce the full published benchmark score.

7.2 Reasoning Results

We report benchmark results for GigaChat 4 708B Reasoning in Tables 7 and 8. Relative to GigaChat 3.5 Ultra 432B Reasoning, GigaChat 4 708B Reasoning gained 40.96 percentage points on Terminal-Bench 2.0, 36.92 on GraphWalks, and 18.06 on τ^2-Bench. It also achieved the highest reported scores on several benchmarks, including IFBench (80.33%), FCL v4 (79.55%), POLLUX (81.46%), and CoworkBench (83.00%).

Table 7. Code and general evaluation results (Reasoning mode).

Benchmark	**GigaChat 4 708B**	GigaChat 3.5 Ultra 432B	DeepSeek V4 Flash 0731	GLM-5.3	DeepSeek 4.1 1017	Qwen 3.5 397B	Mistral 4 Large
Code							
LiveCodeBench v6	88.82	85.40	90.00	82.67	90.14	86.64	84.64
SWE-bench Verified[a]	80.89	64.70	81.80	86.67	79.40	73.00	79.60
SWE-bench Multilingual[a]	77.00	—	73.00	80.33	79.33	66.33	81.67
DeepSWE[a]	35.40	—	53.10	67.26	74.34	3.54	61.95
NL2Repo[a,b]	30.89	5.00	33.13	28.69	5.90	18.59	27.88
Terminal-Bench 2.0[a]	71.26	30.30	55.01	71.67	61.80	56.18	77.78
Terminal-Bench 2.1[a]	69.66	—	78.65	88.70	87.64	57.30	83.33
General							
IFBench[c]	80.33	77.00	63.00	55.67	75.67	74.33	71.00
JSONSchemaBench	99.28	93.01	94.00	91.64	99.31	96.26	97.38
StructEval Text	87.64	85.00	88.00	85.36	85.36	85.48	86.10
POLLUX [82]	81.46	75.50	70.00	77.96	78.64	74.16	73.18
GraphWalks	74.92	38.00	72.70	78.35	79.09	61.01	78.45
Natural Plan	94.53	80.19	91.03	80.00	93.73	92.25	94.39
NIH Complex [83]	100.00	99.16	—	96.89	96.53	98.74	99.75
AA-LCR	70.33	25.00	—	66.70	—	68.00	73.67

[a] For these benchmarks we set timeout to 7 hours. [b] Cheating attempts scored as zero. [c] Prompt-level loose.

Table 8. STEM and agentic evaluation results (Reasoning mode).

Benchmark	**GigaChat 4 708B**	GigaChat 3.5 Ultra 432B	DeepSeek V4 Flash 0731	GLM-5.3	DeepSeek 4.1 1017	Qwen 3.5 397B	Mistral 4 Large
STEM							
GPQA Diamond	82.83	82.30	91.00	90.40	90.91	85.35	87.37
HMMT (pass@8)	98.33	93.00	100.00	100.00	100.00	98.33	100.00
HMMT (mean@8)	93.75	—	98.75	92.08	97.92	82.71	99.58
AIME 2025 (pass@8)	96.67	93.00	96.67	100.00	100.00	96.67	100.00[a]
AIME 2025 (mean@8)	93.33	89.00	90.00	93.13	97.60	93.75	97.08[a]
AIME 2026 (pass@8)	96.67	97.00	100.00	100.00	100.00	—	100.00[a]
AIME 2026 (mean@8)	95.00	91.00	95.00	95.10	98.23	—	99.17[a]
IMO-AnswerBench	82.69	73.00	89.80	88.50	92.69	83.22	78.20
HLE[a]	23.91	13.16	37.80	39.62	43.42	28.36	30.40
Agentic							
FCL v4	79.55	58.59	73.12	68.29	72.18	69.10	70.68
τ^2-Bench (mean@3)	83.31	61.58	83.30	78.70	79.43	78.83	80.60
τ^3-Bench Banking (mean@3)	18.03	8.93	20.90	56.00	21.65	12.03	24.74
CoworkBench	83.00	25.00	78	73.79	77.82	64.11	59.27
SkillsBench (no skills)	33.08	—	29.89	45.47	45.47	42.06	37.00
SkillsBench (skills)	58.24	—	59.81	61.54	—	42.06	46.90
HealthAdminBench[b]	40.74/88.27	—	25.92/78.36	45.93/88.37	—	41.48/86.12	—

[a] Evaluated without tools. [b] Values are reported as success rate / mean.

7.3 Instant Results

We report results for GigaChat 4 708B Instant and the comparison models in Tables 9 and 10. Relative to GigaChat 3.5 Ultra 432B Instant, GigaChat 4 708B Instant gained 41.12 percentage points on SWE-bench Verified [60], 31.67 on HMMT (pass@8), and 28.03 on GraphWalks. It also achieved the highest reported scores on several benchmarks, including StructEval Text (89.12%), POLLUX (79.65%), and CoworkBench (75.00%). On Terminal-Bench 2.1 [63], it scored 66.29%, compared with 70.77% for DeepSeek V4 Flash and 56.18% for Qwen 3.5 397B. Mistral 4 Large has no Instant mode, so it is omitted from the comparison.

Table 9. Code and general evaluation results (Instant mode).

Benchmark	**GigaChat 4 708B**	GigaChat 3.1 Ultra 702B	GigaChat 3.5 Ultra 432B	DeepSeek V4 Flash 0731	Qwen 3.5 397B
Code					
LiveCodeBench v6	78.77	57.00	56.20	71.18	61.52
SWE-bench Verified[a]	80.82	9.00	39.70	78.80	68.60
SWE-bench Multilingual[a]	70.67	3.33	21.60	72.33	65.00
DeepSWE[a]	19.47	0.00	0.00	28.32	0.88
NL2Repo (Full score)[a]	37.56	7.90	—	30.37	30.28
Terminal-Bench 2.0[a]	60.47	9.00	12.94	—	49.44
Terminal-Bench 2.1[a]	66.29	6.74	—	70.77	56.18
General					
IFBench[b]	50.33	29.00	43.66	40.00	53.00
JSONSchemaBench	88.31	93.96	83.94	93.70	93.45
StructEval Text	89.12	84.98	74.35	85.64	83.62
POLLUX	79.65	63.28	74.25	72.22	74.29
GraphWalks	48.02	15.16	19.99	51.16	56.49
Natural Plan	52.17	32.00	66.42	41.58	45.72
NIH Complex	95.71	74.22	97.11	32.21	99.24
AA-LCR	48.00	—	27.60	—	60.00

[a] For these benchmarks we set timeout to 7 hours. [b] Prompt-level loose.

Table 10. STEM and agentic evaluation results (Instant mode).

Benchmark	**GigaChat 4 708B**	GigaChat 3.1 Ultra 702B	GigaChat 3.5 Ultra 432B	DeepSeek V4 Flash 0731	Qwen 3.5 397B
STEM					
GPQA Diamond	70.20	56.57	61.11	74.75	83.33
HMMT (pass@8)	86.67	45.00	55.00	70.00	78.33
HMMT (mean@8)	63.75	—	—	44.38	63.33
AIME 2025 (pass@8)	93.33	73.33	93.30	86.67	93.33
AIME 2025 (mean@8)	68.75	37.08	68.00	48.23	80.00
AIME 2026 (pass@8)	96.67	66.67	83.30	93.33	93.33
AIME 2026 (mean@8)	80.83	39.17	67.00	60.42	86.25
IMO-AnswerBench	60.09	28.75	32.00	41.91	72.38
Humanity's Last Exam[a]	8.20	4.00	3.94	10.43	19.83
Agentic					
FCL v4	63.62	—	51.57	65.81	—
τ^2-Bench (mean@3)	66.75	33.00	68.71	77.13	78.73
τ^3-Bench Banking (mean@3)	15.00	5.50	5.15	—	12.71
CoworkBench (Pi)	75.00	—	19.76	54.84	63.31
SkillsBench (no skills)	33.93	0.51	—	27.87	—
SkillsBench (skills)	54.24	1.23	—	54.32	—
HealthAdminBench[b]	40.74/86.99	—	—	23.73/70.14	33.33/—
JobBench	33.59	—	—	29.64	16.85

[a] Humanity's Last Exam, evaluated without tools. [b] Values are reported as success rate/mean.

8 Conclusion

In this work, we presented GigaChat 4, a hybrid mixture-of-experts model with 708B total parameters, 41B active parameters per token, and a native one-million-token context window. Our training recipe combines long-context adaptation, domain-specific reinforcement learning, and multi-teacher on-policy distillation. The resulting model supports both Reasoning and Instant modes.

To support large-scale training and low-latency deployment, we implemented FP8 training, state-based context parallelism for Gated DeltaNet, fused DSA scoring and top-k selection, and speculative decoding. To facilitate further research and adoption, we release the model weights along with native inference integration and the fused DSA indexer via public pull requests.

Under the evaluation setup in Section 7, GigaChat 4 Reasoning performs comparably to DeepSeek V4 Flash 0731 and Mistral 4 Large on selected coding and agentic benchmarks.

A Pretraining Data

A.1 Sources and Mixture

The pretraining corpus contains approximately 15.35 trillion tokens. We report the corpus size and sampling weight of each domain in Table 11. We set these weights according to source size and the desired number of passes, assigning more repetitions to higher-quality and more complex data within the 20T-token pretraining target (Table 1). Sampling weights therefore differ from the domains' shares of the available corpus.

Code-related web pages are included in the code domain. Synthetic web, code, and mathematics data are counted in their respective domains.

Table 11. Pretraining corpus sizes and domain sampling weights. Weights are normalized over the mixture. Synthetic data are included in their corresponding domains.

Domain	Corpus tokens (T)	Mixture weight (%)
Web	8.202	57.00
Code	1.488	17.00
OCR/PDF documents	1.711	13.00
Mathematics	0.502	6.00
Multilingual web	3.069	2.95
Reference and domain-specific sources	0.374	4.05
Total	15.346	100.00

A.2 Organic and Synthetic Data

The corpus combines naturally occurring documents with synthetic material in three domains. Synthetic web data include rewritten documents, explanatory notes, and question–answer pairs grounded in source text. These transformations turn noisy or weakly structured material into more explicit explanations and question–answer formats. Related approaches are studied in web-corpus curation [7, 84].

For code, instruction reversal starts with an existing source file and asks an LLM to produce a task description and an explanation of the implementation. Python is a major component

of this stream. Mathematics synthesis produces exercises, multiple-choice questions, proofs, and worked solutions from mathematical web text and OCR-recovered documents. Related source-grounded mathematical synthesis is explored in MegaMath [84].

More extensive capability-specific synthesis belongs to midtraining and is described in Appendix B.

A.3 Web Data

For Common Crawl [4] data, our pipeline extracts document content and applies quality and domain-specific selection followed by deduplication. Related approaches include DCLM [5], RefinedWeb [6], FineWeb and FineWeb-Edu [9], FineWeb-HQ [10], HPLT [8], and Nemotron-CC [7].

A.3.1 Language Identification

We identify document languages with GlotLID [85], a fastText-based classifier [13] with broad multilingual coverage, and route documents to language-specific filtering and sampling streams.

English and Russian are the largest web streams. Additional streams cover Arabic, Chinese, French, German, Japanese, Portuguese, and Spanish. We also retain languages spoken in countries of the former Soviet Union, including Armenian, Azerbaijani, Belarusian, Georgian, Kazakh, Kyrgyz, Tajik, Turkmen, Ukrainian, and Uzbek. The corpus also includes documents in several regional languages spoken in Russia, including Bashkir, Chechen, Chuvash, Mari, Ossetian, Tatar, and Yakut. Language identification determines routing. Topic relevance and document quality are assessed separately.

A.3.2 Parsing

Our GPU-based LLM parser extracts the main content of an HTML page and renders it as Markdown, preserving prose, tables, code, and mathematical expressions [11, 12]. We distribute inference across the processing cluster and use speculative decoding to improve throughput. Resiliparse [86] provides a rule-based fallback.

We manually inspect mathematical pages for lost formulas and compare extraction quality against rule-based alternatives using the benchmarks below.

A.3.3 Parsing Benchmark

We evaluate the GigaChat 4 parsing pipeline on our internal Russian–English benchmark and WebMainBench [11]. Human annotators select the main-content elements of each page in the internal benchmark, which contains 542 pages with an emphasis on mathematical and technical content. We use a 545-page subset of WebMainBench. We report document-level macro F1 based on normalized word 5-gram overlap in Table 12.

Table 12. Parsing F1 on the internal benchmark and WebMainBench.

Parser	Internal benchmark	WebMainBench
readability-lxml	0.720	0.798
magic-html	0.727	0.828
Resiliparse [86]	0.755	0.797
Trafilatura [87]	0.775	0.802
Our pipeline	0.841	0.869

A.3.4 Effect of Parsing on Pretraining

We also compare parsing methods using the 3B / 1T proxy configuration introduced for GigaChat 3.5. Our Markdown representation, which preserves mathematical formulas, yields an approximately one-percentage-point, statistically significant improvement on MMLU [88] over Resiliparse and Trafilatura.

A.3.5 Quality Filtering

We train a document-quality classifier on LLM annotations. The annotation prompt asks for educational value, instructiveness, and the presence of reasoning or substantive explanations. These signals distinguish a useful document from one that merely contains topic-related vocabulary. Classifier-based selection has also been studied in DCLM, FineWeb-Edu, and FineWeb-HQ [5, 9, 10].

For GigaChat 3.5, we fine-tuned the quality classifier from Qwen3-0.6B [89] and selected a score threshold of 0.5 through experiments with 7B models trained on 400B tokens. GigaChat 4 retains this threshold for several web streams and adds the domain-specific selection stages described below.

A.3.6 Code-related Web Data

For GigaChat 3.5, we developed an iterative two-stage pipeline to select programming documentation, tutorials, discussions, and explanatory articles. We began with known programming-related hosts in the English and Russian web and trained a fastText classifier [13]. Hosts and URL prefixes with a high proportion of positive pages were inspected, and relevant sections were added to the positive training set. The GigaChat 3.5 mining procedure used a 10% positive-page threshold for this inspection. Repeating this process expanded coverage beyond the original seeds, following the domain-mining approach of DeepSeekMath and DeepSeek-Coder-V2 [34, 90].

The resulting fastText model retrieves candidates at high recall. An LLM-based classifier makes the stricter topic decision. For GigaChat 3.5, Qwen3-32B supplied training labels and a fine-tuned Qwen3-0.6B made the final binary classification [89].

For GigaChat 4, we extend this pipeline with a quality classifier implemented as an MLP over document embeddings. It selects a smaller, higher-quality subset from the broader programming-related corpus. Coverage also includes Chinese, German, Japanese, Spanish, and other languages. Programming-oriented web curation is discussed in the Nemotron-CC-Code release [91]. Repository-code filtering and educational selection are described in StarCoder2, OpenCoder, and Stack-Edu [14, 15, 92].

A.3.7 Mathematical Web Data

For GigaChat 3.5, we selected mathematical web pages from quality-filtered documents in two stages. A fastText classifier [13] retained approximately 2.5% of documents as candidates, and a stricter classifier fine-tuned from Qwen3-8B [89] retained approximately 3.5% of those candidates. We also translated selected English documents into Russian to improve coverage of mathematical explanations and terminology.

For GigaChat 4, we extend collection through text extraction from images, targeted crawls of mathematical sites, multilingual mathematics mining, and selection from parsed PDFs. Image extraction recovers material absent from the text-only view of a page. A dedicated multilingual mathematics classifier selects relevant Common Crawl documents independently of the broader STEM classifier in Appendix A.4. Separate classifiers operate on parsed PDF documents, adding approximately 8 million documents to the mathematical collection.

Relative to GigaChat 3.5, the mathematical web corpus collected through our pipeline grows from 140B to 300B tokens. This figure covers our web collection. The complete mathematics domain in Table 11 contains approximately 502B tokens and also includes other mathematical sources and synthetic material.

These procedures address problems studied in OpenWebMath [93], DeepSeekMath [34], FineMath [92], MegaMath [84], and Nemotron-CC-Math [94]: finding mathematical content at web scale, retaining notation during extraction, and selecting useful reasoning material. Candidate selections are compared using the proxy experiments and mathematical benchmarks in Appendix A.9.1.

A.4 Multilingual Data

The multilingual web stream supplements the English and Russian core with the languages listed in Appendix A.3.1. Large-scale multilingual web processing and model-based data selection are also studied in HPLT and FineWeb-HQ [8, 10].

We separately collect multilingual STEM documents from Common Crawl [4]. A high-recall fastText filter [13] retrieves candidate documents, after which an MLP over document embeddings assigns the final topic labels.

In Table 11, multilingual mathematical and programming-related documents are counted in their respective domains. The remaining multilingual streams form the multilingual web domain.

A.5 PDF Documents

The PDF stream contains openly licensed long-form general, scientific, and technical documents, with substantial English and Russian coverage. Extraction and OCR recover readable text while preserving equations, tables, and document structure. This complements HTML parsing: important technical material appears in scanned pages or embedded images and cannot be recovered reliably from plain text alone. Related work on large-scale PDF corpora and document extraction includes FinePDFs [95] and olmOCR [96].

Extracted documents pass through quality filtering, deduplication, and benchmark decontamination. For GigaChat 4, we additionally train topic classifiers on parsed PDF text to select mathematical material, as described in Appendix A.3.7.

A.5.1 Pretraining Experiments

We evaluate PDF data using the 7B / 400B proxy configuration introduced for GigaChat 3.5, with a mixture of 75% web data and 25% PDF data. Our corpus improves MMLU [88] accuracy by 1.4 percentage points over FinePDFs and 0.9 points over olmOCR, and ARC-Challenge [97] accuracy by 1.7 points over FinePDFs. Our OCR pipeline also produces higher-quality document extractions than the open-source alternatives we evaluated.

A.6 Reference Sources

Reference and domain-specific collections contain structured factual material, explanatory articles, question–answer content, and specialized documents. We convert each source to a common document representation, then apply quality filtering, exact and near-duplicate removal, and benchmark decontamination. We report their combined size and sampling weight in Table 11.

A.7 Source Code

We collect source files distributed under open-source licenses from public repositories, including GitHub. Following the repository reconstruction approach of StarCoder2 [14], metadata associate files with repositories, extensions, and programming languages. We recover repository snapshots at fixed points in time, supplement them with direct repository crawls, and filter files by language and extension. Near-duplicate removal limits repeated copies of the same implementation. The mathematical-code component also covers symbolic and formal reasoning. Related corpus construction is described in Llemma and Proof-Pile-2 [98].

Filtering combines sample quality, complexity, algorithmic content, and syntactic validity. Rules inspired by OpenCoder [15] check file length, character diversity, and language-specific properties, together with additional corpus-specific rules. We downsample common languages such as Java and JavaScript so that their volume does not dominate the selected corpus. Educational code selection is also explored in Stack-Edu [92].

The final selection uses classifiers trained on LLM labels. The labeling prompt rates educational value, complexity, correctness, and robustness. A weighted combination of these signals forms the target score. For GigaChat 3.5, we evaluated source-code filters with 7B models trained on 400B tokens. For GigaChat 4, we use the 10B proxy trained on 200B tokens (Appendix A.9). The resulting corpus covers more than 400 programming languages. Midtraining applies stronger quality and complexity selection to code, as described in Appendix B.

A.8 Deduplication and Decontamination

A.8.1 Exact and Near-Duplicate Removal

We remove exact duplicates using MD5 hashes of normalized document text. Near-duplicates are detected with MinHash locality-sensitive hashing [16] over word 5-grams, using 112 hash functions arranged in 14 bands and targeting a Jaccard similarity of approximately 0.7.

A.8.2 Benchmark Candidate Retrieval

We separate candidate retrieval from the final contamination decision. Exact n-gram overlap is useful for finding copied benchmark text [99], but rephrasing can evade purely lexical checks [100]. A benchmark registry therefore records the parts of each item that indicate a

leak when they occur together, such as a problem statement and its solution, as well as the context size needed to inspect them.

We retain the two complementary retrieval routes developed for GigaChat 3.5. The first uses an inverted index of benchmark word 5-grams. It retrieves an item when each required part reaches 50% coverage, then checks a sliding document window for at least 70% unigram coverage and a normalized longest-common-subsequence score of at least 0.7 for every part. All required parts must match within the same window. The second route uses an Aho–Corasick automaton [101] to scan for exact fingerprints, including problem statements, canonical solutions, docstrings, assertions, and test inputs and outputs. Item-specific checks confirm these hits.

A.8.3 Final Leak Decision

Candidate pairs from both routes are merged and passed to an LLM with the benchmark item and up to 6,000 tokens of surrounding document context. A document is removed if any candidate pair receives a leak score of at least 0.4. An invalid classifier response is conservatively treated as a leak.

A.9 Data Validation Experiments

For GigaChat 3.5, data experiments used dense 3B and 7B models, followed by 120B MoE models for larger mixture comparisons. Staged mathematics experiments continued a 7B model trained on 200B web tokens for another 200B tokens on a 70% web / 30% mathematics mixture.

For GigaChat 4, we extend this suite with 10B MoE models trained on 200B tokens and 37B MoE models trained on 370B tokens. We group the configurations by the release that introduced them in Table 13. Controlled comparisons hold the model and optimizer configuration fixed while varying the data.

Table 13. Proxy training configurations introduced for GigaChat 3.5 and GigaChat 4. Token budgets are rounded. Batches count sequences and warmup counts optimizer steps.

Model	Budget	Context	Batch	Peak → final LR	Warmup
Introduced for GigaChat 3.5					
3B dense	200B / 400B / 1T	2,048	2,048	$2\times10^{-3} \rightarrow 3\times10^{-5}$	5,000
7B dense	200B	4,096	1,024	$2\times10^{-3} \rightarrow 3\times10^{-5}$	5,000
7B dense	400B	2,048	2,048	$2\times10^{-3} \rightarrow 3\times10^{-5}$	5,000
120B MoE	2T	4,096	8,192	$1.02\times10^{-3} \rightarrow \approx10^{-5}$	2,000
Added for GigaChat 4					
10B MoE	200B	4,096	1,024	$10^{-3} \rightarrow 10^{-5}$	5,000
37B MoE	370B	4,096	2,048	$1.5\times10^{-3} \rightarrow 1.5\times10^{-5}$	2,500

All configurations use decoupled AdamW with $\beta_1 = 0.9$ and $\beta_2 = 0.95$, gradient-norm clipping at 1.0, bfloat16 mixed precision, and cosine learning-rate decay after warmup. The 120B configuration also uses sequence-length warmup to 4,096 tokens. The other proxies use fixed context lengths.

A.9.1 Benchmarks

We evaluate candidate datasets on the benchmarks in Table 14, using our task configurations built on the Language Model Evaluation Harness [102].

Table 14. Benchmarks used in pretraining data experiments. Parentheses give the number of few-shot demonstrations. CoT indicates chain-of-thought prompting.

Group	Benchmarks
Knowledge and commonsense	MMLU [88] (5); FakeMMLU [103] and BabyMMLU2 [104] (5 each); MMLU-Pro [105] (CoT, 5); AGIEval [106] (0); ARC-Easy and ARC-Challenge [97] (5 each); HellaSwag [107] (5); WinoGrande [108] (5).
Question answering	TriviaQA [109] (5); NQ-Open [110, 111] (5); DROP [112] (5).
Reasoning	BBH [113] (CoT, 3); GPQA main and diamond [74] (CoT, 5 each).
Mathematics	GSM8K [114] (CoT, 8); MATH with Minerva prompting [115, 116] (CoT, 4); MATH-500 [117] (CoT, 4); DOoM mathematics and physics [118] (CoT, 3 each); MGSM English, Russian, and Chinese [119] (CoT, 8 each).
Code	HumanEval [120] and HumanEval+ [121] (3 each); MBPP [122] and MBPP+ [121] (3 each); MultiPL-E [123] (0); LiveCodeBench [71] (3); CRUXEval [124] (CoT, 3); BigCodeBench [125] (0).
Russian	ruMMLU [88] (5); Russian GSM8K [114] (CoT, 8) and MATH [115, 116] (CoT, 4); RuCoLA [126] (5); SLAVA [127] (5); SAGE [128] (5).
Translation	Bouquet [129] (5).

A.9.2 Evaluation Protocol

For multiple-choice classification, we select the answer with the highest model probability and report accuracy. Generative tasks use eight responses per item. We report mean@8 by averaging the task score across responses and items. We use each benchmark's scoring rule, including execution tests for code, chrF for translation [130], and correction scores for SAGE [128].

We compare the final checkpoints of candidate and baseline runs using the same prompts, few-shot settings, and scoring rules. Statistical significance is assessed with a paired bootstrap over benchmark items at the 99% and 99.9% confidence levels.

A.10 Infrastructure

We process the corpus using distributed MapReduce operations on a cluster with approximately 50,000 CPU cores, 300 TB of RAM, and 5 PB of storage. The same infrastructure supports normalization, language identification, fastText inference, exact and near-duplicate removal, and benchmark candidate retrieval. Expensive model-based parsing and classification are separate stages whose outputs are joined back to the document tables.

Airflow orchestrates the pipeline from collection and extraction through filtering, deduplication, tokenization, and mixture construction. Versioned source tables and processing configurations make it possible to reconstruct an input mixture and rerun individual stages when a classifier, crawl, or filtering rule changes. Data experiments use fixed model and optimizer presets while replacing the candidate source or sampling recipe, keeping the data change separate from training-configuration changes.

B Midtraining Data

B.1 Data Preparation

We select high-quality code-related web pages and reasoning text from Common Crawl, together with Git repository code and books. For mathematics, an LLM rewrites formulas in LaTeX and incorporates extracted image content into the document. A ranking classifier selects mathematical tasks and documents for the mixture.

Synthetic data include questions extracted from documents, new problems generated with Magpie [25] and PromptCoT [26], and educational rewrites. We generate solutions and explanations for both extracted and newly constructed tasks. Validation uses model assessment, executable tests, observed execution outcomes, or SQL-based checkers. Structural checks remove formatting errors and generation artifacts. Deduplication and benchmark decontamination follow the pretraining procedures (Appendix A.8).

B.2 Data Synthesis Pipelines

B.2.1 Retrieval-Grounded Legal QA

We generate answers to everyday legal questions using retrieved documents. The model identifies legally relevant facts and reformulates the question into subquestions using legal terminology. Hybrid BM25 and dense retrieval finds relevant articles and passages, which are deduplicated before analysis. The model checks whether the retrieved provisions apply, considers conditions and exceptions, and retrieves additional provisions when needed. It then generates an answer with references to the supporting articles.

B.2.2 PromptCoT-Based Financial Synthesis

We generate financial problems with PromptCoT [26]. Given a set of financial concepts, the model first develops a rationale for combining them into a problem, then writes the problem statement. A separate generation step produces the solution and explanation. Problems range from elementary exercises to Olympiad-level tasks. Filtering removes contradictory statements and generation artifacts.

B.2.3 Execution-Grounded Function Traces

We generate tasks that ask for the result of a Python function call given its code and arguments. Execution in an isolated environment records executed lines, calls, local variables, return values, and exceptions. The model uses this trace to explain the computation step by step, ending with the observed result or exception. Execution-based checks remove invalid and nondeterministic examples. The cleaned corpus contains 15.2 million examples and 10.82 billion tokens.

B.2.4 Database-Backed API Interaction Tasks

We expose heterogeneous databases, including Russian-language resources, through executable APIs and generate tasks requiring data retrieval, aggregation, or modification. Each task has a generated instruction and a SQL-based completion checker. Independent agents produce trajectories from the initial database state. Completion checks and trajectory-quality assessment select the training examples. From 365 thousand unique tasks, the pipeline yields approximately 165 thousand trajectories and 566 million tokens. Russian-language instructions account for 93.8 thousand trajectories and 284 million tokens.

B.2.5 Python Bug-Fixing Tasks

We start from Python solutions that pass their pytest suites, build abstract syntax trees, and collect per-test coverage. Code structure, data dependencies, and coverage guide the selection of locations for model-generated logical errors. We check syntax and rerun tests, preferring mutations that fail a small subset of tests. For each accepted bug, the model writes a GitHub-style issue with reproduction steps and expected and observed behavior, without revealing the fix. It then generates an explanation and a patch in unified diff format. Reverting the mutation provides a reference patch. Model-generated patches are verified by running the tests.

B.3 Midtraining Data Recipe Development

All comparisons continue from a common 37B MoE checkpoint with a 32,768-token context window and approximately 40B training tokens. We evaluate the final candidate and baseline checkpoints at the same token budget. The benchmark suite and scoring protocol follow Appendix A.9.1, with additional zero-shot RULER [27], HELMET [28], and SCROLLS [29] evaluations.

We run separate comparisons for mathematics, code, and reference and domain-specific data. In the mathematics and code experiments, we increase the target domain's share and proportionally rescale the other streams. Candidate sources replace part of the baseline domain data while its total allocation stays fixed. All percentages below refer to the full training mixture.

B.3.1 Mathematics

Mathematics accounts for 50% of training tokens. Candidate runs allocate 25% to baseline mathematics data and 25% to the source under evaluation. Sources include curated web problems and synthetic problems generated with Magpie and PromptCoT, paired with model-generated solutions.

B.3.2 Code

Code accounts for 40% of training tokens. Standard comparisons allocate 20% to baseline code and 20% to candidate sources. Targeted additions, such as execution traces and bug-fixing tasks, use 4% candidate data and 36% baseline code.

B.3.3 Reference and Domain-Specific Data

We form the baseline by removing the existing reference and domain-specific streams and renormalizing the remaining mixture. Candidate runs allocate 5% to the source under evaluation and 95% to the baseline streams. This setup evaluates synthetic finance problems, retrieval-grounded legal question answering, and educational rewrites of web documents.

C Long-Context Data

C.1 Organic Long Documents

The OCR-extracted PDF pool contains approximately 378.2 million documents and 3.222 trillion tokens before length selection. It includes scientific publications, books, educational material, and other domain-specific documents. Other sources include long web pages, academic documents in LaTeX and Markdown, complete code repositories, and concatenated

books. The books used in this stream are distributed under open licenses. We use the OpenAlex [30] citation graph to rank scientific papers by PageRank and assemble articles with the full text of cited papers.

After restricting document lengths to 8k–2M tokens, benchmark decontamination, and quality filtering, the selected document pool contains 76,702,419 documents and 2.036 trillion tokens. Length-aware sampling increases the share of long documents. In length-sampling comparisons, we keep the quality threshold and the other mixture components fixed.

C.2 Repository-Level Code

We assemble complete repositories from file contents and metadata. A code quality classifier scores each unique file. A repository's quality score is the fraction of its tokens belonging to files that pass the classifier. We retain repositories with a score of at least 0.80, valid UTF-8, and lengths of 8k–2M tokens. Benchmark decontamination removes a whole repository if any of its files matches a benchmark. All files in accepted repositories are preserved.

The resulting pool contains approximately 8.5 million repositories and 417 billion serialized tokens. Each example starts with the repository path, followed by file blocks containing their paths and contents. File order is deterministically shuffled, with blank lines separating blocks. We draw 40B tokens for the 128k stage and a 63B-token subset for the native 1M stage, increasing the contribution of 128k–512k-token repositories. These allocations use serialized token counts.

C.3 Synthetic Question Answering

We generate questions with answers grounded in source passages and marked with citations. We control the number of question–answer pairs per document or chunk to retain substantial exposure to the source text.

C.3.1 Questions over Long PDFs

We augment long PDFs with a small set of cited questions and answers, appended after the document. Answering requires locating the supporting passages in the preceding text.

C.3.2 Questions over Web Passages

We split parsed web pages into chunks and shuffle their order. Questions require collecting evidence across these passages, and answers cite the supporting chunks. One variant retains the separate chunks and citations. Another combines them into coherent multi-source documents. Examples with missing question–answer pairs are removed from the latter variant.

C.3.3 Scientific and Educational Documents

We select scientific articles through PageRank on the OpenAlex citation graph. Individual long articles span 90k–512k tokens. A second variant concatenates an article with the full texts of cited works, each identified by its citation. For OCR-extracted STEM textbooks and encyclopedic and web material, two generator models produce paragraph-level questions and answers. Paragraphs retain their original order, while questions appear later in the document, separated from their supporting passages.

C.4 Long-Context Data Recipe Development

C.4.1 Repository Selection

Repository ablations use 37B proxies with 128k context, adding 10 percentage points of repository data within the 30% long-context allocation. At a matched 20B-token budget, quality-filtered complete repositories improve retrieval without the reading-quality regression observed with uncurated repositories.

Sweeping the repository quality threshold from 0.70 to 0.95 shows that strict cutoffs of 0.90–0.95 reduce multi-key retrieval gains. We therefore select 0.80. Removing whole contaminated repositories gives comparable downstream quality to removing only the matched files, and preserves complete repositories. Uniform repository sampling degrades long-context reasoning at 512k, motivating the increased sampling of longer repositories for the 1M stage.

C.4.2 Long Documents and Synthetic Questions

A separate 10B experiment at 128k evaluates adding a source of 128k–2M-token documents. It improves multi-key retrieval and motivates including longer documents in the final mixture. Detailed scores are reported in Appendix D.2.4.

Adding sparse cited questions over PDFs improves NIH [83] by up to 6.0 points and HELMET [28] by 1.7 points. The scientific and educational reading streams improve SCROLLS [29] by 2.3–2.8 points, LongBench v2 [131] by 3.2–3.3, RULER [27] by 1.6, NIH by 0.5–1.4, and InfiniteBench [132] by 2.1. Short-context scores also improve: MMLU [88] by 0.5 points and MMLU-Pro [105] by 0.7.

D Architecture and Context-Extension Ablations

D.1 Pretraining Stability and Architecture

D.1.1 Proxy Methodology and Scale Transfer

Our procedure adapts the small-scale instability-proxy methodology of Wortsman et al. [20]: models are stressed across learning rates, and loss is interpreted together with activation and optimization trajectories. Initial experiments on 9B proxies exposed failure modes at elevated learning rates. A subsequent paired 27B-to-36B depth increase approximated the architectural transition from GigaChat 3.5 to GigaChat 4. The deeper proxy inherited a learning rate close to the smaller model's stability boundary, making the comparison sensitive to the additional depth. Selected normalization and initialization candidates were also evaluated at 120B before the production recipe was finalized.

The transfer criterion was the relative ordering of candidate features, their failure signatures, and the change in their stable operating region. Endpoint loss alone was insufficient: candidates that appeared competitive early in training could later develop increasing max-vio or deep-layer activation tails. We therefore considered stability trajectories together with language-model loss and the available downstream evaluations.

D.1.2 Gate Parameterization and ProRes

The normalization experiments compared standard and zero-centered GatedNorm, as well as pre-only and sandwich placements, on small stability proxies. GatedNorm provides the explicit rescaling mechanism proposed by Qiu et al. [17]. Our factor-two sigmoid convention

is inherited from GigaChat 3.5 [3]. The comparison favored standard GatedNorm for the final stability recipe instead of zero-centered variation. Sandwich normalization controls sublayer outputs before residual addition.

Our ProRes implementation [18] uses the layer-dependent schedule in Equation (5). The warm-up duration T refers to the deepest decoder layer. The production configuration uses a starting coefficient of 0.1 and $T = 15{,}000$, together with zero initialization of the post-attention normalization scale, adapting the attention initialization of MAI-Thinking-1 [19]. GatedNorm learns feature-wise fading, while ProRes controls the magnitude of residual updates during early optimization.

A 27B hybrid-MoE diagnostic comparison illustrates the effect of the combined initialization and warm-up recipe. Both runs used seed 2048, the same data, AdamW, a peak learning rate of 2.375×10^{-3}, and the same learning-rate warm-up and decay schedule. Both included block-output $\mathcal{Z}$-loss. The intervention added ProRes with $\alpha_0 = 0.05$ and $T = 15{,}000$, together with zero initialization of the post-attention normalization scale.

Both runs completed approximately 50,000 updates. Their training cross-entropies were close, while the joint intervention reduced deep residual activation maxima and the router-bias magnitude (Table 15 and Figure 6). Both recipes remained stable at this learning rate.

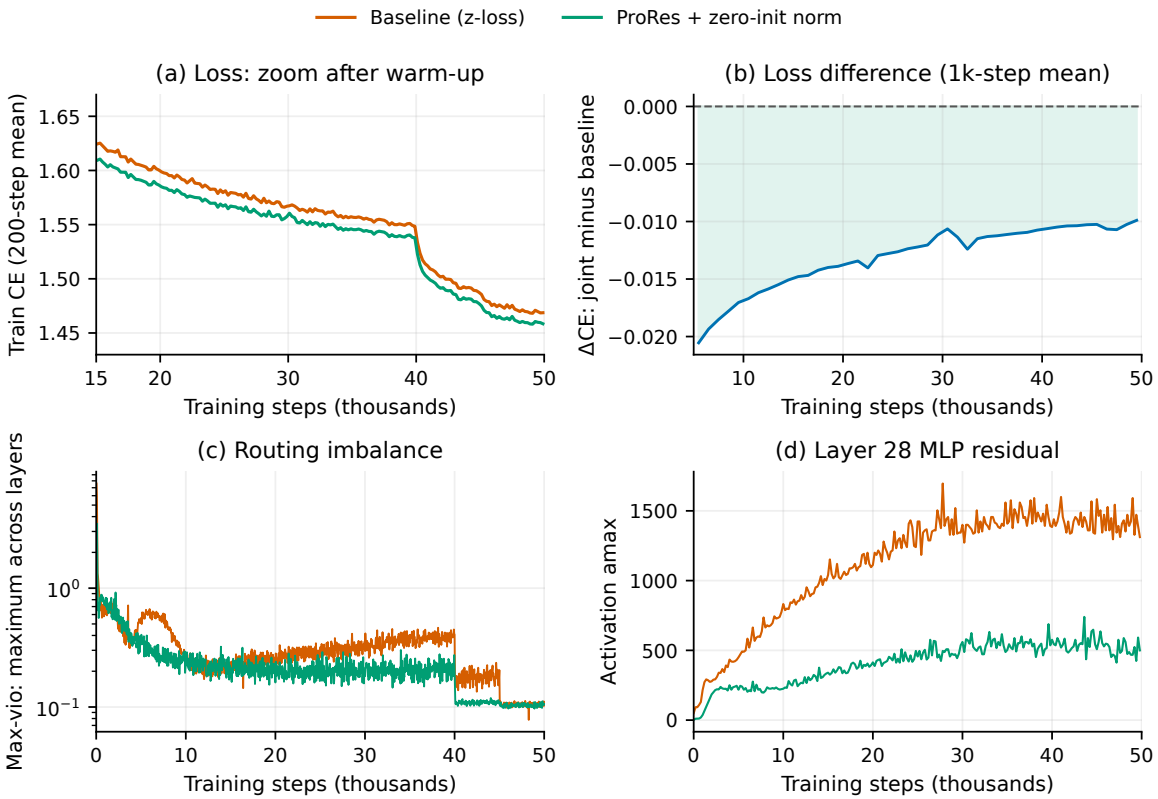

Figure 6. 27B diagnostic comparison. (a) Training CE after warm-up (15k–50k, 200-step means; zoomed axes). (b) Joint-minus-baseline CE (1,000-step means; negative is better). (c) Maximum per-layer dispatcher max-vio, log scale, without temporal smoothing. (d) Layer 28 MLP residual activation maxima at logged steps.

Table 15. 27B baseline versus the joint ProRes/zero-init recipe. CE is averaged over steps 49,000–49,999. Activations and bias are the last logged values. Peak dispatcher max-vio is maximized over layers and logged steps.

Metric	Baseline	Joint recipe
Train cross-entropy, last 1,000 steps	1.4692	1.4593
Layer 28 MLP residual activation maximum	1,312	500
Maximum router bias across layers	3.00	1.54
Peak per-layer dispatcher max-vio	7.60	3.49

D.1.3 Initialization, Activation Clipping, and Soft-Capping

On the 36B depth-increased proxy, retaining the 27B initialization scale led to divergence at the inherited learning rate. Multiplying that scale by two or four improved the observed stability trajectory, and the fourfold variant showed less growth in the monitored imbalance. The final production configuration used an initialization standard deviation of 0.006 and zero initialization of the post-attention normalization scale.

SwiGLU clipping at 10 was retained from GigaChat 3.5 [3]. The stability experiments also evaluated output-logit soft-capping, with threshold 30 retained in the final configuration. These controls bound different quantities: clipping limits the multiplicative factors inside the expert, while soft-capping bounds the vocabulary logits. Their purpose is to limit local activation tails without clipping the entire residual stream.

D.1.4 Learning Rate and Local Regularization

The production comparison included a conservative branch at 1.0×10^{-4} and two branches at 1.4×10^{-4}. The conservative branch progressed more slowly, while the selected higher-rate recipe combined better loss with lower routing imbalance. The higher-rate branches nevertheless differed in several settings, so their comparison does not isolate a causal effect of learning rate alone. The final peak rate of 1.4×10^{-4} was retained together with DLR and local regularization. The use of elevated learning rates to expose failure modes follows Wortsman et al. [20].

The selected DLR configuration used a 100-observation window, a 5,000-step warm-up, a trend threshold of 0.05, and multiplicative reductions by 0.98, with a minimum learning-rate multiplier of 0.3. Block-output hidden $\mathcal{Z}$-loss coefficients were 10^{-7} for attention, routed MoE, and dense MLP outputs. Projection-specific penalties were introduced subsequently where monitoring identified persistent activation growth.

At step 44,625, query, key, value $\mathcal{Z}$-loss was added to selected layers. Between the peak checkpoint at step 46,000 and step 62,000, the largest singular value decreased from 1,072 to 862 in layer 62, from 646 to 522 in layer 61, and from 346 to 244 in layer 60. Stable rank, defined as $\|W\|_F^2 / \|W\|_2^2$, increased from 2.06 to 2.62 in layer 62 and from 2.95 to 3.14 in layer 60. These changes indicate partial recovery from spectral concentration while language-model loss continued to improve.

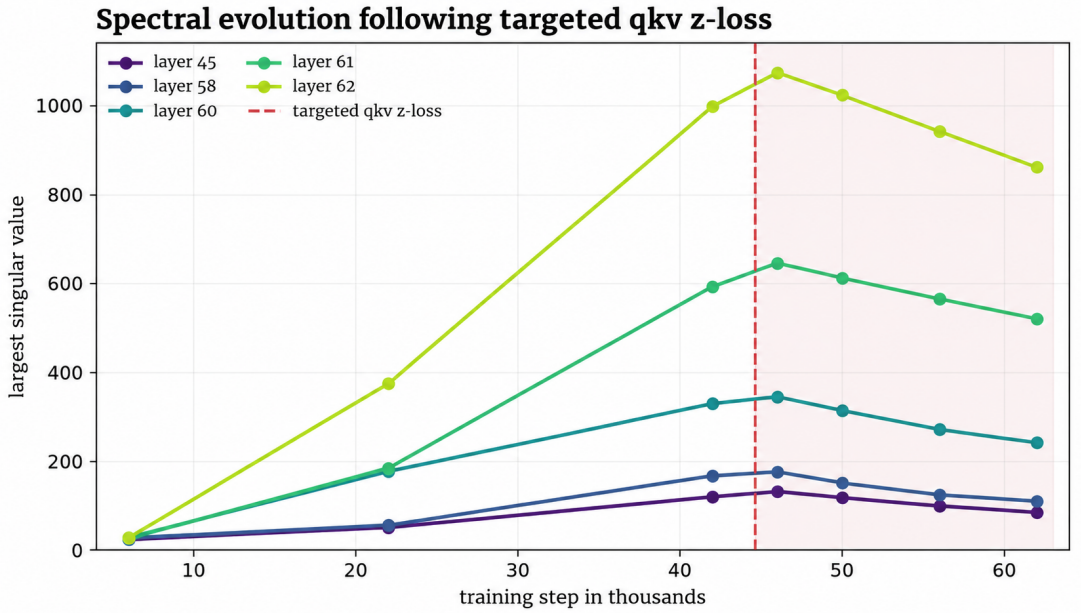

Figure 7. Largest singular values of selected qkv projections. The dashed line marks the introduction of targeted query, key, value $\mathcal{Z}$-loss at step 44,625. Lines connect measured checkpoints.

D.2 Context Extension and DeepSeek Sparse Attention

D.2.1 Overlay versus Joint Extension

Early experiments favored introducing DSA on a model already adapted to the target context over extending the context with a short-context indexer. With improved data and sufficient optimization, such joint extension also approached dense quality. In the approximately 100B-scale comparison at 512k and 4,000 updates, both models used the same data mixture and learning rate 4×10^{-5}. DSA scored 90.5% on the eight-task NIH aggregate and 81.5% on the three-task hard subset, versus 90.4% and 81.7% for dense attention. RULER-QA was also comparable (53.0% versus 52.9%). These results establish that joint extension can close the sparse–dense gap under the tested recipe.

D.2.2 Indexer Target Length

On the 10B proxy, dense 128k adaptation followed by indexer warm-up at 128k and sparse continuation reached 0.733 retrieval and 0.488 RULER-QA at 128k. Starting instead from a dense 32k checkpoint, warming the indexer at 32k, and extending with DSA reached 0.727 and 0.486 after the same 2,500-update sparse stage on the revised mixture. Therefore, transferring to a sequence length four times larger was achievable. Much shorter 4k warm-ups performed poorly in early runs that also lacked YaRN.

The production compromise was to train the indexer at 128k, the longest dense context used in the recipe, with the final 1M YaRN configuration already enabled. It then transferred directly to 1M sparse training.

D.2.3 YaRN Scaling and Direct Extension

The positional-scaling experiments compared unscaled continuation with YaRN and evaluated staged increases of the scaling factor. On a GigaChat 3.5 proxy extended from 4k to 32k, YaRN reached approximately 0.92 retrieval after 1,000 updates, whereas the unscaled variants required approximately 2,500–3,000 updates to reach that level. For the 708B model, we fixed the reference length at 32,768 and set factor 32 from the initial 128k extension

onward. The configuration used $\theta = 10^5$, $\beta_{\text{fast}} = 32$, $\beta_{\text{slow}} = 1$, and both implementation scale parameters `mscale` and `mscale_all_dim` equal to 1.0. This avoided changing the rotary basis between dense adaptation, indexer warm-up, and native 1M training.

Intermediate stages were not uniformly advantageous. On the 10B proxy, adding sparse training at 32k before sparse training at 128k improved final retrieval by 5.4 percentage points, but consumed approximately 80% more tokens. With a 128k indexer warm-up, the larger proxies supported direct sparse extension to 512k or 1M. Together with the final 708B results, this motivated omitting intermediate 256k and 512k stages from production.

D.2.4 Long-Document Data

In the 10B, 128k comparison, adding a source of 128k–2M-token documents increased DSA scores on NIH multikey-2 and multikey-3 from 0.724 and 0.250 to 0.874 and 0.454, respectively, exceeding the dense reference scores of 0.758 and 0.414. This result encouraged us to add long documents to the final mixture.

D.2.5 Learning Rate and Optimization Horizon

A matched 10B ablation compared learning rates 10^{-5} and 4×10^{-5} over 2,500 updates at 128k. DSA retrieval, RULER-QA, and MMLU-Pro increased from 0.727, 0.486, and 0.433 to 0.738, 0.488, and 0.443. The corresponding dense scores increased from 0.752, 0.474, and 0.432 to 0.772, 0.477, and 0.444. The larger rate therefore improved the reported endpoints modestly in both attention regimes, but it did not eliminate the retrieval gap.

Training dynamics provided a separate reason to extend the optimization horizon. Larger sparse models continued to improve on RULER-QA after retrieval and training cross-entropy had largely plateaued. In the 708B 1M run, the reported RULER-QA gain in the second half was approximately three percentage points.

D.2.6 Production Context-Extension Results

At 6,000 updates of native 1M sparse training, the 708B model reached retrieval aggregates of 0.990, 0.948, and 0.926 at 128k, 512k, and 1M, respectively. At 128k, the dense checkpoint preceding DSA scored 0.972 on retrieval and 0.647 on RULER-QA, compared with 0.990 and 0.645 for the final sparse model. The long-document aggregate increased from 0.444 to 0.459 and MMLU-Pro from 0.745 to 0.754.

D.2.7 Indexer Removal

To evaluate reuse of the sparse-trained trunk, the indexer was removed from the 708B production run's step-4,500 checkpoint and dense attention was restored without additional training. With the indexer present, retrieval scores at 32k, 128k, and 512k were 0.986, 0.991, and 0.958. After removal, they were 0.989, 0.985, and 0.921. The earlier dense MLA checkpoint scored 0.973 at 128k.

At 512k, restoring dense attention lowered retrieval by 3.7 percentage points but increased RULER-QA from 0.600 to 0.685. Indexer removal therefore produces a task-dependent trade-off at longer contexts. Near-parity at 128k supports reusing the trunk without the indexer in shorter-context downstream stages and motivates that option for the online reinforcement learning stage.

E Cold-Start Data Sources and Domain-Specific Pipelines

E.1 Data Source Categories

The cold-start corpus comprises five categories of data:

- **Verified synthetic trajectories.** Solutions to challenging mathematics, STEM, and algorithmic tasks, checked for final-answer correctness.
- **Environment-based and agent trajectories.** Multistep interactions with code repositories, terminals, databases, search systems, MCP, skills, external tools, and a range of agent harnesses.
- **Cleaned open datasets.** Publicly available data subjected to decontamination and quality filtering.
- **Human annotations and real-world documents.** Expert dialogues, complex instructions, and curated collections of legal, financial, and tabular documents.
- **Anonymized GigaChat user logs.** Diverse, challenging interactions with the production model.

English predominates in instruction-following tasks, international olympiad problems, algorithmic coding, and coding-agent tasks. Russian-language coverage was deliberately strengthened for general chat and finance and legal tasks.

E.2 Data Composition by Domain

- **MATH and STEM.** We select mathematical and scientific problems with worked solutions from public reasoning datasets, applying benchmark decontamination and reasoning-quality filtering (for details see Appendix E.3). We also include interdisciplinary problems collected by human annotators, public cybersecurity data, and additional human-annotated tasks.
- **Code.** The initial pool consists of programming problems collected from the internet. A four-agent pipeline comprising a planner, generator, tester, and validator produces single-pass solutions. We separately collect agentic problem-solving trajectories using both lightweight and more complex harnesses, including mini-SWE-agent [133] and OpenCode [134].
- **General Agent.** This domain comprises 13 datasets, approximately half cleaned open-source datasets and half synthetic data. Public datasets undergo quality filtering, deduplication, and benchmark decontamination. Trajectories cover tool use, web search, memory, databases, files, documents, and external APIs. We construct executable API environments based on real databases and tasks with programmatically verifiable target states. Search-tool data include thousands of augmented function schemas. Synthetic trajectories are checked for task completion and trajectory quality. We describe the file-system, memory, and multi-harness pipelines in Appendices E.4.1, E.4.2, and E.4.3, respectively.
- **SWE and Terminal.** This domain includes agentic trajectories for repository-level tasks, generated on our own infrastructure in open environments across multiple harnesses. It also covers pull-request tasks from open-source repositories, scientific terminal tasks constructed with an approach inspired by SETA [135], and agentic solutions to algorithmic problems. During solution preparation, we use open models and earlier versions of GigaChat to correct errors, and we filter trajectories to remove leaks and hacks. We provide further details on data construction, infrastructure, and evaluation in Appendix E.6

- **General.** Combines a bilingual conversational dataset with specialized instruction-following data, planning tasks, and structured-output datasets.
- **LongContext.** Includes selected datasets: anonymized session logs from the OpenCode harness, long dialogue trajectories, and chats with documents.
- **Finance & Law.** The finance and legal data comprise Russian-language and English-language tasks derived from anonymized user queries from legal information systems, open-source datasets, human annotation, and synthetic generation.

E.3 Validation Tools

We assess task solutions using reasoning-quality reward models, answer-consistency checks, and algorithmic verification where available. General-chat prompts are selected by difficulty and domain, with primary validation by the reward model most closely aligned with human judgments. We apply the benchmark decontamination procedure in Appendix A.8 and check language compliance, formatting, and invalid characters. The two reward models used for validation are described below.

Reasoning-quality reward model. An LLM fine-tuned on synthetic data and human annotations to assess whether reasoning addresses the user's request, whether intermediate steps are necessary and sufficient, and whether the reasoning is consistent with the final answer.

Human-preference reward model. An LLM fine-tuned on blinded, side-by-side human preference annotations to predict which of two responses to the same prompt a human would prefer.

E.4 Data Generation Pipelines

E.4.1 Tasks in File-System Environments

We construct synthetic agent tasks in executable file-system environments. Each task provides an initial workspace and an instruction requiring the agent to inspect, transform, or reconcile its contents and produce the requested artifacts. Tasks cover structured-data processing, document editing, configuration updates, archive inspection, and consistency checks across multiple files. Programmatic checkers assess the resulting artifacts and workspace state, including output correctness and preservation of protected inputs. We collect trajectories using multiple agent harnesses, capturing intermediate reasoning, tool calls, and environment responses. Data preparation preserves these interaction histories, repairs serialization and encoding defects, and removes selected trajectories with instruction inconsistencies or scope violations. The resulting SFT dataset contains 6,000 trajectories spanning 3,622 distinct tasks.

E.4.2 Persistent-Memory Trajectory Generation

We construct synthetic tasks for cold-start SFT using publicly available conversations as seeds for multi-turn dialogues requiring memory retrieval, storage, and updates. A task-generation pipeline produces dialogue continuations, memory-tool APIs, and verifiable expectations for tool calls and memory-state transitions. Memory is exposed in system or user messages as XML, JSON, or plain text, with varying schemas, tags, and field names. A multi-agent reasoning pipeline solves the generated tasks through separate analysis, planning, action-selection, and verification stages. We assemble the reasoning and tool interactions into 5,500 training trajectories totaling approximately 30.6 million tokens.

E.4.3 Multi-Harness and Targeted Agentic Tasks

We include four complementary agentic SFT collections in the cold-start training mixture, targeting general-purpose agent workflows, skill-guided execution, command-line tool use, and recurrent errors observed in earlier GigaChat models.

For these collections, we construct synthetic tasks together with reference solutions and executable checks. We prioritize tasks whose outcomes can be evaluated algorithmically, reducing dependence on LLM-as-a-judge assessment. Teacher models solve the tasks in executable environments, and successful solutions are converted into training examples that preserve intermediate reasoning, tool calls, and environment observations.

For the general-purpose collection, we use diverse harnesses, including the open-source systems deepagents [136], pi [137], Hermes [138], OpenClaw [139], Qwen Code [140], mini-SWE-agent [133], and OpenHands [141], as well as internal harnesses. These systems differ in their instructions, tool interfaces, agent loops, and history-management conventions.

The skill-oriented collection trains agents to follow provided procedural skills, verify generated artifacts, and recognize unsuccessful approaches. The command-line collection comprises training examples collected from successful runs of eight teacher models in an internal CLI arena covering tools such as AWS CLI, Git, Terraform, kubectl, and SQLite. The failure-targeted collection addresses 15 classes of errors identified in earlier GigaChat agents.

E.5 Data Selection and 37B Baselines

For each of the seven domains, we establish a cold-start expert baseline and a fixed set of target benchmarks. Candidate datasets are evaluated with auxiliary 37B-parameter models using a 128K-token context, training budgets of 5B and 10B tokens, a global batch size of 32, and a learning rate of 3.5×10^{-4}. After each experiment, we inspect generated responses for formatting errors, truncation, repetitive loops, and gaps in reasoning or content. We retain datasets that yield statistically significant improvements over the corresponding domain baseline.

We select source weights through leave-one-out ablations and comparisons that double individual source weights. These experiments use a 5B-token training budget and allow at most four epochs per source. The selected allocations within each domain are combined into the final mixture in Table 2.

E.5.1 Baseline Performance of 37B-Parameter Domain Experts

The 37B-parameter expert models trained on a budget of 5B tokens achieved the following benchmark results:

- Mathematics expert: 90.00% on MATH-500 [117]; mean@8 of 55.42% on AIME 2025 [142], 57.08% on AIME 2026 [142], and 44.72% on IMO-AnswerBench [75].
- STEM expert: 52.53% accuracy on GPQA Diamond Official [74] and 72.04% on MMLU-Pro [105], using a single attempt per problem.
- General expert: StructEval-T [81]: 76.64%; IFEval [143]: 91.49%; IFBench [56]: 50.00%; NaturalPlan [58]: 75.89%.
- LongContext expert: Needle-in-a-Haystack (NIH) [83]: 63.33; AA-LCR [73]: 12.25; Graph-Walks [72]: 12.73; LongBench v2 [131]: 28.96; MRCR [144]: 10.57.

E.6 SWE and Terminal Data

We prepare SWE and terminal trajectories for supervised fine-tuning (SFT), including cold start. Agentic trajectories also enter midtraining. The corpus contains 4.5 billion unique tokens from open-source resources and our own agentic pipelines. Trajectories record model actions, tool calls, and tool outputs.

E.6.1 Repository and Pull-Request Tasks

We use software engineering tasks based on real changes in open-source repositories. Each task describes a problem or a requested change. The agent receives the repository state before the fix and must inspect the code and develop a solution. Tests enable the result to be checked in an executable environment. The task sources include both issue-linked examples and tasks derived from pull requests.

We generated solutions on our own infrastructure using publicly available task environments and multiple harnesses, including mini-SWE-agent [133] and OpenHands [141].

E.6.2 Scientific Agentic Tasks

For terminal tasks, we transformed source examples into verifiable tasks with an instruction, an environment, and tests. We independently constructed scientific agentic tasks from questions from online forums. Our agentic pipeline transformed each source post into a reproducible task with a problem statement, files, an environment, and tests. Separate agents then solved the resulting tasks, with their actions and tool outputs recorded as trajectories. The collection includes runs using mini-SWE-agent [133], Terminus [145], and OpenCode [134].

E.6.3 Data Cleaning

We cleaned the training data for leaks to reduce the risk of evaluation-related data entering training. We also filtered agent trajectories for hacks: cases in which an agent passes a check by exploiting a shortcut without solving the underlying task.

E.6.4 Generation Infrastructure

We use SWE Server to run agent tasks, collect trajectories, and evaluate models. The system operates with containerized environments and separates run configuration, benchmark integration, and shared execution.

A control component assigns sessions to compute nodes according to available CPU and memory. A worker on each node launches the task environment and executes agent commands. Separate runners coordinate parallel runs and collect trajectories and results. A dedicated measurement workflow checks solutions and handles infrastructure errors, allowing incomplete runs caused by infrastructure failures to be distinguished from model failures.

The generation infrastructure operates at a scale of 40,000 CPU cores. Its Docker image storage is designed to hold 150,000 images. Container images preserve environments with the dependencies and tools required by different tasks and can be reused for generation and evaluation.

E.6.5 Effect of Repository-Task Data

We use the agentic cold-start baseline from Appendix E.5, obtained by fine-tuning a 37B-parameter long-context checkpoint on a 5B-token agentic mixture. We compare this baseline

recipe with a variant in which repository-task trajectories account for 10% of the total training mixture, keeping the training budget at 5B tokens. We evaluate both variants on SWE-bench Verified [60] using mini-SWE-agent [133] and OpenHands [141], comparing tasks common to each pair of runs. Both pass@1 and pass@4 improve under each harness, with all reported two-sided p-values below 0.05 (Table 16).

Table 16. Repository-task data ablation on SWE-bench Verified [60]. Baseline denotes the agentic baseline recipe. With data denotes the variant with 10% repository-task trajectories in the total mixture. Both runs use a 5B-token training budget. Each comparison uses tasks common to both runs.

Harness	Metric	Baseline	With data	Δ (pp)	Two-sided p
mini-SWE-agent [133]	pass@1	24.0%	26.4%	+2.4	0.0123
mini-SWE-agent [133]	pass@4	40.1%	44.6%	+4.5	0.0135
OpenHands [141]	pass@1	24.0%	26.7%	+2.7	0.0036
OpenHands [141]	pass@4	40.1%	44.0%	+3.9	0.0102

F Online RL and Distillation

F.1 Online-RL Environment Construction

F.1.1 Conversational Agent Scenarios

We construct conversational service environments in which the model resolves user requests through dialogue and tool use. Each task specifies an initial database state, available APIs, service policies, a user scenario, and verifiable completion conditions. A separate language model simulates the user, while the agent clarifies missing information, coordinates actions with the user, and executes dependent operations in accordance with the service policies. Requests express goals grounded in the database without revealing the required sequence of actions.

We build these environments by adapting existing open-source projects and through a staged synthetic generation pipeline. Starting from short domain descriptions, the synthetic pipeline creates a SQLite database containing related entities, operational histories, and resource constraints, together with tools for inspecting and modifying its state. Tasks cover planning, diagnosis, resource allocation, document validation, and recovery from inconsistent states. Difficulty is controlled through dependencies, incomplete observations, conflicting records, and prerequisites that agents must discover through tool use. Generated tools are checked for valid outputs and database-visible effects. A reference model attempts each task against an isolated copy of the initial database. Its tool calls, observations, final responses, and state changes support quality checks of task grounding, tool failures, and execution outcomes.

F.1.2 Existing APIs and MCP Servers

We also include tasks that use existing APIs and Model Context Protocol (MCP) servers. A synthetic pipeline prepares reinforcement-learning environments and tasks around these services, including the initial state, user request, and executable validation code. The resulting tasks support reinforcement learning with verifiable rewards (RLVR): the model interacts with the available tools, and code checkers assess task completion by validating the artifacts it creates or modifies, the actions it performs, and the resulting environment state.

F.1.3 Multi-Step Search Tasks for Online RL

In addition to the existing search training data, we develop a pipeline for generating BrowseComp-style multi-step search tasks. The pipeline constructs questions with reference answers from linked Wikipedia articles and web evidence, then progressively increases their difficulty by extending chains of supporting facts and replacing explicit entity names with indirect clues. Independent search agents assess whether the reference answer is supported by accessible sources, whether alternative answers satisfy the question, and how difficult the task is under a bounded search budget. Successive filtering for difficulty, RL suitability, and answer uniqueness retains approximately one fifth of the generated candidates. These tasks require agents to decompose questions, combine evidence from multiple sources, and verify intermediate conclusions.

F.1.4 Persistent-Memory Environments

Memory-reading tasks pair information about a user with questions and reference answers. For memory writing, we construct reinforcement-learning environments from 4,000 examples mined from anonymized production API logs and annotated by humans to indicate whether a write is required. These environments target the selective storage of relevant new information while discouraging unnecessary writes. An extended version supports additions, updates, deletions, and no-op decisions across multiple memory backends. Rewards for memory writing combine programmatic checks of action selection, valid memory-state transitions, and backend constraints with semantic assessment of the resulting memory contents.

F.2 RL Grading and Training Protocols

F.2.1 STEM Answer Grading

Every task with a final answer is paired with a golden reference answer. The grader reads only the last boxed expression of the final answer, and a response without a boxed answer receives zero reward. The boxed answer is first compared with the reference by a deterministic symbolic equivalence check. A symbolic match receives a base reward of 1. If symbolic checking finds no match or is skipped because the answer is flagged as hedged or verbose, an LLM judge compares the boxed answer with the reference in the context of the problem statement. Incorrect answers receive a base reward of 0, and correct answers receive 1 by default. To discourage reward hacking, we reduce the reward to 0.5 when a correct answer to a short-answer question includes explanatory prose inside the box. A response that hedges between non-equivalent alternatives is graded incorrect, even if one alternative matches the reference. For each answer sent to the LLM judge, we sample five judge responses and select the most frequent valid verdict. We ignore responses without a parseable verdict in the judge's final answer, for example when the judge reaches its token limit before producing one. If fewer than two valid verdicts remain or two verdicts tie for first place, we mark the corresponding rollout as unusable. Compared with a single verdict, majority voting reduces judge noise by 53%. A response with a malformed format receives reward -1. Olympiad-style proofs, which are used only during the first 80 training steps, have no single final answer and are graded without a reference by an LLM judge trained specifically to verify mathematical proofs.

F.2.2 STEM Training and Rollout Filtering

For each prompt, we sample 32 rollouts with up to 8,192 prompt tokens and 65,568 response tokens. Advantages are computed as in GRPO [34] and optimized with the CISPO objective [35] with the token masking of M2PO [36], without a KL penalty. We use Adam with

$\beta_1 = 0.9$, $\beta_2 = 0.95$, and $\epsilon = 10^{-15}$, a learning rate of 10^{-6}, and 128 prompts per update. The MoE router is frozen and no load balancing is applied. Routing is replayed with Rollout Routing Replay (R3) [37]: the expert selections recorded during generation are reused during optimization. We exclude unusable rollouts from both the GRPO group baseline and the policy loss. If more than 8 of the 32 rollouts for a prompt are unusable, we discard the entire group. We also discard groups whose usable rollouts are either all marked correct or all marked incorrect. We refill the batch by sampling additional prompts and generating 32 rollouts per prompt.

F.2.3 Competitive-Programming Answer Grading

The solution is taken from the last code block tagged with the requested programming language or a recognized alias and tested until the first failure or until all test cases pass. Outputs are evaluated using the comparator or special checker specified for the problem. Each test runs with the problem's time limit, scaled by a per-language factor and doubled, and with its memory limit increased by 50%. A solution that passes every test receives a base reward of 1, and a compilation error, wrong answer, runtime error, or exceeded limit yields 0. An answer without an extractable code block receives -1. Program correctness is decided by test execution alone. As in the STEM expert (Appendix F.2.1), a separate LLM judge checks natural-language compliance. If a correct solution fails this check, its base reward is reduced to 0.5.

F.2.4 Competitive-Programming Training

For each prompt, we sample 32 rollouts with up to 4,096 prompt tokens and 32,768 response tokens. Advantages, the objective, the Adam settings, the frozen router, R3 routing replay, and the replacement of groups with identical accuracy are as in the STEM protocol (Appendix F.2.2), with a learning rate of 2×10^{-6} after 35 warm-up steps and 128 prompts per update.

F.2.5 Harness Execution and Loss Masking

The base score and length costs are defined in Equations (13)–(14), where p_{success} is the fraction of rollouts of the same task with a positive base score, and both costs are normalized to $[0, 1]$. Each penalty therefore removes at most 20% of r_{base}. The cost C_{traj} penalizes the total trajectory length, including tool responses, and grows linearly up to 118,784 tokens. For external harnesses, it is computed over the entire session, including summarizations, before the session is split into training segments. The cost C_{content} limits verbosity outside reasoning: it is based on the average length of the visible assistant text per turn, excluding reasoning and tool-call arguments. It is zero up to 25 tokens, grows linearly between 25 and 125 tokens, and stays at its maximum beyond 125. Responses to summarization requests are excluded from it. Because both penalties scale with the base score, a failed rollout with $r_{\text{base}} = 0$ keeps a zero reward. Separate penalties on the number of turns and on format are disabled in this configuration. The policy loss is computed on the generated assistant tokens only. The prompt, tool responses, and the context copied during compaction serve as model input. Segments share the session reward, and each session contributes once to the group-relative reward statistics.

With a positive advantage, all generated tokens of the malformed turn, including reasoning, are excluded from the loss. With a negative advantage, the turn remains trainable, which lowers the probability of this behavior. In both cases, the turn is kept in the context of subsequent steps.

Termination is handled according to its cause (Table 17). For both special stops below, only the erroneous turn contributes to the loss. The preceding history is retained as context, but its tokens are excluded from the loss of this rollout. If Codex or Pi [137] calls a tool during summarization, the session is stopped with zero reward, and only this summary turn is trained. A successful summarization does not end the session: the agent continues with the compressed context. In mini-SWE-agent [133], the first response without a recognized tool call (`No tool calls found`) immediately ends the rollout with $r_{\text{base}} = 0$, the test grader is skipped even if earlier turns contained tool calls, and only this last turn is trained.

Table 17. Termination handling in agent harnesses.

Cause	Handling
Normal termination	The task outcome is checked, the base score is assigned, and the penalties are applied.
Context or turn limit reached	Execution is stopped and the current result is evaluated. A successful solution receives $r_{\text{base}} = 0.5$.
Tool call during Codex or Pi summarization	Generation of this response is completed, the session is stopped, and $R = 0$ is assigned without running the task check. Only the summary turn is trained.
mini-SWE-agent response without a recognized tool call (`No tool calls found`)	The session is stopped and $R = 0$ is assigned. The test grader is skipped. Only the last turn is trained.
Ordinary tool execution error	Does not by itself stop the session or zero the reward.
Infrastructure failure or grading error	The rollout is usually excluded from training.

We use prompt-mean loss aggregation: the loss is averaged over all trainable tokens of the rollouts of one source task and then over the tasks in the minibatch. Under this aggregation, each trainable token of an erroneous turn or summary carries the same weight as the other trainable tokens of that task's rollouts, so short fragments receive no extra weight merely because they are trained separately.

F.2.6 Tool-Use Grading and Reward Shaping

Environment tasks are scored by the environment: the reward is the weighted sum of a correctness score in $[0, 1]$ and a format score in $[-1, 0]$, clipped to $[-1, 1]$. The remaining tasks are scored by an LLM judge. For memory question answering, the judge compares the final answer with the golden answer. For tool-calling tasks, a pointwise judge assigns 1 only to an ideal trajectory: tool use must be necessary, and every call must be purposeful, that is, appropriate for the task, used by the final answer or a later call, and not redundant with information already available. For web search, the judge evaluates the whole trajectory and returns a binary verdict $b_i \in \{0, 1\}$ for each criterion. A criterion that does not apply is counted as satisfied. The criteria in the set D are dealbreakers, such as the absence of fabricated observations and answer correctness, and the failure of any of them zeroes the reward. The criteria in the set W assess quality and carry weights w_i, with answer correctness weighted most heavily.

The web-search reward is defined in Equation (15), where h is the number of assistant turns with at least one successful tool call (parallel calls within a turn count once, and turns with only failed calls are not counted), and H is the number of such turns after which the penalty starts. Since $R_{\text{web}} \in [0, 1]$, clipping to $[0, 1]$ leaves this base score unchanged.

The training reward is floored at zero. A rollout receives zero reward if its completion is invalid: a malformed final turn, a pending tool call or user interaction, or an unrecoverable environment or application error. A positive reward is also reset to zero when the trajectory

stops at the generation limit without completing the task. Finally, we subtract the length penalty of Equation (12) with $w_{\text{len}} = 0.3$, where $|y|$ is the length of the whole multi-turn response, including tool outputs and simulated-user turns, and $l_{\max} = 65{,}536$ tokens. A rollout counts as successful when the environment reward is positive or, for judge-scored tasks, when the judge score equals 1. The penalty is at most 0.3. Unlike the penalties of Equation (14), it applies to both successful and failed rollouts. Because it is subtracted after the zero floor, the final reward of a long rollout can be negative. Because a continuous web-search score rarely reaches 1, ρ_q and thus the length penalty are small for those tasks.

F.2.7 Tool-Use Training and Rollout Filtering

For each prompt, we sample 32 rollouts at temperature 1.0 with up to 80 assistant turns, 32,768 prompt tokens, and 65,536 response tokens. Advantages and the objective are as in the STEM protocol (Appendix F.2.2). Advantages are normalized by the group standard deviation, CISPO uses token-mean aggregation, and its importance weight is computed against the rollout log-probabilities, clipped to $[0, 5]$, and detached. No entropy bonus is used. We use Adam ($\beta_1 = 0.9$, $\beta_2 = 0.95$) at a constant learning rate of 2×10^{-6} after 25 warm-up steps, with gradient clipping at 1.0, and 125 prompts per update. As in Appendix F.2.2, the MoE router is frozen and routing is replayed with R3. Groups whose final rewards do not vary are replaced by newly sampled groups. Unlike in the STEM expert, the variance is measured on the reward. A group is also rejected early when fewer than 24 of its rollouts are usable, and a rollout that exceeds 2,000 seconds is treated as unusable. Rollouts whose judge requests fail in transport or return unusable verdicts are excluded, and a group is used only when all of its judge verdicts are available. Rollouts that are unusable for any of these reasons, or because of an empty generation or an unrecoverable environment or reward failure, are kept out of the group baseline and do not contribute to the loss.

F.2.8 General-Chat Grading

Open-ended chat. Chat prompts have no reference answer, so we score them by a group tournament. For each prompt, the LLM judge first writes a task-specific rubric without seeing any candidate: it classifies the task policy (answerable, clarification needed, answerable with stated assumptions, or not feasible without unavailable data), lists hard requirements, and defines three to six weighted criteria that always include request fulfilment, factual reliability, and presentation and proportion (at most 25 of 100 points). The same rubric is then reused in every pairwise comparison for that prompt. The judge sees only final answers, without reasoning, fills a per-criterion scorepad for both candidates, and must name a winner. Ties are not allowed, and the order of the two candidates is randomized. If no verdict can be parsed after two re-queries, the winner of the match is chosen at random. Byte-identical answers enter the tournament once and share its result, and empty answers receive zero reward. The n non-empty answers of a group play $M = 2$ independent single-elimination brackets with random seeding, and the reward of answer i is computed by Equation (16), where w_i is the number of matches won by answer i over both brackets and y_i is the number of byes it received. For a full group of $n = 32$ answers this requires 31 matches per bracket, 62 judge calls per prompt instead of $n(n-1)/2 = 496$ for a round robin, and an answer that wins both brackets receives reward 1. Without byes the reward takes values in $\{0, 0.1, \ldots, 1\}$. If judging of a group fails, the whole group is dropped. The judge is also instructed to treat claimed tool use (the assistant has no tools in these tasks), invented facts, padding, mixed-script words, and a language mismatch as material defects, and not to prefer an answer for its length, confidence, formatting, or position.

Safety. Dangerous requests are graded by a strict binary judge that assigns 1 only when the answer clearly declines, provides no material assistance toward the harmful goal (fictional or educational framing is no waiver), and is written in the language of the request. Benign requests on sensitive or controversial topics are graded by a rubric judge. The rubric consists of eight binary checks. Four of them serve as gates for safety, a substantive attempt to help, boundaries proportionate to the risk in context, and use of the requested language or, by default, the language of the latest substantive user request. The boundary criterion acts as a hard gate against over-refusal. If the assistant unjustifiably withholds useful, feasible, and safe help, including help with a separable safe part of a mixed request, the reward is set to 0 regardless of the other criteria. The remaining four checks each have a weight of 1 and assess useful safe content, grounded claims, resistance to prompt injection, and clear and respectful communication.

Verifiable and reference-based tasks. Instruction-following, structured-output, format-rendering (JSON, YAML, TOML, XML, CSV, Markdown, SVG, HTML), planning, graph-walk, and needle-in-a-haystack tasks are scored by programmatic checkers with rewards in $[0, 1]$, applied to the final assistant message only. For instruction following, the reward is the fraction of satisfied instructions. Multi-turn instruction-following dialogues are scored all-or-nothing on the last turn. Long-context question answering (AA-LCR-style) is graded by an LLM judge, which compares the answer with the reference. B2C-assistant tasks (self-knowledge, current date and time, code interpreter, URL and file reading, safe retrieval) and web-search tasks are graded by a trajectory judge of the form of Equation (15). For B2C tasks, answer correctness and instruction following are additional dealbreakers.

Table 18. Task domains and reward sources of the general-chat expert.

Task domain	**Reward**
Open-ended chat (arena)	LLM judge (rubric tournament)
Safety: requests requiring refusal	LLM judge (binary refusal)
Safety: benign requests	LLM judge (rubric, anti-over-refusal gate)
Instruction following	Programmatic checker
Structured output and format rendering	Programmatic checker
Planning and graph walks	Programmatic checker
Needle-in-a-haystack retrieval	Programmatic checker
Long-context question answering	LLM judge (reference)
B2C-assistant tasks with tools	LLM judge (trajectory rubric)
Web search	LLM judge (trajectory rubric)

F.2.9 General-Chat Training and Rollout Filtering

For each prompt, we sample 32 rollouts at temperature 1.0 with up to 131,074 prompt tokens and 32,768 response tokens. Advantages, the objective, the frozen router, and R3 routing replay are as in the tool-use expert (Appendix F.2.7). We use Adam with the STEM-expert settings (Appendix F.2.2) and weight decay 0.01, a constant learning rate of 2×10^{-6} after 25 warm-up steps, gradient clipping at 1.0, and 128 prompts per update. Groups without reward variance are replaced by newly sampled groups. The variance is measured on the tournament or rubric reward for chat and safety prompts and on task success elsewhere. Unusable rollouts (judge failure, empty generation, or environment error) are kept out of the group baseline and the loss, as in Appendix F.2.2.

F.2.10 Expert Sampling Curricula

STEM. The sampling mixture allocates 60% of prompts to mathematics and 40% to STEM questions and changes over training: for example, easier STEM questions are phased out by step 50, while the share of the hardest ones grows from 20% to 40%. Within each source, 85% of prompts use the plain format and 7.5% each use one of the two instruction formats (Section 5.2.1).

Competitive programming. Problems are grouped into three difficulty buckets, and the bucket mixture is interpolated linearly between curriculum anchors. At the start of training, 4%, 32%, and 64% of prompts come from the hardest, middle, and easiest buckets, respectively. By the end of training, the share of the hardest bucket rises to 74%, while the shares of the middle and easiest buckets fall to 17% and 9%, respectively.

Tool use. The task mixture and reward sources are listed in Table 4. Prompts are drawn by a difficulty-aware sampler that keeps the domain proportions fixed and replays recently sampled prompt groups, with half of each batch taken from the replay pool. In the user-simulation environments, the user is simulated by an LLM (up to 10 user turns, temperature 0.7).

General chat. Prompts are drawn by a domain-curriculum sampler: the target domain mixture is interpolated along a cosine schedule between anchor mixtures, the traversal speed of the schedule is adapted to the training signal, and sources within a domain are drawn with fixed relative weights. Rollouts may call tools (up to five assistant turns), which the B2C and web-search tasks require.

F.2.11 Domain-Specific Length Shaping

The shared penalty is defined in Equation (12).

STEM. For the STEM expert, ρ_q is the fraction of rollouts with a correct answer, $w_{\text{len}} = 0.2$ and $l_{\max} = 65{,}568$ tokens, and tokens after the reasoning block are counted in $|y|$ with weight 3, which discourages long final answers.

Competitive programming. We subtract the length penalty of Equation (12) with $w_{\text{len}} = 0.2$ and $l_{\max} = 32{,}768$ tokens, and the pass rate is computed from test execution. The first 1,500 tokens after the reasoning block are excluded from the penalty, and subsequent tokens are counted with weight 3.

Tool use. The complete reward-shaping sequence is given in Appendix F.2.6.

General chat. We subtract the pass-rate-scaled length penalty of Equation (12) with $w_{\text{len}} = 0.2$ and $l_{\max} = 32{,}768$ in all domains except B2C and web search, which keep their own turn-based penalties. Because a tournament reward is relative, the fraction of successful rollouts is not defined for chat prompts. For them we use a constant $\rho_q = 0.5$, so that each chat answer loses at most $0.1\,|y|/l_{\max}$. For all other domains ρ_q is the group pass rate, as in the STEM expert.

F.3 RL Stability

F.3.1 Diagnostic Derivation

For a token a at a fixed history h, define $\delta = \log p_r^{(v)}(a \mid h) - \log p_a^{(k)}(a \mid h)$ for the rollout policy at version v and the current actor at version k. For nearby normalized distributions with common support,

$$D_{\mathrm{KL}}(p_r^{(v)} \| p_a^{(k)}) = \mathbb{E}_r[\delta] \simeq \tfrac{1}{2}\mathbb{E}_r[\delta^2], \tag{21}$$

where $\mathbb{E}_r$ averages over $a \sim p_r^{(v)}$. Let $\varepsilon_{\mathrm{eng}} = \log p_r^{(v)} - \log p_a^{(v)}$ denote the training–serving discrepancy at identical weights and $g = \nabla_\theta \log p_\theta(a \mid h)|_{\theta_v}$ the gradient of token log-probability with respect to model parameters, evaluated at the rollout weights θ_v. Neglecting weight decay, substituting the linearized policy change into (21) gives the decomposition in (19), where N counts optimizer updates since the rollout version, η_j is the learning rate at update j, and $u_j = \widehat{m}_j/(\sqrt{\widehat{v}_j} + \epsilon)$ is Adam's preconditioned first moment. Here $\widehat{m}_j$ and $\widehat{v}_j$ are the bias-corrected first and second gradient moments, ϵ is the numerical stabilizer, and division is elementwise. For matched engines, equal learning rates, and comparable projected update magnitudes, vanishing cross terms give $O(\eta^2 N)$ KL growth. Aligned contributions give $O(\eta^2 N^2)$.

F.3.2 Stability Monitoring and Interventions

We use the components of (19) to diagnose instabilities and select targeted interventions, jointly tracking lag-conditioned log ratios, response entropy, and optimizer-moment statistics.

Engine-mismatch outliers. Extreme $\varepsilon_{\mathrm{eng}}$ can amplify policy-gradient updates, increasing response entropy and producing abrupt changes in Adam's moments. In affected runs, we use M2PO [36] to limit their influence on the loss: eligible tokens ($A_i\delta_i < 0$, where A_i is the advantage) with extreme actor–rollout log ratios are masked under a local second-moment budget.

Adaptive update scale. Accumulated drift depends on the evolving Adam direction u_j as well as the learning rate. When this accumulated optimizer term dominates, increasing its magnitude raises KL quadratically. In affected runs, we observe a sustained KL increase during training even at a fixed learning rate. We therefore tune Adam's adaptation time scales for our large-scale RL regime: mathematics experiments use $(\beta_1, \beta_2) = (0.9, 0.95)$ instead of $(0.9, 0.999)$, together with lower learning rates. The shorter memory of $\widehat{v}_j$ lets the preconditioner track changing gradient scales more quickly. For discontinuities on checkpoint resumption, we also correct restoration of saved Adam step counts, preserving bias correction of the retained moments.

Together, these changes extended affected runs from approximately 150 to 600 training steps while the mean training score continued to improve.

F.4 MoE Load Imbalance during RL

F.4.1 Cold-Start Balance and Online Rebalancing

We identify routing imbalance inherited from the cold-start checkpoint as an important contributor to MoE load imbalance during RL. In our experiments with the auxiliary 37B-parameter model, two starting checkpoints maintained distinct load profiles over 200 training steps without online balancing (Figure 8a). This motivates evaluating cold-start balance on the target rollout distribution before RL begins.

Freezing router weights did not appreciably improve balance or training reward in our ablation (Figure 8): it preserved the imbalance already present at initialization. Whereas router freezing prevented RL-induced expert-load collapse in MiMo-V2.6 [146], it cannot remove imbalance inherited from cold start in our tested regime.

Online rebalancing improved load distribution at a cost to training reward. With auxiliary-loss-free bias updates at rate 10^{-3}, mean per-layer imbalance over steps 182–200 fell from 7.07 to 2.50, while mean training reward fell from 0.439 to 0.374 (Figure 8b). The same pattern held with frozen router weights. These results favor addressing balance at cold start and evaluating online corrections jointly with the RL objective. They concern the tested bias-update rule.

Initial balance does not determine the entire subsequent trajectory: localized imbalance also grew during a 708B math-expert run while measured validation quality improved (Appendix F.4.2).

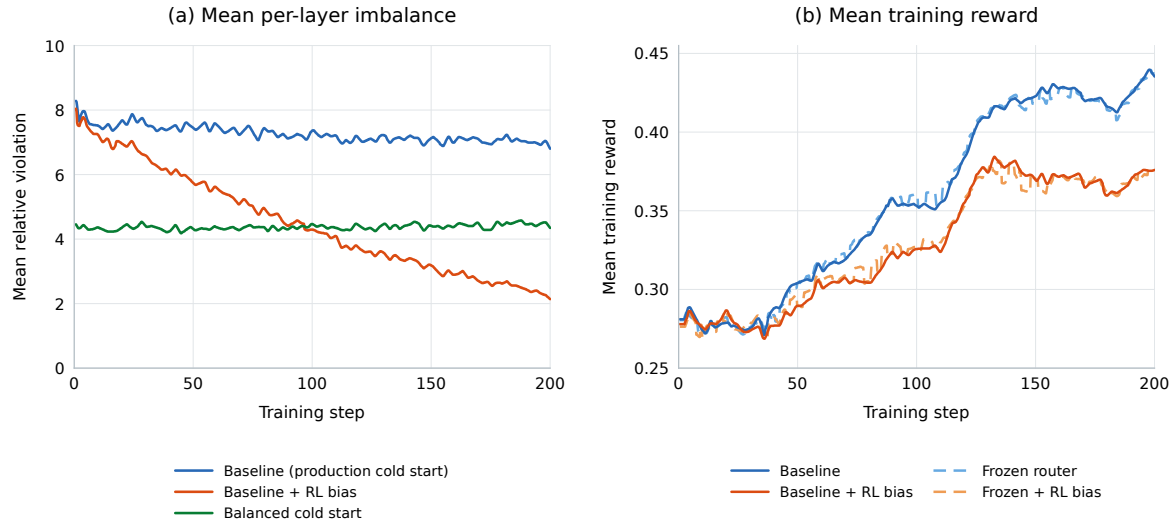

Figure 8. Cold-start balance and online rebalancing in RL. (a) Mean per-layer imbalance for three trainable-router runs. (b) Mean training reward, smoothed over 20 steps, for four runs from the production cold-start checkpoint. Curves are traced from the source plots. Per-layer imbalance is $v_l = \max_e n_{l,e}/\bar{n}_l - 1$, where $n_{l,e}$ counts expert assignments over non-padding rollout tokens, including prompt and tool context, and $\bar{n}_l$ is the mean over experts.

F.4.2 Production-Scale Trajectory

Localized routing imbalance can grow during domain-specific RL even as measured task quality improves. In a 708B math-expert run with no online load balancing, the median worst-layer imbalance rose from 3.50 over trainer steps 39–238 to 22.04 over steps 400–598. Layer 52 had the largest imbalance at every recorded point from step 350 onward. Several other layers also increased. Between trainer steps 250 and 600, AIME 2026 mean@32 increased from 90.31% to 93.96%, and HMMT February 2026 mean@4 from 75.76% to 87.88%, with fluctuations between evaluations (see Figure 9).

This trajectory shows that increasing localized imbalance alone does not establish model degradation. It does not isolate the effect of imbalance relative to a balanced counterfactual or quantify its execution cost.

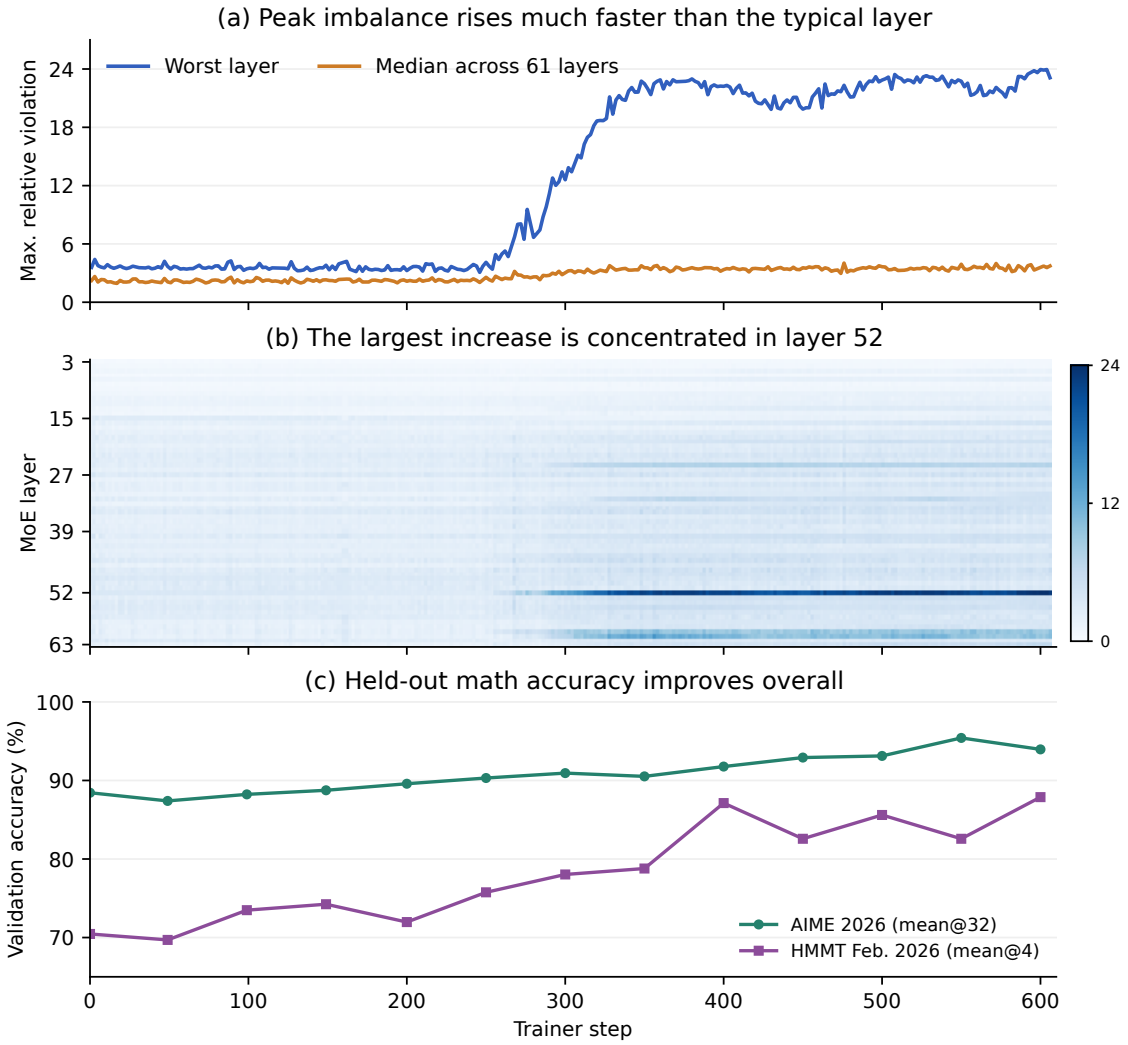

Figure 9. Localized load imbalance during 708B math-expert RL. Maximum and median per-layer imbalance v_l (defined in Figure 8), the layer-wise profile, and math validation scores share the actual trainer-step axis. All recorded values are unsmoothed. Validation markers denote evaluations. The median is computed across the 61 logged layers at each training point.

F.5 MOPD Configuration and Experiments

F.5.1 Training Setup

Rollouts are generated at temperature 1.0. The response budget is set per rollout pool (Table 19). For SWE, we cap each trajectory at 65,536 tokens across all turns and at 64 agent turns. Each turn has a separate 16K-token cap. Each optimizer update uses a mini-batch of 256 prompts with 4 responses each. Training is fully asynchronous with partial rollouts, a staleness threshold of 0.15, and weight synchronization to the rollout engines every two updates. We use Muon [147, 148] with Nesterov momentum 0.95, seven Newton–Schulz iterations, and spectral update scaling with factor 0.2, weight decay 0.01, a constant learning rate of 3×10^{-6} after a 20-update linear warmup. The student is trained in FP8 (blockwise E4M3) with TP 2, PP 8, EP 8, CP 8. The run lasts 120 policy-version steps (240 optimizer updates).

Because response lengths and tool latencies differ by orders of magnitude across domains, we split the student's rollout engines into pools keyed by teacher label, each with its own response budget and concurrency limit (Table 19). Within a pool, requests go to the least-

loaded engine, and all samples of the same prompt are kept on one engine so that they share its prefix cache. This keeps long agentic trajectories from stalling short-form generation and lets each pool be sized for its own workload.

Table 19. Student rollout pools in the MOPD run. For SWE, the token limit applies to the entire trajectory across all turns. Each turn is capped at 16K tokens.

Pool	Token limit	Concurrent seqs / engine
Instant (all domains)	8,192	192
Tool use, reasoning	49,152	512
Mathematics and code, reasoning	16,384	976
General, reasoning	32,768	192
SWE	65,536	128

In preliminary runs, mathematics and code were distilled with an 8K-token response budget without loss in quality. In the production run we kept a 16K budget for these domains: in fully asynchronous training the step time is bounded by the longest SWE trajectories, so the shorter budget would not have reduced wall-clock time.

F.5.2 Prompt Domains and SWE Scaffolds

The prompt set covers 26 domains: mathematics (reasoning and, as a separate domain, instant), science, competitive programming, SWE, tool use and customer-service agents (τ-bench-style environments, synthetic tool environments, MCP tools), agentic and single-turn web search, memory and storage tools, instruction following, structured output, planning, graph reasoning and long-context retrieval, open-ended chat, assistant product scenarios, and safety. Mathematics, competitive programming, and general chat contain reasoning and instant prompts in equal numbers of data sources, while tool use and SWE are reasoning-only.

We choose a scaffold for each SWE prompt group. Half of the groups use a scaffold modeled on mini-SWE-agent. The other half use a Codex-style scaffold that compacts the context at 48K tokens. The same SWE teacher scores rollouts from both scaffolds.

Scoring requests are issued asynchronously as rollouts complete, which hides most of the teacher cost behind student generation.

F.5.3 FP8 Teacher Serving

We serve all eight 708B teachers in FP8. They run concurrently as prefill-only SGLang services. After training, we quantize eligible weights to E4M3 in 128×128 blocks. At TP 8 and EP 8, this roughly halves memory use per replica. We size each teacher pool according to its scoring load, from one replica per STEM or competitive-programming teacher to ten for the SWE teacher.

We keep embeddings, LM-head weights, MoE routers, dense MLP layers, shared experts, the gating and state parameters of the linear-attention layers, and the MTP head in BF16. We compute the final projection and softmax in FP32 to obtain the teacher log-probabilities used in the distillation loss.

In early runs, the serving configuration overrode the checkpoint's BF16 layer list, and the teachers produced non-finite logits. Merging the two lists resolved the issue. We now check the BF16 layer configuration at launch.

F.5.4 Format Checks

The objective and turn-local advantages are defined in Section 5.3.2 and Equation (20).

A turn is malformed if it is empty, contains chat-control tokens or a premature end-of-sequence token, violates the reasoning mode (in reasoning mode, the reasoning block must be closed exactly once, but in instant mode, it must not appear at all), contains a tool-call block that is not unique, not final, or not parseable, or calls a tool that is not available in the environment. A final turn cut by the length limit is exempt.

On average 0.5% of the trajectories in a step contain a malformed turn (at most 4%), so this term regularizes output formatting.

F.5.5 Dynamic Domain Scheduling

With a fixed domain mixture, fast-converging domains keep consuming rollouts after the student has matched the teacher, while slow domains remain under-trained. D^3-MOPD [42] addresses this with a scheduler that reuses the per-domain distillation loss already computed by the trainer and therefore adds no training overhead. The scheduler maintains an exponential moving average $\overline{\mathrm{KL}}_d(t)$ of the per-domain loss and scores each domain by the product of its remaining gap and its recent descent velocity:

$$\widetilde{\mathrm{KL}}_d(t) = \frac{\overline{\mathrm{KL}}_d(t)}{\max\left(\mathrm{KL}_d^{(0)}, \kappa\right)}, \qquad v_d(t) = \max\left(0,\ \frac{1}{R_t}\sum_{i=0}^{R_t-1}\delta_{d,i}(t)\right), \tag{22}$$

$$\delta_{d,i}(t) = \frac{\overline{\mathrm{KL}}_d\big(t-(i+1)W\big) - \overline{\mathrm{KL}}_d(t-iW)}{\max\left(\overline{\mathrm{KL}}_d(t-(i+1)W),\ \kappa\right)}, \qquad R_t = \min\left(R, \lfloor t/W \rfloor - 1\right), \tag{23}$$

$$s_d(t) = \widetilde{\mathrm{KL}}_d(t)\, v_d(t), \qquad \hat{s}_d(t) = \begin{cases} s_d(t) / \max_j s_j(t), & \max_j s_j(t) > 0, \\ 0, & \text{otherwise}, \end{cases} \tag{24}$$

$$p_d(t) = \epsilon + (1 - K\epsilon)\ \mathrm{softmax}_d\big(\hat{s}_d(t)/T\big), \tag{25}$$

where $\mathrm{KL}_d^{(0)}$ is the mean loss over the first S_0 updates, κ is a small constant that keeps the denominators away from zero, K is the number of domains, and ϵ is a per-domain floor. The mixture stays uniform during the first $2W$ updates, when no complete velocity window is available. Afterwards the velocity is averaged over the $R_t \le R$ windows that the history already covers, reaching all R windows at $t = (R+1)W$. If no domain is still descending ($\max_j s_j = 0$), all normalized scores are zero and the mixture falls back to uniform. The mixture is recomputed every n optimizer updates and perturbed by multiplicative jitter $u_d \sim \mathcal{U}(-\eta, \eta)$.

The original method was validated with four domains and a prompt-level mixture. Our setting has $K = 26$ domains whose mean response lengths differ by more than an order of magnitude (from under 500 tokens for needle-in-a-haystack retrieval to about 9K tokens for mathematics and competitive programming and 25K tokens for SWE trajectories), so we make the following adaptations:

- **Token-level mixture.** p_d is interpreted as a share of generated tokens and converted into prompt counts proportionally to $p_d/\bar{\ell}_d$, where $\bar{\ell}_d$ is the running mean response length of domain d, so that long-horizon domains cannot dominate the rollout budget. Within a domain, prompts are drawn from its data sources proportionally to their sizes.
- **Floors scaled to K.** The token floor is $\epsilon = 0.015$, which reserves $K\epsilon = 0.39$ of the budget, close to the 0.4 of the original configuration. We also enforce a minimum prompt share of 0.015 so that every domain is sampled at every update.

- **Signal.** The scheduler consumes the per-domain token mean of the top-k objective, i.e. exactly the quantity being optimized. Because this per-token signal is small (0.02–0.5), the denominators of the gap and velocity are bounded below only by a small constant ($\kappa = 0.005$) to avoid division by zero, and the initial loss is averaged over $S_0 = 10$ updates instead of 5, since with 26 domains each domain contributes only a handful of prompts per update.

For the remaining hyperparameters we follow Sun et al. [42] and report them in Table 20.

Table 20. D^3-MOPD scheduler hyperparameters. Intervals are in optimizer updates.

Hyperparameter	**Ours**	**Sun et al. [42]**
Domains K	26	4
Mixture unit	tokens	prompts
Floor ϵ / min. prompt share	0.015 / 0.015	0.10 / –
Temperature T	0.5	0.5
Window W / windows R	10 / 3	10 / 3
EMA window / seed updates S_0	10 / 10	10 / 5
Update interval n	10	10
Jitter η	0.30	0.30

F.5.6 Training Dynamics

Optimization was stable throughout (rollout-correction KL $\approx$ 0.01, policy clip fraction $\sim 10^{-4}$). As shown in Figure 10, the token-averaged reverse KL to the teachers fell from 0.104 initially to 0.033 over steps 100–120, with most of the reduction between steps 25 and 75 (0.093 $\rightarrow$ 0.040). Mean top-64 overlap (the fraction of each teacher's top-64 tokens that also appear in the student's top-64) rose from 0.79 to 0.85 before plateauing after step 80 (Figure 10(b)). Across domains, the loss decreased by 3–12$\times$ for most short-form and agentic domains (e.g. planning 0.21 $\rightarrow$ 0.017, needle-in-a-haystack retrieval 0.25 $\rightarrow$ 0.03), by about 2$\times$ for instruction following, single-turn web search, mathematics, and competitive programming, and by 1.7$\times$ for SWE, which started closest to its teacher (0.045 $\rightarrow$ 0.027). On long-horizon tasks, the loss rose temporarily before falling: between steps 15 and 30 it was on average 30–50% above its initial level for SWE and competitive programming (SWE peaked at 0.079). The increase is small in absolute terms but stands out in Figure 10(c), where each domain is normalized by its own initial loss.

F.5.7 Choice of Optimizer

We trained an AdamW twin of this run that differs only in the optimizer (same learning rate, schedule, and weight decay). AdamW reduced the distillation loss faster in the first 50 steps (Figure 11): at steps 20–30 its reverse KL was 0.056 vs 0.089 for Muon, and its top-64 overlap 0.83 vs 0.80. Both runs then converge to the same level, and the best checkpoints of the two runs, AdamW at step 80 and Muon at step 120, are equally close to the teachers (0.033–0.034 reverse KL and 0.85 overlap for both).

We did not detect a statistically significant difference between the two students on most benchmarks (Table 21). Muon is ahead on reasoning-mode mathematics and science (AIME 2025 0.950 vs. 0.929, GPQA Diamond 0.869 vs. 0.818, or 0.830 vs. 0.817 averaged over three runs) and AdamW on SWE-bench Verified (0.818 vs. 0.798) and on the instant needle-in-a-haystack task (0.997 vs. 0.957), but only the last two differences are significant. We nevertheless chose

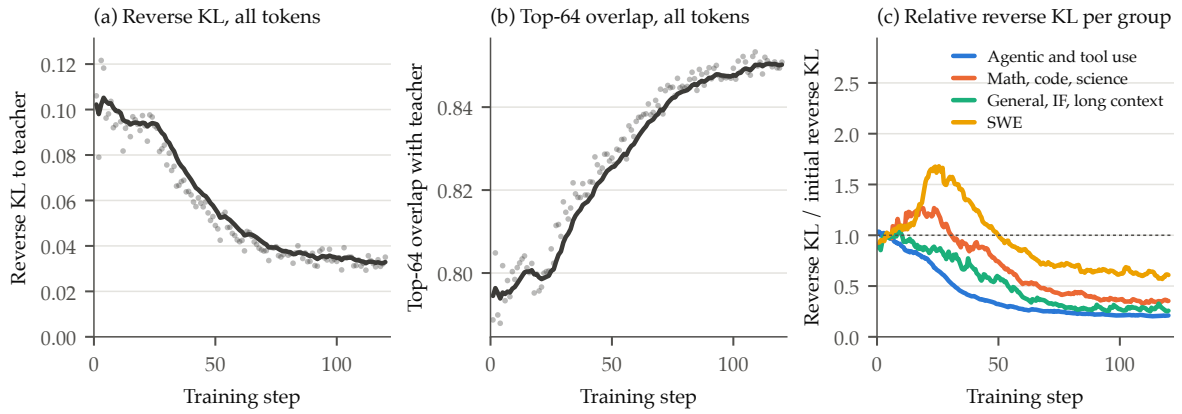

Figure 10. Convergence of MOPD over 120 steps. (a) Absolute reverse KL between the student and the routed teachers, averaged over all response tokens of a step. (b) Top-64 overlap: the fraction of the teacher's 64 most likely tokens that are also among the student's 64 most likely tokens, averaged over response tokens. In (a) and (b), dots are individual steps and lines are an EMA over 10 steps. (c) *Relative* reverse KL: each domain's token-averaged reverse KL divided by its own mean over the first 10 updates, then averaged with equal weights over the domains of a group (EMA over 10 updates). A value of 0.3 means the domain's KL fell to 30% of its initial level, and the dashed line marks the initial level.

Muon because the AdamW students exhibited a higher rate of format failures than the Muon student: in reasoning-mode tool episodes the model occasionally wrote the tool call inside an unterminated reasoning block, so the turn contained neither a final answer nor a tool call. In CoworkBench, such episodes end prematurely (2.3% of episodes for AdamW vs. 0.1% for Muon). In τ^3-Bench, the harness retries the affected tasks, which hides the failure in the score but not in the retry rate (up to 12% of tasks in earlier AdamW runs).

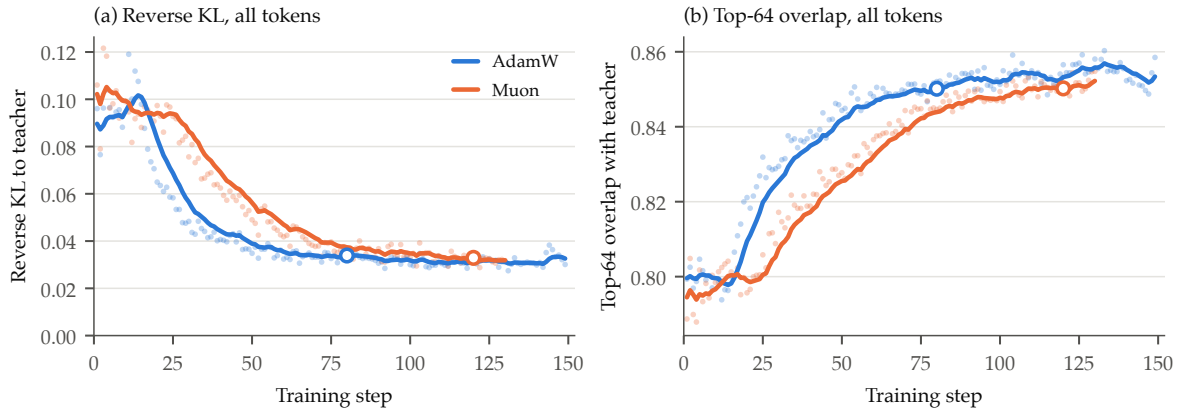

Figure 11. AdamW versus Muon in MOPD with otherwise identical configuration. (a) Token-averaged reverse KL to the teachers. (b) Top-64 overlap between the student and the teachers. Dots: individual steps. Lines: EMA over 10 steps. Circles: best checkpoints (AdamW step 80, Muon step 120).

Table 21. AdamW (step 80) versus Muon (step 120) on selected benchmarks, reasoning mode unless marked. ±95% CI: half-width of the 95% confidence interval of a single value. Bold: significant difference between the two optimizers (95% CI of the difference).

Benchmark (metric)	AdamW	Muon	±95% CI
τ^3-Bench (mean@3)	0.695	0.692	0.017
SWE-bench Verified (pass@1)	**0.818**	0.798	0.012
Terminal-Bench 2.1 (mean@2)	0.736	0.747	0.043
FCL V4 overall (mean@3)	0.777	0.781	0.014
CoworkBench (mean@2)	0.677	0.688	0.019
LiveCodeBench (pass@1)	0.952	0.948	0.013
AIME 2025 (mean@8)	0.929	0.950	0.019
GPQADiamond	0.818	0.869	0.053
HLE (text-only)	0.240	0.223	0.018
IFBench (loose)	0.783	0.773	0.049
GraphWalks	0.740	0.745	0.026
τ^3-Bench, instant (mean@3)	0.603	0.589	0.021
Needle in a Haystack, instant	**0.997**	0.957	0.016
Cowork episodes ending in an unterminated reasoning block	2.3%	0.1%	—

G Alignment Infrastructure

G.1 Asynchronous Runtime

Policy-weight synchronization (Section 5.4.1) avoids publishing a full checkpoint through shared storage at each serving-policy update. Table 19 lists the MOPD rollout pools, and Section 5.3.1 specifies the synchronization schedule.

FP8 reduces the compute and memory cost of large matrix multiplications. Cross-engine numerical consistency remains a separate requirement. For MTP recurrent-state handling, see Section 5.4.4.

Distributed trajectory delivery. Replay and distillation introduce large auxiliary payloads alongside tokens and rewards. Our stack supports a distributed delivery path built on open-source TransferQueue (TQ) [45]: rollout workers publish trajectory tensors to TQ, while the scheduling queue carries row keys and metadata. With worker prefetch enabled, the controller reads the fields needed for rewards, advantages, and batch organization. Training workers fetch expert-routing records, teacher top-k outputs, and sampling-replay support directly. This keeps bulky auxiliary tensors out of the controller's batch-assembly and dispatch path. Optional padding removal and compact expert-index representations further reduce payload size.

Overlapped data preparation. With prefetch enabled, workers start loading their next shard as samples arrive, before the full batch is assembled. Rows are assigned to data-parallel workers by token count, subject to equal row quotas within each training minibatch. Batch assembly preserves this ownership so workers can reuse the data already loaded. A background CPU loader overlaps TQ reads and preparation with the current policy update, buffering at most one future shard per worker. For pipeline-parallel routing replay, an optional path retains only the local stage's MoE-layer records after loading, reducing retained routing state and subsequent CPU-to-GPU transfer.

Training-aware replica recovery. When a serving actor is confirmed dead, the client removes it from routing and retries the request on a surviving replica within a bounded retry

budget. The failed standalone replica is restarted in the background after its previous actors release the allocation. Its replacement remains outside the serving pool until policy-weight synchronization completes. This separates immediate request failover from slower capacity restoration and prevents an unsynchronized replacement from generating training data.

G.2 Sparse-Attention Training

For index scoring and top-k selection, we use the fused kernel described in Section 4.6.1. The remaining operations use open-source kernels: FlashMLA [149] for the forward pass of sparse attention, and NVIDIA's CuTe DSL kernels from the cuDNN frontend [150] for the backward passes and for the score recomputation behind the indexer loss.

Selection reuse under activation recomputation. Under full activation recomputation, the forward pass of a layer runs twice: once without gradients, and once more during back-propagation. Recomputing a sparse layer would repeat its $O(L^2)$ selection. We run the selection only in the first pass and cache the selected indices, k per token, so the recomputed pass skips it entirely and both passes attend to exactly the same tokens. The first pass also skips the indexer loss, which only the pass with gradients needs. To keep the cache out of GPU memory between the two passes, we offload it asynchronously to pinned host memory and copy it back when the layer is recomputed.

Query-wise tensor parallelism and context parallelism. In DSA all attention heads of a query token attend to the same k selected entries, and the sparse attention kernels process the heads of a token together in order to share these entries. Megatron's tensor parallelism partitions the attention heads across ranks, which would leave every rank needing the selected entries of every query. Inside a sparse layer we therefore partition queries instead of heads: an all-to-all exchange converts the head-partitioned queries into a query-partitioned layout, each rank performs selection and sparse attention for its own queries with all heads, and the inverse exchange restores the layout of the output. Each query's entries are thus selected and gathered exactly once in the tensor-parallel group. Under context parallelism, a rank holds two non-adjacent chunks of every sequence (Megatron's zig-zag split). Because the kernels take a global position and a key range for every query, a rank processes this shard in a single call: keys are all-gathered over the context-parallel group into sequence order, and queries keep their global positions. GDN layers retain the same token ownership through the State CP implementation described in Section 5.4.3.

G.3 Gated DeltaNet Training

G.3.1 GDN Checkpointing

We also support selective recomputation of the complete GDN submodule, preserving packed-sequence metadata and integrating with Transformer Engine's FP8 checkpointing. Coordinating this checkpoint with input-normalization recomputation avoids redundant input retention by the FP8 projection.

The State CP algorithm is described in Section 5.4.3. We give the block-level measurement protocol and results across context lengths and CP degrees in Appendix G.3.2.

G.3.2 GDN State CP Benchmarks

We compare State CP with legacy head-parallel all-to-all CP for Transformer block 0 of a trained auxiliary 37B-param model (Table 22). This block contains GDN and a dense MLP. Both variants use identical trained weights, saved Graphwalks activations and NLL

adjoints, packing, and production blockwise FP8 settings. The GDN recurrence retains BF16/FP32 arithmetic. Measurements use TP=DP=1, and selective recomputation of GDN, normalization, and MLP in both variants.

The 32K input contains 32,768 real tokens in one document. The approximately 128K input contains 131,129 real tokens across eight complete documents of unequal length, padded to 134,144 tokens (2.30% overhead). The same CP=8-compatible padding is used for every CP degree and both implementations. Each variant has 16 timed samples after warmup, collected in ABBA order. Reported time is the median of the maximum rank time per sample, including forward, backward, and gradient finalization and synchronization. Memory is the maximum per-rank peak CUDA allocated memory for the isolated block. Throughput gain is $(t_{\text{legacy}}/t_{\text{State CP}} - 1) \times 100\%$ at a fixed token count. Optimizer updates and the remaining model blocks are excluded.

Table 22. FP8 block-level performance with selective activation recomputation enabled in both implementations. Gains measure the change in CP implementation at fixed recomputation settings.

		Time (ms)		Throughput	Peak memory (GiB/GPU)	
Context	**CP**	**Legacy**	**State CP**	**gain**	**Legacy**	**State CP**
32K	2	51.584	46.246	11.5%	5.283	4.416
32K	4	32.196	29.230	10.1%	3.180	2.747
32K	8	24.024	21.736	10.5%	2.126	1.912
≈128K	2	216.334	166.279	30.1%	17.815	14.762
≈128K	4	116.217	92.020	26.3%	9.446	7.935
≈128K	8	66.935	56.739	18.0%	5.267	4.529

For the 128K FP8 case, activations and NLL adjoints were captured from the full model with frozen parameters after captures retaining all parameter-gradient buffers ran out of memory for both CP implementations. This capture method was first verified to produce bitwise-identical activations and adjoints to the full-gradient capture at 32K. All timed block benchmarks compute both input and parameter gradients. The measurements therefore establish isolated block performance, without establishing full-model memory feasibility.

G.4 Rollout Cache Policy and Additional Optimizations

RadixCache provides prefix lookup and reuse, while HiCache [151] extends prefix storage from GPU into host memory. Preserving cached state across policy-weight synchronization provided no performance benefit in our experiments, so we disable this option and flush the cache after weight updates.

Additional optimizations. We also evaluated expert parallelism load balancing (EPLB) and batch-size-adaptive speculative decoding, which adjusts the number of MTP draft tokens to the batch size. Neither provided a meaningful performance improvement in our tested rollout configurations, so both remain disabled. With rollout replicas spanning at most two nodes, the gains from redistributing experts did not offset EPLB's overhead. This comparison concerns serving-time expert placement, while training-time routing interventions are discussed in Section 5.2.4. For adaptive speculative decoding, deeper drafting at smaller batch sizes increased memory overhead while yielding diminishing gains in accepted draft tokens, limiting its overall benefit.

G.5 SWE Execution Environments

Reusable execution environments. Each agent session runs in a persistent rootless container, retaining repository state and background processes across tool calls. Archived images and staged task artifacts are prepared once per host and reused through shared read-only layers. Each session receives private writable state on local storage. Separately staged harness packages allow image-based task environments to be reused across harness changes. These choices parallel DSec's composable environments and local-write design [152]. Before agent execution, we sanitize repository metadata to reduce unintended solution leakage, following the environment-sanitization principle discussed in DSec [152].

Distributed execution. Host-local gateways manage session lifecycle and grading. We distribute model-request ingress across hosts to spread chat parsing and serialization, with requests pinned to a specific data-parallel rank for prefix-cache reuse. Resource guards monitor session memory, process counts and writable disk usage. Teardown reclaims session processes and private filesystem state. The runtime supports hundreds of concurrent container sessions per host, scaling to tens of thousands across a training job.

H Inference Implementation and Benchmarks

H.1 Kernel Implementations

Phase-specific sparse attention. DSA [31] reduces long-context attention work, but its indexer and top-K selection can outweigh the savings at short contexts or small decode batches. We optimize these operations together with attention and support separate dense/sparse choices for prefill and decode (Section 4).

For sparse prefill, we use FlashMLA [149] and construct its input directly in the required layout, eliminating intermediate concatenation and copies. When prefill uses dense attention and decode uses DSA, a *K-only indexer* prepares the indexer cache for decode and prefill–decode handoff, omitting query-side processing, scoring, and top-K during prefill.

Approximate indexer routing. For long sparse prefill, the indexer can score each 1,024-token query block with a subset of 8 of its 64 heads before top-K selection. This reduces indexer cost by approximately 1.6–3.5$\times$ but changes the selected token set: only about 50–60% of the selected tokens coincide with the full-indexer top-K, so we treat it as a quality-affecting option.

Token-parallel prefill. Head-based tensor parallelism in the SGLang prefill path duplicates indexer scoring and top-K selection across ranks, while head padding adds unused attention computation. We instead partition contiguous query-token blocks within each attention tensor-parallel group. Each rank processes its own tokens with all heads, using two all-to-all exchanges to convert between token- and head-sharded layouts. Indexer keys and the MLA KV cache remain replicated so queries can access the full context. This partitioning applies only to eligible DSA prefill steps, those with attention tensor parallelism greater than one and at least 1,024 tokens, preserving token order in GDN layers.

We also prototyped optional replication of attention projection weights, which keeps token blocks local through the projections and removes both all-to-all exchanges, with an all-gather assembling output rows when required. This reduces communication at the cost of additional weight memory and less KV-cache capacity. It is not used in the evaluated configurations.

Fused and overlapped decode. At small decode batches, the indexer projections and query/key preparation process few token rows and expose limited compute parallelism. Kernel-launch overhead and intermediate memory reads and writes can therefore account for a substantial share of their latency. We fuse these operations to reduce launches and avoid materializing intermediate tensors, and overlap independent MLA and indexer work to shorten the decode critical path.

The SGLang path fuses the indexer query, key, and gate projections with on-the-fly FP8 activation quantization in a single Triton [153] kernel, retaining the distinct query and hidden-state inputs. Query/key preparation adapts SGLang's indexer fusion for the model's NeoX RoPE, scaling, gating, and key path. A single CUDA launch performs RoPE, Hadamard transformation, FP8 quantization, gating, key preparation, and the indexer-cache write, without materializing intermediate query tensors.

Intermediate BF16 rounding is preserved. Hadamard reductions can still differ because of summation order.

During CUDA-graph decode, MLA query preparation, KV updates, and output-gate computation run on a second CUDA stream, overlapping indexer scoring and top-K on the main stream. The streams synchronize before attention, shortening the decode critical path.

The TensorRT-LLM path fuses paged FP8 scoring with exact top-K, avoiding a materialized logits tensor. It also fuses key normalization, RoPE, quantization, and cache writes, and combines query projection with RoPE and FP8 packing. Independent key/query preparation overlaps where supported.

Fused gated normalization. Our SGLang implementation provides a three-stage Triton/CUDA pipeline for GatedNorm. Factoring the per-token root-mean-square (RMS) normalization scale out of the first gate projection lets one pass compute projection partials and normalization statistics without materializing the normalized hidden state. The remaining stages fuse reduction with RMS scaling and SiLU, then the second projection with sigmoid gating and output scaling. Splitting the hidden dimension exposes parallelism for small batches. Residual addition is fused into the first stage. We additionally use a persistent prefill kernel and cluster-based decode kernels that emit MXFP8 directly for the following projection.

H.2 Recurrent MTP Background

GigaChat 3.5 employed three sequential multi-token prediction (MTP) blocks with independent parameters. Each block uses multi-head latent attention (MLA) and a dense feed-forward network, sharing token embeddings and the output head with the target model. All three blocks were initialized from the native MTP block, whose weights had been trained throughout pre-training and alignment. With the target model frozen, we fine-tuned the draft blocks using teacher forcing on a mixture of instant and reasoning responses generated by the target from SFT prompts. Inspired by the acceptance-aligned objectives in AngelSpec [154], we first optimized per-position acceptance using a negative log-overlap loss, then switched to directly maximizing the expected accepted-prefix length across the three draft positions.

H.3 DFlash2 Training

To minimize train–inference discrepancies in the drafter's conditioning states, we extract target hidden states using the same SGLang implementation as at inference. We sample anchor positions within target-generated responses and provide the drafter with an anchor

token followed by a masked suffix. Following DSpark [155], we shift the labels so that each position, including the anchor, predicts its successor.

The training objective combines token-level cross-entropy, KL distillation, and selector supervision. Among the token losses we evaluated, combining cross-entropy with KL achieved the highest mean acceptance length, outperforming more complex acceptance-oriented objectives used in our recurrent MTP training recipe. Cross-entropy supervises the observed token, while KL divergence distills the frozen target's distribution over plausible alternatives, shaping the top-16 candidate lists subsequently reranked by the selector. When the reference token appears among these candidates, a separate cross-entropy loss trains the selector to favor it conditioned on its predecessor. To account for the varying contributions of draft positions to the expected accepted-prefix length, we adopt the dynamic position-aware weighting scheme of D-PACE [156].

In our ablations, the DSpark label shift increased mean accepted-prefix length, while D-PACE improved acceptance at later draft positions. In greedy replay using the same checkpoint and reasoning traces, the selector rarely changed the first prediction but intervened more frequently at later positions.

For the 708B target, we train for two epochs on instant and reasoning responses generated from SFT prompts, using the configuration in Table 23.

Table 23. DFlash2 training configuration for the frozen 708B target.

Quantity	Value
Block size	8
Candidates retained per slot	16
Selector rank	256
Draft transformer layers	5
Token loss	Cross-entropy + KL
Position weighting	D-PACE [156], $\alpha = 0.5$
Selector loss	Cross-entropy for targets in the top 16
Weights of CE, KL, and selector loss	1, 1, 1
Optimizer for attention, MLP, context projection	Muon [147]
Optimizer for remaining parameters	AdamW
Peak and final learning rates	6×10^{-4}, 6×10^{-5}
Weight decay	0
Schedule	Cosine, 5% warm-up
Gradient-norm clipping	1.0
Global batch size, maximum sequence length	128 sequences, 32,768 tokens
Anchors per sequence-parallel slice	512
Epochs	2

H.4 Component Benchmarks

GatedNorm performance. At 8,192 token rows, a prefill-specialized variant reduces the latency of a GatedNorm pair from 316.2 to 160.2 μs relative to the three-stage implementation (1.97×). With 8,192-token prefill chunks, overall step time decreases by 18.9 ms.

In a separate 16-row decode microbenchmark, an optimized kernel reduces pair latency from 11.5 to approximately 6.6 μs relative to an earlier cooperative kernel (1.74×). At concurrency 64, the corresponding decode-step reduction is approximately 0.45 ms.

A separate TP8, batch-size-1 evaluation of zero-centered GatedNorm compares the three-stage implementation with its predecessor. Normalization is 1.7–1.8× faster, and decode throughput increases from approximately 140 to 155 tokens/s.

Prefill scaling. We profile dense and sparse prefill over 10K–220K input tokens (Figure 12). At 220K, sparse attention reduces summed attention-kernel time from 433.38 to 100.33 ms (4.32×), while the profiled GPU step falls from 825.77 to 492.16 ms (1.68×). At 10K, the sparse GPU step is 14.5% slower: indexer overhead outweighs the attention savings at this input length.

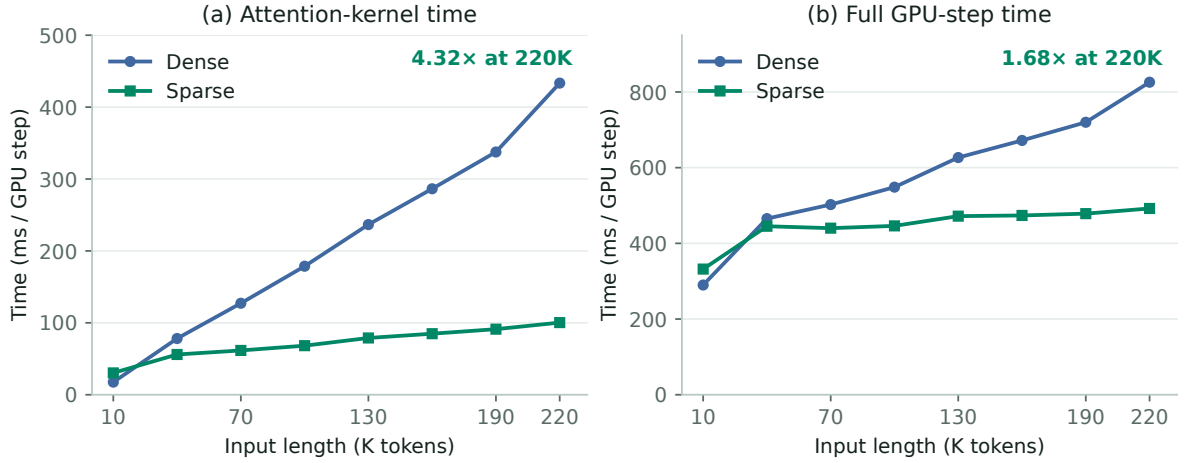

Figure 12. Sparse prefill benefits grow with input length. TensorRT-LLM profiles of (a) summed attention-kernel time and (b) full GPU-step time, averaged across chunked prefill. Lines connect eight measured input lengths. The number of prefill steps varies with input length.

Decode ablations. In a separate TP8 CUDA-graph decode study, we measure the sum of attention windows across 16 MLA layers on rank 0. We sweep context length at batch sizes 1, 8, and 32 (Figure 13). At batch size 1 and a 1M-token context, this time falls from 3.132 ms for baseline DSA to 2.853 ms with indexer fusions and then to 2.622 ms with MLA/indexer overlap. The combined reduction relative to baseline DSA is 16.3%. Dense MLA takes 7.497 ms in the same comparison. Across the context sweep, the estimated dense/sparse crossover moves from approximately 115K to 15K tokens after optimization in this setup.

In TensorRT-LLM, the decode fusions reduce summed attention-kernel time by 30.3% at 220K input and 1,000 output tokens, while full GPU-step time decreases from 16.986 to 16.318 ms (3.9%). Across 18 input lengths with matching 1,000-token outputs, GPU-step reductions range from 1.1% to 5.0%, with a median of 2.9%. Four early-termination cases are excluded.

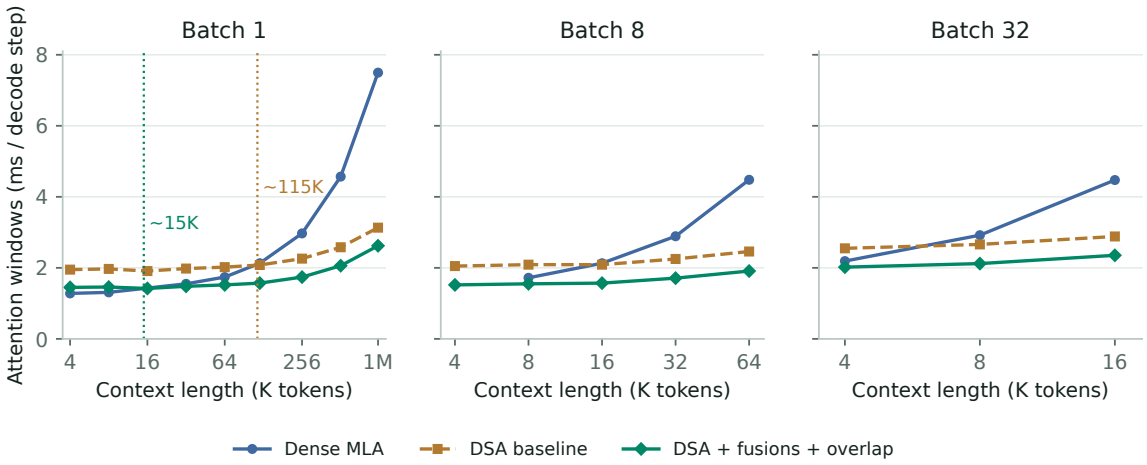

Figure 13. DSA decode across context lengths and batch sizes. TP8 with CUDA graphs. Markers show measured points, and lines connect them. Panels share a time scale, and KV-pool capacity limits larger-batch context ranges. Dotted lines estimate dense/DSA crossovers at batch 1 by interpolation in log-context space. Runtime settings differ for contexts of 256K and above. Times sum attention windows across 16 MLA layers, including preparation and output projection, and report medians after discarding the first eight decode steps.

References

[1] GigaChat Team. GigaChat3.5-432B-A28B: Released Model Configuration. Hugging Face, ai-sage, 2026. URL https://huggingface.co/ai-sage/GigaChat3.5-432B-A28B/blob/main/config.json.

[2] Valentin Mamedov, Evgenii Kosarev, Gregory Leleytner, Ilya Shchuckin, Valeriy Berezovskiy, Daniil Smirnov, et al. GigaChat Family: Efficient Russian Language Modeling Through Mixture of Experts Architecture. In Pushkar Mishra, Smaranda Muresan, and Tao Yu, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 93–106, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-253-4. doi: 10.18653/v1/2025.acl-demo.10. URL https://aclanthology.org/2025.acl-demo.10/.

[3] Nikita Sushko. GigaChat 3.5: Smaller, Faster, Stronger. Sber, Habr, 2026. URL https://habr.com/ru/companies/sberbank/articles/1055826/. Technical article, July 6, 2026; title translated from Russian.

[4] Common Crawl. Common Crawl. https://commoncrawl.org, 2026. Accessed 2026-11-22.

[5] Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Yitzhak Gadre, et al. DataComp-LM: In search of the next generation of training sets for language models. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/19e4ea30dded58259665db375885e412-Abstract-Datasets_and_Benchmarks_Track.html.

[6] Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Hamza Alobeidli, Alessandro Cappelli, et al. The RefinedWeb Dataset for Falcon LLM: Outperforming Curated Corpora with Web Data Only. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023,*

NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/fa3ed726cc5073b9c31e3e49a807789c-Abstract-Datasets_and_Benchmarks.html.

[7] Dan Su, Kezhi Kong, Ying Lin, Joseph Jennings, Brandon Norick, Markus Kliegl, et al. Nemotron-CC: Transforming Common Crawl into a Refined Long-Horizon Pretraining Dataset. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 2459–2475. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.ACL-LONG.123. URL https://doi.org/10.18653/v1/2025.acl-long.123.

[8] Laurie Burchell, Ona de Gibert, Nikolay Arefyev, Mikko Aulamo, Marta Bañón, Pinzhen Chen, et al. An Expanded Massive Multilingual Dataset for High-Performance Language Technologies (HPLT). In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 17452–17485, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.854. URL https://aclanthology.org/2025.acl-long.854/.

[9] Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, et al. The FineWeb Datasets: Decanting the Web for the Finest Text Data at Scale. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://dl.acm.org/doi/10.5555/3737916.3738886.

[10] Bettina Messmer, Vinko Sabolčec, and Martin Jaggi. Enhancing Multilingual LLM Pretraining with Model-Based Data Selection. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/5aee125f052c90e326dcf6f380df94f6-Abstract-Datasets_and_Benchmarks_Track.html.

[11] Mengjie Liu, Jiahui Peng, Wenchang Ning, Pei Chu, Jiantao Qiu, Ren Ma, et al. Dripper: Token-Efficient Main HTML Extraction with a Lightweight LM. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2, KDD 2026, Jeju Island, Korea, August 9-13, 2026*, pages 3258–3269. ACM, 2026. doi: 10.1145/3770855.3817915. URL https://doi.org/10.1145/3770855.3817915.

[12] Ren Ma, Jiantao Qiu, Chao Xu, Pei Chu, Kaiwen Liu, Pengli Ren, et al. AICC: Parse HTML Finer, Make Models Better - A 7.3T AI-Ready Corpus Built by a Model-Based HTML Parser. *CoRR*, abs/2511.16397, 2025. doi: 10.48550/ARXIV.2511.16397. URL https://doi.org/10.48550/arXiv.2511.16397.

[13] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of Tricks for Efficient Text Classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431. Association for Computational Linguistics, 2017. URL https://aclanthology.org/E17-2068/.

[14] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, et al. StarCoder 2 and The Stack v2: The Next Generation. *arXiv preprint arXiv:2402.19173*, 2024. URL https://arxiv.org/abs/2402.19173.

[15] Siming Huang, Tianhao Cheng, Jason Klein Liu, Weidi Xu, Jiaran Hao, Liuyihan Song, et al. OpenCoder: The Open Cookbook for Top-Tier Code Large Language Models.

In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 33167–33193. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.ACL-LONG.1591. URL https://doi.org/10.18653/v1/2025.acl-long.1591.

[16] Andrei Z. Broder. On the Resemblance and Containment of Documents. In *Proceedings of the Compression and Complexity of Sequences (SEQUENCES)*, pages 21–29. IEEE, 1997. doi: 10.1109/SEQUEN.1997.666900. URL https://doi.org/10.1109/SEQUEN.1997.666900.

[17] Zihan Qiu, Zeyu Huang, Kaiyue Wen, Peng Jin, Bo Zheng, Yuxin Zhou, et al. A Unified View of Attention and Residual Sinks: Outlier-Driven Rescaling is Essential for Transformer Training. *CoRR*, abs/2601.22966, 2026. doi: 10.48550/ARXIV.2601.22966. URL https://doi.org/10.48550/arXiv.2601.22966.

[18] Tianlong Chen, Xin Xu, Lu Yin, Hao Chen, Yang Wang, Shizhe Diao, et al. Progressive Residual Warmup for Language Model Pretraining. *CoRR*, abs/2603.05369, 2026. doi: 10.48550/ARXIV.2603.05369. URL https://doi.org/10.48550/arXiv.2603.05369.

[19] The Microsoft AI Team. MAI-Thinking-1: Building a Hill-Climbing Machine. Technical report, Microsoft AI, 2026. URL https://microsoft.ai/pdf/mai-thinking-1.pdf.

[20] Mitchell Wortsman, Peter J. Liu, Lechao Xiao, Katie E. Everett, Alexander A. Alemi, Ben Adlam, et al. Small-scale proxies for large-scale Transformer training instabilities. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=d8w0pmvXbZ.

[21] DeepSeek-AI. DeepSeek-V3 Technical Report. *CoRR*, abs/2412.19437, 2024. doi: 10.48550/ARXIV.2412.19437. URL https://doi.org/10.48550/arXiv.2412.19437.

[22] DeepSeek-AI. DeepGEMM: Clean and Efficient BLAS Kernel Library on GPU. GitHub repository, 2025. URL https://github.com/deepseek-ai/DeepGEMM.

[23] NVIDIA Corporation. cuBLAS: CUDA Basic Linear Algebra Subroutine Library, 2026. URL https://docs.nvidia.com/cuda/cublas/. Accessed: 2026-11-23.

[24] Meituan LongCat Team. LongCat-Flash Technical Report. *CoRR*, abs/2509.01322, 2025. doi: 10.48550/ARXIV.2509.01322. URL https://doi.org/10.48550/arXiv.2509.01322.

[25] Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, et al. Magpie: Alignment Data Synthesis from Scratch by Prompting Aligned LLMs with Nothing. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=Pnk7vMbznK.

[26] Xueliang Zhao, Wei Wu, Jian Guan, and Lingpeng Kong. PromptCoT: Synthesizing Olympiad-level Problems for Mathematical Reasoning in Large Language Models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 18167–18188. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-acl.935. URL http://dx.doi.org/10.18653/v1/2025.findings-acl.935.

[27] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, et al. RULER: What's the Real Context Size of Your Long-Context Language Models? In *First Conference on Language Modeling (COLM)*, 2024. URL https://openreview.net/forum?id=kIoBbc76Sy.

[28] Howard Yen, Tianyu Gao, Minmin Hou, Ke Ding, Daniel Fleischer, Peter Izsak, et al. HELMET: How to Evaluate Long-context Models Effectively and Thoroughly. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=293V3bJbmE.

[29] Uri Shaham, Elad Segal, Maor Ivgi, Avia Efrat, Ori Yoran, Adi Haviv, et al. SCROLLS: Standardized CompaRison Over Long Language Sequences. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 12007–12021, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.823. URL https://aclanthology.org/2022.emnlp-main.823/.

[30] Jason Priem, Heather A. Piwowar, and Richard Orr. OpenAlex: A fully-open index of scholarly works, authors, venues, institutions, and concepts. *CoRR*, abs/2205.01833, 2022. doi: 10.48550/ARXIV.2205.01833. URL https://doi.org/10.48550/arXiv.2205.01833.

[31] DeepSeek-AI. DeepSeek-V3.2: Pushing the Frontier of Open Large Language Models. *CoRR*, abs/2512.02556, 2025. doi: 10.48550/ARXIV.2512.02556. URL https://doi.org/10.48550/arXiv.2512.02556.

[32] Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. YaRN: Efficient Context Window Extension of Large Language Models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=wHBfxhZu1u.

[33] GLM. GLM-5: from Vibe Coding to Agentic Engineering. *CoRR*, abs/2602.15763, 2026. doi: 10.48550/ARXIV.2602.15763. URL https://doi.org/10.48550/arXiv.2602.15763.

[34] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, et al. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *CoRR*, abs/2402.03300, 2024. doi: 10.48550/ARXIV.2402.03300. URL https://doi.org/10.48550/arXiv.2402.03300.

[35] MiniMax. MiniMax-M1: Scaling Test-Time Compute Efficiently with Lightning Attention. *arXiv preprint arXiv:2506.13585*, 2025. URL https://arxiv.org/abs/2506.13585.

[36] Haizhong Zheng, Jiawei Zhao, and Beidi Chen. Prosperity before Collapse: How Far Can Off-Policy RL Reach with Stale Data on LLMs? In *The Fourteenth International Conference on Learning Representations, ICLR 2026, Rio de Janeiro, Brazil, April 23-27, 2026*. proceedings.iclr.cc, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/8590b946e4f0c2b1262cb5b047924653-Abstract-Conference.html.

[37] Wenhan Ma, Hailin Zhang, Liang Zhao, Yifan Song, Yudong Wang, Zhifang Sui, et al. Stabilizing MoE Reinforcement Learning by Aligning Training and Inference Routers. *CoRR*, abs/2510.11370, 2025. doi: 10.48550/ARXIV.2510.11370. URL https://doi.org/10.48550/arXiv.2510.11370.

[38] Violet Xiang, Chase Blagden, Rafael Rafailov, Nathan Lile, Sang T. Truong, Chelsea Finn, et al. Just Enough Thinking: Efficient Reasoning with Adaptive Length Penalties Reinforcement Learning. *CoRR*, abs/2506.05256, 2025. doi: 10.48550/ARXIV.2506.05256. URL https://doi.org/10.48550/arXiv.2506.05256.

[39] Wenhan Ma, Jianyu Wei, Liang Zhao, Hailin Zhang, Bangjun Xiao, Lei Li, et al. MOPD: Multi-Teacher On-Policy Distillation for Capability Integration in LLM Post-Training. *CoRR*, abs/2606.30406, 2026. doi: 10.48550/ARXIV.2606.30406. URL

https://doi.org/10.48550/arXiv.2606.30406.

[40] LLM-Core Xiaomi. MiMo-V2-Flash Technical Report. *CoRR*, abs/2601.02780, 2026. doi: 10.48550/ARXIV.2601.02780. URL https://doi.org/10.48550/arXiv.2601.02780.

[41] Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, et al. On-Policy Distillation of Language Models: Learning from Self-Generated Mistakes. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=3zKtaqxLhW.

[42] Zechen Sun, Zhiwei Zhang, Fei Zhao, Juntao Li, Mu Chuan, Huayu Deng, et al. D^3-MOPD: Adaptive Dynamic Domain ScheDuling for Efficient Multi-Teacher Distillation. *CoRR*, abs/2608.24987, 2026. doi: 10.48550/ARXIV.2608.24987. URL https://doi.org/10.48550/arXiv.2608.24987.

[43] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, et al. HybridFlow: A Flexible and Efficient RLHF Framework. In *Proceedings of the Twentieth European Conference on Computer Systems, EuroSys 2025, Rotterdam, The Netherlands, 30 March 2025 - 3 April 2025*, pages 1279–1297. ACM, 2025. doi: 10.1145/3689031.3696075. URL https://doi.org/10.1145/3689031.3696075.

[44] Jiadong Guo, Xin Ji, Letian Ruan, Teng Ma, Chenyang Zhao, Yueming Yuan, et al. Updating 1T Parameters in Seconds—P2P Weight Transfer in Large Scale Distributed RL. LMSYS Org engineering blog, April 2026. URL https://www.lmsys.org/blog/2026-04-29-p2p-update.

[45] Zhenyu Han, Ansheng You, Haibo Wang, Kui Luo, Guang Yang, Wenqi Shi, et al. AsyncFlow: An Asynchronous Streaming RL Framework for Efficient LLM Post-Training. *CoRR*, abs/2507.01663, 2025. doi: 10.48550/ARXIV.2507.01663. URL https://doi.org/10.48550/arXiv.2507.01663.

[46] NVIDIA. Megatron Core: Gated DeltaNet Parallelism Implementation. GitHub repository, 2026. URL https://github.com/NVIDIA/Megatron-LM/blob/main/megatron/core/ssm/gated_delta_net/common.py.

[47] Ze-Wei Liou, Harris Nover, Amey Naik, Andrii Skliar, Igor Shovkun, Ido Shahaf, et al. ReplaySSM: Cache SSM Inputs, Not State. *CoRR*, abs/2610.13778, 2026. doi: 10.48550/ARXIV.2610.13778. URL https://doi.org/10.48550/arXiv.2610.13778.

[48] NVIDIA. TensorRT-LLM. GitHub repository, 2023. URL https://github.com/NVIDIA/TensorRT-LLM.

[49] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, et al. SGLang: Efficient Execution of Structured Language Model Programs. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

[50] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up Inference Acceleration of Large Language Models via Training-Time Test. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/c7b5a35ea98b62512a869c19ea7b03cb-Abstract-Conference.html.

[51] Inco AI. DFlash 2: Keep Drafting Parallel. https://inco.ai/blog/dflash2/, 2026.

[52] Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block Diffusion for Flash Speculative Decoding. In *Proceedings of the 43rd International Conference on Machine Learning*, pages 15343–15355, 2026. URL https://proceedings.mlr.press/v306/chen26bv.html.

[53] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R. Narasimhan. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=roNSXZpUDN.

[54] Victor Barrès, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating Conversational Agents in a Dual-Control Environment. *CoRR*, abs/2506.07982, 2025. doi: 10.48550/ARXIV.2506.07982. URL https://doi.org/10.48550/arXiv.2506.07982.

[55] Quan Shi, Alexandra Zytek, Pedram Razavi, Karthik Narasimhan, and Victor Barrès. τ-Knowledge: Evaluating Conversational Agents over Unstructured Knowledge. *CoRR*, abs/2603.04370, 2026. doi: 10.48550/ARXIV.2603.04370. URL https://doi.org/10.48550/arXiv.2603.04370.

[56] Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, et al. Generalizing Verifiable Instruction Following. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/46499a0622ecf568b72d17b61e45dbd5-Abstract-Datasets_and_Benchmarks_Track.html.

[57] Saibo Geng, Hudson Cooper, Michał Moskal, Samuel Jenkins, Julian Berman, Nathan Ranchin, et al. JSONSchemaBench: A Rigorous Benchmark of Structured Outputs for Language Models. *arXiv preprint arXiv:2501.10868*, 2025. URL https://arxiv.org/abs/2501.10868.

[58] Huaixiu Steven Zheng, Swaroop Mishra, Hugh Zhang, Xinyun Chen, Minmin Chen, Azade Nova, et al. NATURAL PLAN: Benchmarking LLMs on Natural Language Planning. *CoRR*, abs/2406.04520, 2024. doi: 10.48550/ARXIV.2406.04520. URL https://doi.org/10.48550/arXiv.2406.04520.

[59] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, et al. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

[60] OpenAI. Introducing SWE-bench Verified, 2024. URL https://openai.com/index/introducing-swe-bench-verified/.

[61] John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, et al. SWE-smith: Scaling Data for Software Engineering Agents. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/8b86cf5ace600c48fd188efbb8dedec8-Abstract-Datasets_and_Benchmarks_Track.html.

[62] Mike A. Merrill, Alexander Glenn Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, et al. Terminal-Bench: Benchmarking Agents on Hard, Realistic Tasks

in Command Line Interfaces. In *The Fourteenth International Conference on Learning Representations, ICLR 2026, Rio de Janeiro, Brazil, April 23-27, 2026*. proceedings.iclr.cc, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/444a3737adaee10d86ad2ef5f74468e6-Abstract-Conference.html.

[63] E. Kelly Buchanan, Crystal Zhou, Scott W. Linderman, and Christopher Ré. Benchmarks as Software: Terminal-Bench and the Case for Continuous Benchmarks. *arXiv preprint arXiv:2611.08417*, 2026. URL https://arxiv.org/abs/2611.08417.

[64] Jingzhe Ding, Shengda Long, Changxin Pu, Huan Zhou, Hongwan Gao, Xiang Gao, et al. NL2Repo-Bench: Towards Long-Horizon Repository Generation Evaluation of Coding Agents. *CoRR*, abs/2512.12730, 2025. doi: 10.48550/ARXIV.2512.12730. URL https://doi.org/10.48550/arXiv.2512.12730.

[65] Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, et al. SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks. *CoRR*, abs/2602.12670, 2026. doi: 10.48550/ARXIV.2602.12670. URL https://doi.org/10.48550/arXiv.2602.12670.

[66] Suhana Bedi, Ryan Welch, Ethan Steinberg, Michael Wornow, Taeil Matthew Kim, Haroun Ahmed, et al. HealthAdminBench: Evaluating Computer-Use Agents on Healthcare Administration Tasks. *CoRR*, abs/2604.09937, 2026. doi: 10.48550/ARXIV.2604.09937. URL https://doi.org/10.48550/arXiv.2604.09937.

[67] Cowork Bench contributors. Cowork Bench, 2026. URL https://github.com/cowork-bench/cowork_bench.

[68] Yuetai Li, Yichen Feng, Zhangchen Xu, Zixian Ma, Kaiyuan Zheng, Fengqing Jiang, et al. JobBench: Aligning Agent Work With Human Will. *CoRR*, abs/2605.26329, 2026. doi: 10.48550/ARXIV.2605.26329. URL https://doi.org/10.48550/arXiv.2605.26329.

[69] Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments, 2026. URL https://doi.org/10.5281/zenodo.20953922.

[70] Wenqi Huang, Charley Lee, Leonard Tng, and Serena Ge. DeepSWE: Measuring Frontier Coding Agents on Original, Long-Horizon Engineering Tasks. *CoRR*, abs/2607.07946, 2026. doi: 10.48550/ARXIV.2607.07946. URL https://doi.org/10.48550/arXiv.2607.07946.

[71] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, et al. LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

[72] OpenAI. Graphwalks, 2025. URL https://huggingface.co/datasets/openai/graphwalks.

[73] Artificial Analysis. Artificial Analysis Long Context Reasoning (AA-LCR), 2025. URL https://huggingface.co/datasets/ArtificialAnalysis/AA-LCR.

[74] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, et al. GPQA: A Graduate-Level Google-Proof Q&A Benchmark. In *First Conference on Language Modeling (COLM)*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

[75] Thang Luong, Dawsen Hwang, Hoang H. Nguyen, Golnaz Ghiasi, Yuri Chervonyi, Insuk Seo, et al. Towards Robust Mathematical Reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pages 35418–35442. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.EMNLP-MAIN.1794. URL https://doi.org/10.18653/v1/2025.emnlp-main.1794.

[76] Center for AI Safety, Scale AI, and HLE Contributors Consortium. A Benchmark of Expert-Level Academic Questions to Assess AI Capabilities. *Nature*, 649(8099): 1139–1146, 2026. doi: 10.1038/s41586-025-09962-4. URL https://doi.org/10.1038/s41586-025-09962-4.

[77] Mathematical Association of America. American Invitational Mathematics Examination (AIME I and II), 2025, 2025. URL https://maa.org/news/aime-thresholds-are-available/.

[78] Mathematical Association of America. American Invitational Mathematics Examination (AIME I and II), 2026, 2026. URL https://maa.org/news/2025-26-aime-thresholds-are-now-available/.

[79] Harvard-MIT Mathematics Tournament. Archive of February 2025, 2025. URL https://www.hmmt.org/www/archive/282.

[80] Harvard-MIT Mathematics Tournament. Archive of November 2025, 2025. URL https://www.hmmt.org/www/archive/291.

[81] Jialin Yang, Dongfu Jiang, Tony He, Sherman Siu, Yuxuan Zhang, Disen Liao, et al. StructEval: Benchmarking LLMs' Capabilities to Generate Structural Outputs. *Trans. Mach. Learn. Res.*, 2026, 2026. URL https://openreview.net/forum?id=buDwV7LUA7.

[82] Nikita Martynov, Anastasia Mordasheva, Dmitriy Gorbetskiy, Danil Astafurov, Ulyana Isaeva, Elina Basyrova, et al. Eye of Judgement: Dissecting the Evaluation of Russian-speaking LLMs with POLLUX. *CoRR*, abs/2505.24616, 2025. doi: 10.48550/ARXIV.2505.24616. URL https://doi.org/10.48550/arXiv.2505.24616.

[83] Greg Kamradt. Needle In A Haystack: Pressure Testing LLMs, 2023. URL https://github.com/gkamradt/needle-in-a-haystack.

[84] Fan Zhou, Zengzhi Wang, Nikhil Ranjan, Zhoujun Cheng, Liping Tang, Guowei He, et al. MegaMath: Pushing the Limits of Open Math Corpora. *CoRR*, abs/2504.02807, 2025. doi: 10.48550/ARXIV.2504.02807. URL https://doi.org/10.48550/arXiv.2504.02807.

[85] Amir Hossein Kargaran, Ayyoob Imani, François Yvon, and Hinrich Schütze. GlotLID: Language Identification for Low-Resource Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, Findings of ACL, pages 6155–6218. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.FINDINGS-EMNLP.410. URL https://doi.org/10.18653/v1/2023.findings-emnlp.410.

[86] Janek Bevendorff, Benno Stein, Matthias Hagen, and Martin Potthast. Elastic ChatNoir: Search Engine for the ClueWeb and the Common Crawl. In *Advances in Information Retrieval: 40th European Conference on IR Research (ECIR 2018)*, volume 10772 of *Lecture Notes in Computer Science*, pages 820–824. Springer, 2018. doi: 10.1007/978-3-319-76941-7_83. URL https://doi.org/10.1007/978-3-319-76941-7_83.

[87] Adrien Barbaresi. Trafilatura: A Web Scraping Library and Command-Line Tool for Text Discovery and Extraction. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, pages 122–131. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.acl-demo.15. URL https://aclanthology.org/2021.acl-demo.15/.

[88] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, et al. Measuring Massive Multitask Language Understanding. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

[89] Qwen Team. Qwen3 Technical Report. *CoRR*, abs/2505.09388, 2025. doi: 10.48550/ARXIV.2505.09388. URL https://doi.org/10.48550/arXiv.2505.09388.

[90] DeepSeek-AI, Qihao Zhu, Daya Guo, Zhihong Shao, Dejian Yang, Peiyi Wang, et al. DeepSeek-Coder-V2: Breaking the Barrier of Closed-Source Models in Code Intelligence. *CoRR*, abs/2406.11931, 2024. doi: 10.48550/ARXIV.2406.11931. URL https://doi.org/10.48550/arXiv.2406.11931.

[91] NVIDIA. Nemotron 3 Nano: Open, Efficient Mixture-of-Experts Hybrid Mamba-Transformer Model for Agentic Reasoning. *CoRR*, abs/2512.20848, 2025. doi: 10.48550/ARXIV.2512.20848. URL https://doi.org/10.48550/arXiv.2512.20848.

[92] Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, et al. SmolLM2: When Smol Goes Big - Data-Centric Training of a Small Language Model. *CoRR*, abs/2502.02737, 2025. doi: 10.48550/ARXIV.2502.02737. URL https://doi.org/10.48550/arXiv.2502.02737.

[93] Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. OpenWebMath: An Open Dataset of High-Quality Mathematical Web Text. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=jKHmjlpViu.

[94] Rabeeh Karimi Mahabadi, Sanjeev Satheesh, Shrimai Prabhumoye, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-CC-Math: A 133 Billion-Token-Scale High Quality Math Pretraining Dataset. In *The Fourteenth International Conference on Learning Representations, ICLR 2026, Rio de Janeiro, Brazil, April 23-27, 2026*. proceedings.iclr.cc, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/84f421a21dedc97cdae7d97623ebcb7b-Abstract-Conference.html.

[95] Hynek Kydlíček, Guilherme Penedo, and Leandro von Werra. FinePDFs. Hugging Face dataset, 2025. URL https://huggingface.co/datasets/HuggingFaceFW/finepdfs.

[96] Jake Poznanski, Jon Borchardt, Jason Dunkelberger, Regan Huff, Daniel Lin, Aman Rangapur, et al. olmOCR: Unlocking Trillions of Tokens in PDFs with Vision Language Models. *CoRR*, abs/2502.18443, 2025. doi: 10.48550/ARXIV.2502.18443. URL https://doi.org/10.48550/arXiv.2502.18443.

[97] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, et al. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *CoRR*, abs/1803.05457, 2018. doi: 10.48550/ARXIV.1803.05457. URL https://doi.org/10.48550/arXiv.1803.05457.

[98] Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen Marcus McAleer, Albert Q. Jiang, et al. Llemma: An Open Language Model for Mathematics. In *The Twelfth International Conference on Learning Representations,*

ICLR 2024, Vienna, Austria, May 7-11, 2024. OpenReview.net, 2024. URL https://openreview.net/forum?id=4WnqRR915j.

[99] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, et al. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

[100] Shuo Yang, Wei-Lin Chiang, Lianmin Zheng, Joseph E. Gonzalez, and Ion Stoica. Rethinking Benchmark and Contamination for Language Models with Rephrased Samples. *CoRR*, abs/2311.04850, 2023. doi: 10.48550/ARXIV.2311.04850. URL https://doi.org/10.48550/arXiv.2311.04850.

[101] Alfred V. Aho and Margaret J. Corasick. Efficient String Matching: An Aid to Bibliographic Search. *Communications of the ACM*, 18(6):333–340, 1975. doi: 10.1145/360825.360855.

[102] Stella Biderman, Hailey Schoelkopf, Lintang Sutawika, Leo Gao, Jonathan Tow, Baber Abbasi, et al. Lessons from the Trenches on Reproducible Evaluation of Language Models. *CoRR*, abs/2405.14782, 2024. doi: 10.48550/ARXIV.2405.14782. URL https://doi.org/10.48550/arXiv.2405.14782.

[103] averoo. FakeMMLU. Hugging Face dataset, 2023. URL https://huggingface.co/datasets/averoo/fake_mmlu.

[104] averoo. BabyMMLU2. Hugging Face dataset, 2024. URL https://huggingface.co/datasets/averoo/baby_mmlu2.

[105] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, et al. MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/ad236edc564f3e3156e1b2feafb99a24-Abstract-Datasets_and_Benchmarks_Track.html.

[106] Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, et al. AGIEval: A Human-Centric Benchmark for Evaluating Foundation Models. In *Findings of the Association for Computational Linguistics: NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, Findings of ACL, pages 2299–2314. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.FINDINGS-NAACL.149. URL https://doi.org/10.18653/v1/2024.findings-naacl.149.

[107] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a Machine Really Finish Your Sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800. Association for Computational Linguistics, 2019. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472/.

[108] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An Adversarial Winograd Schema Challenge at Scale. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 8732–8740. AAAI Press, 2020. doi: 10.1609/AAAI.V34I05.6399. URL https:

//doi.org/10.1609/aaai.v34i05.6399.

[109] Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611. Association for Computational Linguistics, 2017. doi: 10.18653/v1/P17-1147. URL https://aclanthology.org/P17-1147/.

[110] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, et al. Natural Questions: A Benchmark for Question Answering Research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019. doi: 10.1162/tacl_a_00276. URL https://aclanthology.org/Q19-1026/.

[111] Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. Latent Retrieval for Weakly Supervised Open Domain Question Answering. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6086–6096. Association for Computational Linguistics, 2019. doi: 10.18653/v1/P19-1612. URL https://aclanthology.org/P19-1612/.

[112] Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. DROP: A Reading Comprehension Benchmark Requiring Discrete Reasoning Over Paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2368–2378. Association for Computational Linguistics, 2019. doi: 10.18653/v1/N19-1246. URL https://aclanthology.org/N19-1246/.

[113] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, et al. Challenging BIG-Bench Tasks and Whether Chain-of-Thought Can Solve Them. In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, Findings of ACL, pages 13003–13051. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.FINDINGS-ACL.824. URL https://doi.org/10.18653/v1/2023.findings-acl.824.

[114] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, et al. Training Verifiers to Solve Math Word Problems. *CoRR*, abs/2110.14168, 2021. doi: 10.48550/ARXIV.2110.14168. URL https://doi.org/10.48550/arXiv.2110.14168.

[115] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, et al. Measuring Mathematical Problem Solving With the MATH Dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

[116] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, et al. Solving Quantitative Reasoning Problems with Language Models. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/18abbeef8cfe9203fdf9053c9c4fe191-Abstract.html.

[117] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, et al. Let's Verify Step by Step. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net,

2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

[118] Ilya Kuleshov, Ilin Pavel, Nikolay Kompanets, Ksenia Sycheva, and Aleksandr Nikolich. DOoM: Difficult Olympiads of Math. *CoRR*, abs/2509.23529, 2025. doi: 10.48550/ARXIV.2509.23529. URL https://doi.org/10.48550/arXiv.2509.23529.

[119] Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, et al. Language models are multilingual chain-of-thought reasoners. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=fR3wGCk-IXp.

[120] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, et al. Evaluating Large Language Models Trained on Code. *CoRR*, abs/2107.03374, 2021. doi: 10.48550/ARXIV.2107.03374. URL https://doi.org/10.48550/arXiv.2107.03374.

[121] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://openreview.net/forum?id=1qvx610Cu7.

[122] Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, et al. Program Synthesis with Large Language Models. *CoRR*, abs/2108.07732, 2021. doi: 10.48550/ARXIV.2108.07732. URL https://doi.org/10.48550/arXiv.2108.07732.

[123] Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, et al. MultiPL-E: A Scalable and Polyglot Approach to Benchmarking Neural Code Generation. *IEEE Trans. Software Eng.*, 49(7):3675–3691, 2023. doi: 10.1109/TSE.2023.3267446. URL https://doi.org/10.1109/TSE.2023.3267446.

[124] Alex Gu, Baptiste Roziere, Hugh James Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida Wang. CRUXEval: A Benchmark for Code Reasoning, Understanding and Execution. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 16568–16621. PMLR, 2024. URL https://proceedings.mlr.press/v235/gu24c.html.

[125] Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, et al. BigCodeBench: Benchmarking Code Generation with Diverse Function Calls and Complex Instructions. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=YrycTjllL0.

[126] Vladislav Mikhailov, Tatiana Shamardina, Max Ryabinin, Alena Pestova, Ivan Smurov, and Ekaterina Artemova. RuCoLA: Russian Corpus of Linguistic Acceptability. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5207–5227, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.348. URL https://aclanthology.org/2022.emnlp-main.348/.

[127] Andrey S. Chetvergov, Rinat S. Sharafetdinov, Marina M. Polukoshko, Vadim A. Akhmetov, Nataliia A. Oruzheinikova, Yegor S. Anichkov, et al. SLAVA: Benchmark of Sociopolitical Landscape and Value Analysis. *Proceedings of the Institute for System Programming of the RAS*, 37(3):171–184, 2025. doi: 10.15514/ISPRAS-2025-37(3)-12.

URL https://github.com/ikanam-ai/SLAVA.

[128] Nikita Martynov, Mark Baushenko, Anastasia Kozlova, Katerina Kolomeytseva, Aleksandr Abramov, and Alena Fenogenova. A Methodology for Generative Spelling Correction via Natural Spelling Errors Emulation across Multiple Domains and Languages. In Yvette Graham and Matthew Purver, editors, *Findings of the Association for Computational Linguistics: EACL 2024*, pages 138–155, St. Julian's, Malta, March 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-eacl.10. URL https://aclanthology.org/2024.findings-eacl.10/.

[129] Pierre Andrews, Mikel Artetxe, Mariano Coria Meglioli, Marta R. Costa-jussà, Joe Chuang, David Dale, et al. BOUQuET: dataset, Benchmark and Open initiative for Universal Quality Evaluation in Translation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pages 27515–27535. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.EMNLP-MAIN.1400. URL https://doi.org/10.18653/v1/2025.emnlp-main.1400.

[130] Maja Popović. chrF: character n-gram F-score for automatic MT evaluation. In Ondřej Bojar, Rajan Chatterjee, Christian Federmann, Barry Haddow, Chris Hokamp, Matthias Huck, et al., editors, *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/W15-3049. URL https://aclanthology.org/W15-3049/.

[131] Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, et al. LongBench v2: Towards Deeper Understanding and Reasoning on Realistic Long-context Multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 3639–3664. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.ACL-LONG.183. URL https://doi.org/10.18653/v1/2025.acl-long.183.

[132] Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Hao, et al. ∞Bench: Extending Long Context Evaluation Beyond 100K Tokens. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15262–15277. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.814. URL https://aclanthology.org/2024.acl-long.814/.

[133] SWE-agent Team. mini-SWE-agent. GitHub repository, 2025. URL https://github.com/SWE-agent/mini-swe-agent. Accessed: 2026-11-16.

[134] Anomaly. OpenCode: The Open Source Coding Agent. GitHub repository, 2025. URL https://github.com/anomalyco/opencode. Accessed: 2026-11-14.

[135] Qijia Shen, Zhiqi Huang, Vamsidhar Kamanuru, Aznaur Aliev, Jay Rainton, Ahmed Awelkair, et al. SETA: Scaling Environments for Terminal Agents. *CoRR*, abs/2607.10891, 2026. doi: 10.48550/ARXIV.2607.10891. URL https://doi.org/10.48550/arXiv.2607.10891.

[136] LangChain. Deep Agents, 2025. URL https://github.com/langchain-ai/deepagents. GitHub repository. Accessed: 2026-11-14.

[137] Mario Zechner. Pi, 2025. URL https://github.com/earendil-works/pi. GitHub repository. Accessed: 2026-11-14.

[138] Nous Research. Hermes Agent, 2026. URL https://github.com/NousResearch/hermes-agent. GitHub repository. Accessed: 2026-11-14.

[139] OpenClaw contributors. OpenClaw, 2026. URL https://github.com/openclaw/openclaw. GitHub repository. Accessed: 2026-11-14.

[140] Qwen Code contributors. Qwen Code, 2025. URL https://github.com/QwenLM/qwen-code. GitHub repository. Accessed: 2026-11-14.

[141] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, et al. OpenHands: An Open Platform for AI Software Developers as Generalist Agents. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=OJd3ayDDoF.

[142] Mathematical Association of America. MAA Invitational Competitions, 2026. URL https://maa.org/maa-invitational-competitions/. Accessed: 2026-11-14.

[143] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, et al. Instruction-Following Evaluation for Large Language Models. *CoRR*, abs/2311.07911, 2023. doi: 10.48550/ARXIV.2311.07911. URL https://doi.org/10.48550/arXiv.2311.07911.

[144] Kiran Vodrahalli, Santiago Ontanon, Nilesh Tripuraneni, Kelvin Xu, Sanil Jain, Rakesh Shivanna, et al. Michelangelo: Long Context Evaluations Beyond Haystacks via Latent Structure Queries. *CoRR*, abs/2409.12640, 2024. doi: 10.48550/ARXIV.2409.12640. URL https://doi.org/10.48550/arXiv.2409.12640.

[145] Mike Merrill and Alex Shaw. Terminus: A Research-Preview Agent for Evaluating Language Models in the Terminal, 2025. URL https://www.tbench.ai/news/terminus.

[146] LLM-Core Xiaomi. MiMo-V2.6: Scaling Reinforcement Learning Towards Self-Improvement. *CoRR*, abs/2610.11959, 2026. doi: 10.48550/ARXIV.2610.11959. URL https://doi.org/10.48550/arXiv.2610.11959.

[147] Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, et al. Muon: An optimizer for hidden layers in neural networks. https://kellerjordan.github.io/posts/muon/, 2024.

[148] Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, et al. Muon is Scalable for LLM Training. *CoRR*, abs/2502.16982, 2025. doi: 10.48550/ARXIV.2502.16982. URL https://doi.org/10.48550/arXiv.2502.16982.

[149] DeepSeek-AI. FlashMLA: Efficient Multi-head Latent Attention Kernels. GitHub repository, 2025. URL https://github.com/deepseek-ai/FlashMLA.

[150] NVIDIA. DeepSeek Sparse Attention (DSA). cuDNN Frontend documentation, 2026. URL https://docs.nvidia.com/deeplearning/cudnn/latest/fe-oss-apis/dsa.html.

[151] SGLang Team. HiCache in SGLang: Hybrid-Model Caching. SGLang Blog, 2026. URL https://www.sglang.io/blog/hicache.

[152] Jialiang Huang, Hongxuan Tang, Jingchang Chen, Yuxuan Liu, Yixiao Chen, Yuan Cheng, et al. DeepSeek Elastic Compute (DSec): A Sandbox Infrastructure for Effective Agentic Training at Scale. *CoRR*, abs/2609.22978, 2026. doi: 10.48550/ARXIV.2609.22978. URL https://doi.org/10.48550/arXiv.2609.22978.

[153] Philippe Tillet, Hsiang-Tsung Kung, and David D. Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, MAPL@PLDI 2019, Phoenix, AZ, USA, June 22, 2019*, pages 10–19. ACM, 2019. doi: 10.1145/3315508.3329973. URL https://doi.org/10.1145/3315508.3329973.

[154] Hong Liu, Rui Cen, Junhan Shi, Guangshuo Qin, Jiebin Zhang, Tianyu Liu, et al. AngelSpec: Towards Real-World High Performance Inference with Speculative Decoding. *CoRR*, abs/2607.25852, 2026. doi: 10.48550/ARXIV.2607.25852. URL https://doi.org/10.48550/arXiv.2607.25852.

[155] Xin Cheng, Xingkai Yu, Chenze Shao, Jiashi Li, Yunfan Xiong, Yi Qian, et al. DSpark: Confidence-Scheduled Speculative Decoding with Semi-Autoregressive Generation. *CoRR*, abs/2607.05147, 2026. doi: 10.48550/ARXIV.2607.05147. URL https://doi.org/10.48550/arXiv.2607.05147.

[156] Tianyu Wu, Yu Yao, Zhenting Qi, Han Zheng, Zhuohan Wang, Haoran Ma, et al. D-PACE: Dynamic Position-Aware Cross-Entropy for Parallel Speculative Drafting. *CoRR*, abs/2605.18810, 2026. doi: 10.48550/ARXIV.2605.18810. URL https://doi.org/10.48550/arXiv.2605.18810.