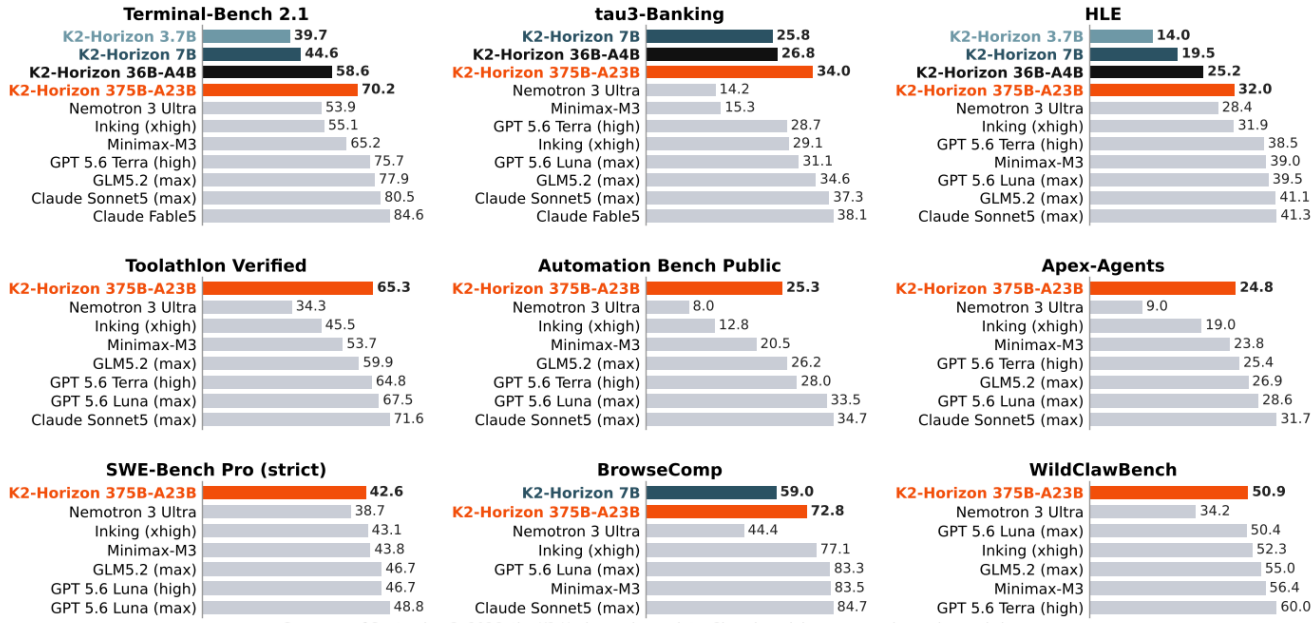
arXiv:2611.21003v1 [cs.AI] 16 Nov 2026

K2-Horizon: A Frontier Fleet of Open Source Language Models

K2 Team, Institute of Foundation Models*

Mohamed bin Zayed University of Artificial Intelligence

We present K2-Horizon, a fleet of six fully open language models, from a 0.9B dense model to a 375B-A23B MoE, built for general-purpose reasoning and agentic work across math and science, software engineering, deep research, office automation, and beyond. The models lead or are competitive with strong open models of their size on agentic and coding benchmarks. On the modeling side, to strengthen reasoning, we introduce Mixture-of-Value Attention (MoVA), applying MoE to attention value projections, and softplus attention gating, a more flexible, theoretically motivated gate. Our 36B-A4B model uses both and nearly matches a 32B dense model trained on the same data, with one-seventh the active parameters. We also introduce Uno, lossless diffusion adapters that speed up decoding by up to 2.2× without a separate draft model. Our release follows the LLM360 definition of open source for transparency and reproducibility: the full development history, including intermediate checkpoints from every training stage, pre-training data, training logs, and all training, evaluation, and infrastructure code, notably our production-scale xLLM framework for dense and MoE training and the RL360 reinforcement-learning stack. Beyond the data itself, we release and document in detail every pipeline that built it, the part of model development that open releases most rarely disclose. These pipelines produced roughly 10T synthetic pre-training tokens, showing that synthetic data can be a major ingredient of pre-training at scale. Four of the models share one 22T-token pre-training corpus and recipe, revealing consistent scaling behavior and enabling controlled studies of scale and sparsity. All artifacts are available at https://ifm.ai/k2/.



Scores as of September 3, 2026, the K2-Horizon release date. Closed-model scores may have changed since. K2-Horizon 3.7B and 7B scores on Terminal-Bench 2.1 and HLE are from the later anti-doom-loop checkpoints.

*Correspondence to: {hector.liu, eric.xing}@mbzuai.ac.ae

Released Artifacts

`K2-Horizon` is released together with the artifacts used to build it: for all six models, the final weights, intermediate checkpoints from every training stage, quantized builds and training logs, together with the training data and the code for each step of the pipeline. Every entry below links to the public artifact, and section numbers point to where this report describes it.

6	380	23	21.6 TB	58	17
models	checkpoints	quantized builds	training data	training runs	code repositories

Models and checkpoints

K2-Horizon	375B-A23B	36B-A4B	32B	7B	3.7B	0.9B
Architecture	MoE	MoE, MoVA	Dense	Dense	Dense	Dense
🤗 Final weights	758 GB	75 GB	70 GB	18 GB	10 GB	2.2 GB
🤗 Checkpoints	68 tags	69 tags	69 tags	70 tags	68 tags	36 tags
Training logs	8 runs	7 runs	7 runs	14 runs	11 runs	11 runs
🤗 FP8, NVFP4	FP8, NVFP4	FP8	FP8, NVFP4	FP8	–	–
🤗 GGUF	–	6 quants	6 quants	6 quants	6 quants	6 quants
🤗 MLX	–	bf16, 8, 4-bit	–	bf16, 8, 4-bit	bf16, 8, 4-bit	bf16, 8, 4-bit
🤗 Uno adapter	–	–	–	Uno	–	Uno

Pre-training Mid-training Math RL Code RL Other RL experts Merge SFT MOPD Anti-doom-loop

Each tick is one checkpoint, in training order, linked to its revision on Hugging Face. Load any revision with `revision="<tag>"`. Stages follow Figure 1. Quantized formats are covered in Section 10 and the Uno adapters in Section 9.

Training data

Dataset	Contents	Subsets	Size	Section
🤗 `TxT360-v2`	Web and question-answering text	3	5.29 TB	§4.1
🤗 `Math-Reasoning`	Math solutions, rewrites, dialogue	5	4.46 TB	§4.2
🤗 `Code-Reasoning`	Code reasoning and task synthesis	7	3.28 TB	§4.3
🤗 `SFT-Reasoning`	Instruction following, SFT-style data	2	159 GB	§4.4
🤗 `Pretrain-Behaviors`	Reasoning, planning, games, rewrites	7	8.37 TB	§4.5

Mid- and post-training data can be regenerated with the released pipelines listed under Code below, each documented in §5.

Code

Stage	Repositories, with the section that describes each
Pre-training data	`pretraining-data-toolkit` §4, `search360` §4.5
Mid- and post-training data	`PRism-curator` §5.2, `PRism-annotator` §5.3, `PRism-synthesis` §5.5, `file-manipulation-pipeline` §5.9, `task-factory` §5.12.1, `LCQA` §5.13, `advisor` §5.16.4, `process_entry` §5.17
Pre-, mid-training, SFT	`xllm` §3.2.2, `xattn` §3.2.2, `xbridges` §3.2
Post-training	`RL360` §6.1, `ISO-RAM weight merge` §6.4
Evaluation and inference	`Eval360` §7.1, `uno` §9, `llama.cpp` §10

Hugging Face	`huggingface.co/collections/IFM/k2-horizon`, and the `-quantization`, `-gguf` and `-mlx` collections
Training logs	`forge.coreweave.com/wandb/llm360`
Code	`github.com/ifm-ai`
Licenses	Models and datasets are released under Apache 2.0, except TxT360-v2 (CC BY 4.0).

Contents

1 Introduction **5**
- 1.1 Model Development Lifecycle . . . 6

2 Architecture **8**
- 2.1 Grouped RMSNorm . . . 9
- 2.2 MoVA: Mixture-of-Value Attention . . . 9
- 2.3 Softplus Activation in Attention Gate . . . 11

3 Pre-Training and Mid-Training **12**
- 3.1 Optimizer Recipe . . . 12
- 3.2 Infrastructure . . . 15
- 3.3 The Training Runs . . . 17
- 3.4 Base Model Evaluation . . . 22

4 Pre-Training Data **26**
- 4.1 Web data . . . 29
- 4.2 Math reasoning . . . 31
- 4.3 Code reasoning . . . 34
- 4.4 SFT reasoning . . . 34
- 4.5 Pre-training Behaviors . . . 36
- 4.6 Public sources . . . 37
- 4.7 Multilingual data . . . 38
- 4.8 Benchmark decontamination . . . 40
- 4.9 Data mix . . . 40
- 4.10 Tokenizer . . . 43

5 Mid- and Post-Training data **44**
- 5.1 PRism Data Processing . . . 45
- 5.2 PRism Data Curation . . . 45
- 5.3 PRism Annotation . . . 46
- 5.4 Developer thoughts synthesis . . . 46
- 5.5 PRism task synthesis . . . 47
- 5.6 PRism RL task synthesis . . . 48
- 5.7 Subagent delegation . . . 50
- 5.8 SWE Efficiency . . . 51
- 5.9 File Manipulation . . . 52
- 5.10 Search . . . 55
- 5.11 Tool use . . . 56
- 5.12 AI for Science . . . 59
- 5.13 Long Context Document Understanding . . . 72
- 5.14 Abstention . . . 75
- 5.15 Browser Use . . . 77
- 5.16 Beyond Teacher Imitation . . . 77
- 5.17 Programmatic quality and format controls . . . 83
- 5.18 Public Data Collection . . . 85
- 5.19 Chat template . . . 88

6 Post-Training **89**
6.1 RL Infrastructure . . . 90
6.2 RL algorithm . . . 93
6.3 Training Domain Experts for K2-Horizon-375B-A23B . . . 95
6.4 Model Merging with ISO-RAM . . . 98
6.5 Post-Training Small Models . . . 101
6.6 Supervised fine-tuning (SFT) . . . 101

7 Post-Training Evaluation **102**
7.1 Evaluation Setup . . . 102
7.2 375B-A23B Results . . . 104
7.3 Mid-size Results (36B-A4B and 32B) . . . 105
7.4 Small-size Results (7B, 3.7B, and 0.9B) . . . 106
7.5 Agentic Behavior across Training Stages . . . 107
7.6 Reward Hacking Audit . . . 110

8 Math Research Review Evaluation **111**
8.1 The Benchmark . . . 112
8.2 Results . . . 113
8.3 What Remains Hard . . . 114

9 Uno Diffusion: Efficient and Lossless Diffusion Adapters **116**
9.1 AR and Diffusion Pathways . . . 117
9.2 Diffusion Distillation . . . 118
9.3 Inference with Ψ-Spec Samplers . . . 119
9.4 Experiments . . . 119

10 Inference and Deployment **122**

11 Safety Evaluation **125**
11.1 Chat Safety Evaluation . . . 126
11.2 Agentic Safety Evaluation . . . 129
11.3 Discussion and Conclusion . . . 133

Contributors **135**

1 Introduction

We present `K2-Horizon`, a fleet of six fully open language models designed for agentic tasks. While many capable open-weight models are available, the data, intermediate states, and decisions behind them are rarely released. `K2-Horizon` succeeds K2-V2 (K2 Team, Institute of Foundation Models, 2025) in the LLM360 family. It extends that fully open approach from a single base model to the complete development lifecycle of six models: pre-training, mid-training, post-training, and deployment. The family ranges from a 0.9B dense model for on-device use to a 375B mixture-of-experts flagship that activates 23B parameters per token, and achieves competitive performance on general-purpose reasoning and agentic tasks, such as math and science, software engineering, deep research, file manipulation, customer service, and tool calling.

For all six models, we release the final weights, intermediate checkpoints from every training stage, quantized builds, training logs, the training data, and the code used throughout the training and evaluation pipeline. This code includes our training framework (xLLM), RL stack (RL360), and evaluation harness (Eval360). All released artifacts are documented in the Released Artifacts section. Four of the models (3.7B, 7B, 32B, and 36B-A4B) were pre-trained on the same 22T-token corpus with the same recipe, including two pairs trained on identical data batches. The release provides a complete, transparent recipe for reproducing frontier models at various sizes, allows controlled comparisons across scale and between dense and sparse architectures, and supports studies of how each ingredient influences the final model.

The family covers the main deployment settings (Table 1). The 0.9B targets phones and other memory-constrained devices. The 3.7B and 7B models are designed for local assistants and coding tools on consumer hardware, and serve as accessible platforms for fine-tuning and reinforcement-learning research. The 32B and 36B-A4B target self-hosted technical and agentic assistants, and the 375B-A23B targets multi-GPU server deployment.

When compared against four open-weight models released around the same time (GLM-5.2, MiniMax M3, Nemotron 3 Ultra, and Inkling), our 375B-A23B is smaller than most of them in both active and total parameters, yet ranks first or second on most agentic and coding benchmarks (see the overview figure on the first page). It leads all four on Toolathlon-Verified (65.3), where it also surpasses GPT-5.6 Terra and comes close to GPT-5.6 Luna. It also leads them on repository-level code understanding without internet access (SWE-Atlas-QnA, 48.4), and it exceeds both GPT-5.6 variants on τ^3-Banking. Its main gap is expert-level scientific reasoning (HLE and CritPt; §7.2). The 36B-A4B matches or stays within a few points of our own dense 32B (seven times its active parameters), trained on the same data, on every benchmark (§7.3). Among small models, the 7B outperforms similarly sized open-weight models on most benchmarks, and both the 7B and 3.7B lead SWE-bench Verified by wide margins (§7.4).

Our main contributions are:

- **Architecture. Grouped RMSNorm** across the family, and **Mixture-of-Value Attention (MoVA)** with a **softplus attention gate** in the 36B-A4B (§2).
- **Pre-training and mid-training.** A production-scale test of the scaling-collapse approach of Bergsma et al. (2026): four models that span dense and sparse architectures and nearly an order of magnitude in size, trained on 22T tokens with hyperparameters scaled together, follow nearly identical normalized loss curves. Pre-training, mid-training, and context extension all run on xLLM (§3.2.2), our open-source training infrastructure, which supports both dense and MoE models at production scale (§3, §3.3.2).
- **Data.** About 10T synthetic pre-training tokens in a web-heavy mix chosen by benchmark scores predicted at production scale (§4), and agentic trajectories introduced from the start of context extension rather than only in post-training (§5).

- **Post-training.** For the 375B-A23B, reinforcement learning of separate domain experts for search, knowledge work, tool use, scientific reasoning, and instruction following with abstention, merged into one model by weight merging and refined with targeted supervised fine-tuning (§6.3, §6.4); the other models follow adapted versions of this pipeline (§1.1).
- **Inference.** Uno, lossless diffusion adapters distilled from each model that speed up decoding by up to 2.2× without a separate draft model (§9, §10).

§1.1 walks through the development lifecycle and serves as a guide to the rest of the report.

1.1 Model Development Lifecycle

Developing a large language model involves a sequence of training stages, each starting from the checkpoint produced by the previous one. We describe the sequence for the 375B flagship; the other models follow it with minor differences noted in later sections. Figure 1 shows the training timeline for all six models, including pre-training and mid-training for the 32B and 36B-A4B models.

Pre-training. Pre-training gives the model most of its knowledge and its general language ability: starting from initialization, the model is trained to predict the next token on 8k-token sequences of web text, code, mathematics, scientific literature, and multilingual data, about 10T tokens of which are synthetic (§4). This stage accounts for most of the compute, so its key decisions are made in advance through small-scale experiments. Learning rates follow a scaling rule fitted across model sizes (§3.1), and the data mix is chosen by its benchmark scores predicted at the production scale rather than measured in small ablations, which led us to a web-heavy mix (§4.9). The flagship is trained on 15.1T tokens in two phases, the second with an updated data mix, while the dense and MoVA models see 22T tokens. All runs except the 0.9B hold a constant learning rate to the end, and the decay is instead realized by averaging the final checkpoints before mid-training.

Mid-training. Mid-training continues the same objective on about 1.9T tokens of more carefully curated data in four stages, extending the context window from 32k tokens in stage 1 to 128k in stage 2 and 512k in stages 3 and 4, with the RoPE base raised alongside (§3.1.3). Most of the budget goes to stage 1, during which the learning rate also decays to 10% of its peak. The data shift toward the capabilities the final model needs: long documents, reasoning, coding, and instruction following, together with agent trajectories in our chat template. We introduce agentic data as soon as context extension begins rather than reserving it for post-training (§5).

Post-training. Post-training sharpens the model on selected domains, especially long-horizon agentic tasks. This is where our recipes differ most across model sizes, as we adjust the recipe to fit the capability of each model. For our 375B flagship model, we choose the final mid-training checkpoint as the starting point. Based on it, a separate RL expert is trained for each of five domains: search, knowledge work, tool use, scientific reasoning, and instruction following with abstention. In each, the model acts in an environment such as a sandbox, a search engine, or a simulated user, a verifier scores the outcome, and the model is updated to favor successful behavior (§6.3). The experts are then merged into a single model by weight merging, which takes a few minutes and retains most of each expert's gains (§6.4). Evaluating the merged model reveals the remaining gaps, such as unfamiliar agent harness formats, and a final round of supervised fine-tuning (SFT) on demonstrations targets them.

Evaluation. Evaluation runs throughout the lifecycle, both to track progress and to catch regressions, and its findings feed back into later data and training stages (Sections 3.4 and 7). Finally, the model is optimized for efficient inference (§10) and checked for safety (§11).

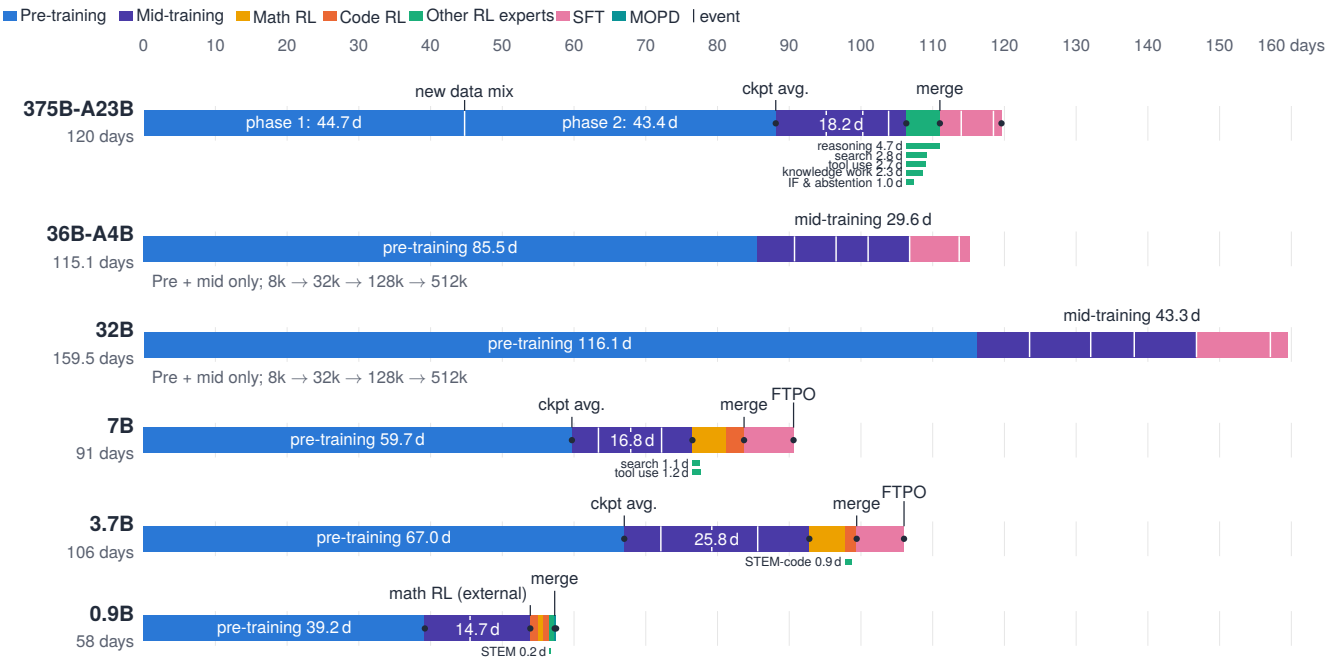


Figure 1: Training time of all six `K2-Horizon` models, measured from W&B run logs and drawn on a shared axis of training days. The 32B and 36B-A4B rows show pre-training, mid-training and SFT only; each includes four mid-training stages plus an SFT with final decay. Gaps between stages (data preparation, ablations, queueing) are excluded. Pre-training, mid-training and SFT durations are wall-clock run times, with mid-training stages separated by white lines; RL durations sum the logged rollout-step times. Thin bars below a row are RL experts trained in parallel, so each RL block spans its longest serial path. The 375B post-training takes 13.3 days (RL 4.7, SFT 8.6 in two phases: a constant-learning-rate phase, run in two segments, then a linear decay); the 7B and 3.7B take 14.1 and 13.2 days, and the 0.9B 3.6 days. Ticks mark events of negligible duration (checkpoint averaging, weight merging) or without a logged duration (FTPO; the first 0.9B math RL stage, delivered as a checkpoint).

Smaller models. The bottom three rows of Figure 1 show the lifecycle of the three smallest models. The 3.7B and 7B models follow the flagship most closely. They are pre-trained in a single phase on about 22T tokens with the dense-model data mix and go through the same four mid-training stages (Tables 2 and 3). Post-training differs in three ways. The coding expert is initialized from a math-trained checkpoint, which worked better than starting from the mid-trained base. SFT follows the flagship recipe but uses 250B tokens instead of 330B. Finally, because extensive long-horizon training makes small models prone to repetitive loops, we follow SFT with a brief stage of Final Token Preference Optimization (FTPO), trained on preference pairs mined from the model's own looping outputs. To address this, we follow SFT with a brief stage of Final Token Preference Optimization (FTPO), trained on preference pairs mined from the model's own looping outputs.

The 0.9B model, designed for memory- and compute-constrained deployment, differs the most. Its pre-training runs on 5.03T tokens with a cosine learning-rate decay. Mid-training is shorter: after a brief context-extension phase, it runs two stages at 40k and 128k tokens of context for about 0.6T tokens, each ending in a cosine decay to 1% of its peak. The first stage uses 40k rather than 32k because the model's smaller 64k vocabulary tokenizes the same text into more tokens. In post-training, applying the math RL recipe to a late mid-training checkpoint caused entropy collapse, so we alternate math and coding RL in a Math–Code–Math–Code sequence. Instruction-following and STEM experts are then trained in parallel from the last coding checkpoint, merged by task arithmetic, and refined with multi-teacher on-policy distillation (MOPD) instead of SFT.

	0.9B	3.7B	7B	32B	36B-A4B	375B-A23B
Parameter counts and vocabulary						
Total parameters (B)	1.078	5.058	8.999	34.779	37.445	379.167
Backbone parameters (B)	0.881	3.775	6.946	32.213	36.162	376.087
Active backbone parameters/token (B)	0.881	3.775	6.946	32.213	4.665	23.590
Vocabulary size	64,256	250,624	250,624	250,624	250,624	250,624
Tied input/output embeddings	No	No	No	No	No	No
Backbone configuration						
Architecture	Dense	Dense	Dense	Dense	MoE+MoVA	MoE
Layers	28	36	36	64	48	61
Hidden dimension	1,536	2,560	4,096	5,120	2,560	6,144
Query heads	32	32	32	64	32	48
KV heads	8	8	8	8	8	8
Head dimension	64	128	128	128	128	128
RoPE dimension per head	64	128	128	128	128	64
Dense FFN intermediate dimension	5,120	10,240	12,288	26,624	6,144	16,384
Normalization groups	1	2	4	4	2	1
Pre-training context length	8,192	8,192	8,192	8,192	8,192	8,192
Pre-training RoPE base	500,000	500,000	500,000	500,000	500,000	500,000
Expert configuration						
Initial dense FFN layers	All	All	All	All	3	3
Routed FFN experts per layer	–	–	–	–	100	192
Active routed FFN experts per token	–	–	–	–	8	8
Shared FFN experts per layer	–	–	–	–	1	1
FFN expert intermediate dimension	–	–	–	–	768	1,792
Value experts (total/active)	–	–	–	–	64/4	–

Table 1: Architecture configurations of the K2-Horizon family. All models use SwiGLU feed-forward networks. Backbone counts exclude the input embedding and output projection matrices; total counts include both. Active backbone counts include shared components and selected experts, accounting for both feed-forward and value-expert routing in MoVA. For sparse models, the dense FFN dimension applies to the initial dense layers; subsequent layers use the listed expert configuration. FFN denotes a feed-forward network. Active routed-expert counts exclude the shared expert. Parameter counts are rounded to three decimal places; model names denote nominal size classes.

2 Architecture

The `K2-Horizon` series retains the established Transformer (Vaswani et al., 2017) architecture, including GQA (Ainslie et al., 2023), rotary positional embedding (RoPE; Su et al. 2021), SwiGLU (Shazeer, 2020), and sparse Mixture-of-Experts (MoE) (Shazeer et al., 2017; Lepikhin et al., 2021; Fedus et al., 2022).

Table 1 summarizes the configurations. Training budgets and optimization are covered in §3.1. The 0.9B model targets memory-constrained mobile deployment. It uses a 64,256-token vocabulary, compared with 250,624 tokens for the larger models. For the 3.7B, 7B, and 32B dense models, we adopt the depth, hidden width, and attention-head configurations of Qwen3-4B, Qwen3-8B, and Qwen3-32B, respectively, as established architectural baselines (Yang et al., 2025).

The two MoE models (36B-A4B and 375B-A23B) use fine-grained sparse mixture-of-experts layers with routed and shared experts, following DeepSeekMoE (Dai et al., 2024). Both keep dense feed-forward networks in the first three layers.

Beyond the commonly used modules, we also introduce three components to improve capability and training stability: *grouped RMSNorm*, *mixture-of-value attention (MoVA)*, and a *softplus attention gate*.

2.1 Grouped RMSNorm

To facilitate the description of the new techniques, we use $\boldsymbol{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \in \mathbb{R}^{n \times d}$ and $\boldsymbol{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n\} \in \mathbb{R}^{n \times d}$ to denote the input and output sequences with length n, and assume the representations of the input and output sequences have the same dimension d.

Root Mean Square Layer Normalization. The root mean square layer normalization mechanism (RMSNorm) (Zhang and Sennrich, 2019) simplifies the original layer normalization (LayerNorm) (Ba et al., 2016) by dropping the re-centering (mean subtraction) and bias terms, and instead individually re-scaling each input vector $\mathbf{x}_t, t \in \{1, \ldots, n\}$ with its root mean square statistic:

$$v_t = \sqrt{\frac{1}{d}\sum_{i=1}^{d} x_{t,i}^2 + \epsilon} \tag{1}$$

$$y_{t,j} = \frac{x_{t,j}}{v_t}, \forall j \in \{1, \ldots, d\} \tag{2}$$

where we omit the commonly used scaling parameter for simplicity. RMSNorm achieves performance comparable to LayerNorm at a lower computational cost, and has been adopted as the default normalization mechanism in a range of frontier large language models (Touvron et al., 2023; Meta AI, 2024; Qwen Team et al., 2025; DeepSeek-AI et al., 2024). Theoretically, each output element $y_{t,j}$ lies in the range $(-\sqrt{d}, \sqrt{d})$. While a typical element has magnitude around 1, this bound grows with model width d (e.g., $\sqrt{d} \approx 90$ for $d = 8192$). As a result, individual outputs of RMSNorm can become disproportionately large in wide models, which may lead to training instability in extreme cases.

Grouped RMSNorm. We propose Grouped RMSNorm and apply it to the 3.7B, 7B, 32B, and 36B-A4B models. Formally, Grouped RMSNorm first partitions the d feature dimensions of each $\mathbf{x}_t$ into k groups, with $d_g = d/k$ elements per group. The root mean square (RMS) statistic (3) is then computed independently within each group:

$$v_{t,g} = \sqrt{\frac{1}{d_g}\sum_{i \in \mathcal{G}_g} x_{t,i}^2 + \epsilon}, \quad g = 1, \ldots, k. \tag{3}$$

By controlling the group dimension d_g through the number of groups k, particularly by increasing k as the model dimension grows, Grouped RMSNorm provides an explicit upper bound on the magnitude of each normalized output element, restricting it to the range $(-\sqrt{d_g}, \sqrt{d_g})$. Empirical results demonstrate that Grouped RMSNorm substantially improves training stability across a wide range of model scales in the `K2-Horizon` series.

2.2 MoVA: Mixture-of-Value Attention

Self-Attention Mechanism. The standard self-attention mechanism is a function:

$$\boldsymbol{Y} = \mathrm{Attn}(\boldsymbol{X}) = f\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\tau(\boldsymbol{X})}\right)\boldsymbol{V} \tag{4}$$

where $\mathrm{Attn} : \mathbb{R}^{n \times d} \to \mathbb{R}^{n \times v}$ is the self-attention function. $\boldsymbol{Q} = \boldsymbol{X}W_q$, $\boldsymbol{K} = \boldsymbol{X}W_k$, and $\boldsymbol{V} = \boldsymbol{X}W_v$ are the sequences of queries, keys and values, with learnable parameters. $f(\cdot)$ is an attention function, e.g., the softmax function $f_{\mathrm{softmax}}(\cdot)$ (Bahdanau et al., 2015), the squared ReLU function $f_{\mathrm{relu}^2}(\cdot)$ (Hua et al., 2022) or the Laplace function $f_{\mathrm{laplace}}(\cdot)$ (Ma et al., 2023). $\tau(\boldsymbol{X})$ is a scaling term, which is commonly set to $\tau(\boldsymbol{X}) = \sqrt{d_k}$ for $f_{\mathrm{softmax}}(\cdot)$, or $\tau(\boldsymbol{X}) = n$ for $f_{\mathrm{relu}^2}(\cdot)$ and $f_{\mathrm{laplace}}(\cdot)$. The commonly used multi-head variant of attention performs the attention function h times in parallel.

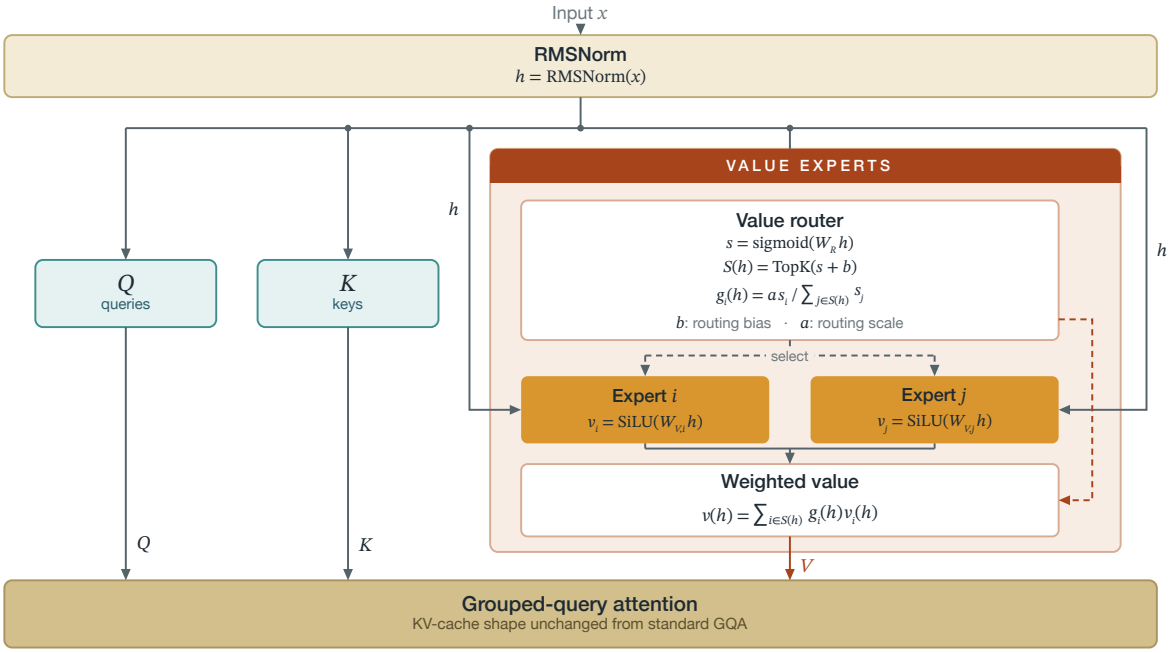


Figure 2: Mixture-of-Value Attention in the 36B model. A router selects 4 of 64 nonlinear value experts and combines their weighted outputs into the values used by attention. Each expert produces values for all eight KV heads jointly. Since only the combined values are cached, the KV-cache size is unchanged. Solid arrows carry representations, and dashed arrows carry routing decisions or weights. Query/key positional embeddings, attention-output gating, the output projection, and the residual connection are omitted for clarity.

Mixture-of-Experts (MoE). MoE models extend standard transformer architectures by replacing the feed-forward network (FFN) with E experts and a routing function that sparsely selects a subset for each token (Shazeer et al., 2017; Fedus et al., 2022). Given token representation x, the router assigns probabilities over experts and activates only the top-k:

$$\mathrm{MoE}(x) = \sum_{i \in \mathcal{K}(x)} p_i(x) \cdot E_i(x),$$

where $\mathcal{K}(x) \subset \{1, \ldots, E\}$ with $|\mathcal{K}(x)| = k$ denotes the selected expert indices and $p_i(x)$ are the normalized routing weights. Since $k \ll E$ in practice, per-token floating-point operations (FLOPs) scale with k, enabling parameter count to grow quasi-independently of computation. In practice, each expert E_i is instantiated as a two-layer MLP network with SwiGLU activation (Shazeer, 2020).

Mixture-of-Value Attention (MoVA). In `K2-Horizon`-MoVA-36B-A4B, we propose mixture-of-value attention (MoVA), which incorporates MoE-style sparsity into the computation of value vectors in multi-head attention. Formally, MoVA replaces the single value projection $\boldsymbol{v}_t = \boldsymbol{x}_t W_v$ in the standard attention mechanism with a routing function that sparsely selects a subset of V value projections:

$$V_i(\boldsymbol{x}_t) = \boldsymbol{x}_t W_v^{(i)} \tag{5}$$

$$\mathrm{V}(\boldsymbol{x}_t) = \sum_{i \in \mathcal{V}(\boldsymbol{x}_t)} p_i(\boldsymbol{x}_t) \cdot \phi_{\mathrm{silu}}\left(V_i(\boldsymbol{x}_t)\right), \tag{6}$$

where $\mathcal{V}(\boldsymbol{x}_t) \subset \{1, \ldots, V\}$ with $|\mathcal{V}(\boldsymbol{x}_t)| = k$ denotes the selected expert indices and $p_i(\boldsymbol{x}_t)$ are the normalized routing weights. Unlike conventional MoE layers, where each expert E_i typically consists of a two-layer

MLP and thus introduces nonlinearity, an individual value projection in (5) is purely linear. To introduce nonlinearity into the resulting value vector $\mathrm{V}(\boldsymbol{x}_t)$ in (6), we therefore apply a SiLU activation function to the output of each value projection. Without this activation, the mixture reduces to $\boldsymbol{x}_t\big(\sum_i p_i(\boldsymbol{x}_t)W_v^{(i)}\big)$, an input-dependent linear map whose only nonlinearity comes from the routing weights.

MoVA opens a second axis for scaling sparsity in an LLM, beyond MoE in the FFN module. Importantly, MoVA enjoys the following advantages:

- **Compatibility with efficient attention.** MoVA only modifies how each value vector is computed from its own token, leaving the attention computation itself unchanged. It is therefore directly compatible with FlashAttention, GQA, and sparse attention.
- **No additional KV cache.** Each $\boldsymbol{v}_t$ is fully determined once token t is processed, so only the mixed value vector is cached, with the same size as in standard GQA.

`K2-Horizon`-MoVA-36B-A4B combines MoVA with MoE FFN layers and activates about 4.6B parameters per token. Empirically, its pre-training loss closely tracks that of our 32B dense model, which was trained on the same data with identical settings, with the dense model maintaining a modest lead. Downstream evaluations reflect this: at 1.1M steps, MoVA trails the dense model by only 1.2 to 3.7 points on MMLU, GSM8K, HellaSwag, and HumanEval, despite activating roughly one-seventh as many parameters (see §3.4 and full evaluation logs at W&B[1]).

2.3 Softplus Activation in Attention Gate

In the `K2-Horizon`-MoVA-36B-A4B model, we use gated attention with a softplus gate activation. The gated attention mechanism has been extensively studied and widely used in frontier models (Hua et al., 2022; Ma et al., 2024; Qiu et al., 2025; Ma et al., 2026, 2023). Notably, Ma et al. (2023) show that gating substantially increases the expressiveness of attention: single-head gated attention is as expressive as multi-head attention.

Formally, the gated attention mechanism applies a gating vector to the attention output:

$$\boldsymbol{\gamma} = \phi(\boldsymbol{X}W_\gamma + b_\gamma) \tag{7}$$

$$\boldsymbol{Y} = \left(f\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\tau(\boldsymbol{X})}\right)\boldsymbol{V}\right) \odot \boldsymbol{\gamma}, \tag{8}$$

where $\odot$ denotes element-wise product, and $\phi(\cdot)$ is the activation function used to compute the attention gate. Two common choices for $\phi(\cdot)$ are sigmoid and SiLU, and Qiu et al. (2025) empirically show that sigmoid slightly outperforms SiLU. However, sigmoid is bounded in $(0, 1)$: at initialization it sits near $\mathrm{sigmoid}(0) = 0.5$, halving the attention output, and even at its maximum it can only recover the ungated output, never amplify it. This limits the expressiveness of the gate, since the result of Ma et al. (2023) requires the gate to represent the ratio between multi-head and single-head attention outputs, which can exceed 1. We therefore adopt the softplus function as the gate activation:

$$\mathrm{softplus}(x;\beta) = \frac{1}{\beta}\cdot\log\left(1+\exp(\beta\cdot x)\right), \tag{9}$$

where β is a scaling coefficient. Like sigmoid, softplus is strictly positive, so the gate acts as a pure element-wise scaling factor; unlike sigmoid, it is unbounded above, allowing the gate to both attenuate and amplify. Moreover, setting $\beta = \log 2$ yields $\mathrm{softplus}(0;\log 2) = 1$, so that at initialization, when the gate

[1] https://forge.coreweave.com/wandb/llm360/K2-Horizon-36B, https://forge.coreweave.com/wandb/llm360/K2-Horizon-32B

pre-activations are close to zero, the gate is approximately the identity and gated attention behaves like standard attention.

3 Pre-Training and Mid-Training

This section describes the pre-training and mid-training of the `K2-Horizon` models, including the training recipe, infrastructure, and challenges encountered during training. §3.1 presents the optimizer recipe, covering hyperparameter selection at small scale, the final pre-training settings for each model, and the mid-training and long-context schedule. §3.2 describes the training infrastructure, including the cluster, tooling for maintaining long-running jobs, the training framework, and the parallelism configuration. §3.3 details the flagship 375B model's two pre-training phases, the loss spikes and associated rollbacks, and the subsequent mid-training run.

3.1 Optimizer Recipe

Our hyperparameter-selection strategy builds on the dense-model approach used for K2-V2. We use small-scale experiments to guide learning-rate selection across model sizes and parameterize weight decay through the AdamW averaging timescale. We extend this approach to sparse models by investigating learning-rate transfer as the number of routed experts increases, while holding model width and the number of active experts fixed. These experiments inform the production recipes for our dense and sparse models.

3.1.1 Pre-Training Hyperparameter Selection

AdamW timescale. Following Wang and Aitchison (2025); Bergsma et al. (2025), we express the coupling between learning rate, weight decay, and training duration through the nominal AdamW timescale:

$$\tau_{\text{epoch}} = \frac{B}{\eta\lambda D} = \frac{1}{\eta\lambda T}, \tag{10}$$

where B is the global batch size in tokens, D is the training token budget, $T = D/B$ is the number of optimizer steps, η is the peak learning rate, and λ is the weight-decay coefficient. The timescale expresses how quickly weight decay discounts earlier parameter values, measured as a fraction of the training budget. Holding it fixed as learning rate or training length changes keeps the nominal accumulated decay comparable. Because the timescale is defined using the peak learning rate, it provides only a nominal measure; the accumulated decay also depends on the learning-rate schedule. We used small-scale experiments to inform timescale selection and the corresponding production weight-decay settings. For the 3.7B, 7B, 32B, and 36B-A4B, the production settings scale the timescale as $\tau_{\text{epoch}} \propto 1/\sqrt{\text{TPP}}$, where TPP is the number of training tokens per active backbone parameter: $\tau_{\text{epoch}}\sqrt{\text{TPP}}$ lies between 2.6 and 2.9 for these four models.

Learning-rate scaling for dense models. We swept learning rates across dense models at a fixed tokens-per-parameter ratio, varying model size and attention-head count. The best observed learning rates decreased approximately as the inverse square root of the number of parameters:

$$\eta_{\text{best}} \propto N^{-1/2}, \tag{11}$$

where N is the number of active backbone parameters. This is consistent with the width-dependent shift under standard parameterization shown in Yang et al. (2022, Figure 1, left) and the extrapolation strategy used for K2-V2 (K2 Team, Institute of Foundation Models, 2025). The learning rates of Table 2 follow the same rule from the 0.9B to the flagship 375B: $\eta\sqrt{N}$ lies between 6.8×10^{-4} and 8.6×10^{-4} for the active backbone counts of Table 1. We used this relationship to guide learning-rate choices across scales.

Learning-rate transfer across MoE sparsity. We additionally swept learning rates for six-layer MoE models. We increased the number of routed experts from 4 to 128 while holding model width (256), expert width (512), and active-expert count (2) fixed. Batch sizes and training horizons varied across expert counts. The comparison therefore reflects the study's training policy rather than holding all other training conditions constant.

Among completed runs, the best tested learning rates ranged from 3.16×10^{-3} to 10^{-2}, without a monotonic trend with expert count. We selected these rates using the final exponentially weighted moving average of training loss, with a 50-step half-life. The dense baseline and the sparsest tested configuration, with two of 128 routed experts active per token, both selected a learning rate of 7.5×10^{-3}. Although noisy, the results supported keeping a common starting learning rate as sparsity increases. Weak sensitivity to sparsity has also been observed under hypersphere optimization with HyperP (Ren et al., 2026), which uses a different optimization method.

Batch-size selection. We initially targeted a batch size, measured in tokens, close to that of DeepSeek-V3 (DeepSeek-AI et al., 2024), which was trained on a similar token budget (14.8T tokens versus our 15.1T): 8k sequences of 8k tokens (67M tokens; k = 1,024 throughout), compared with DeepSeek-V3's 15,360 sequences of 4k tokens (63M tokens). We tested this target on a small scale by scaling the batch down in proportion to the square root of the token budget, following the Power Lines framework (Bergsma et al., 2025), and compared that scaled-down batch with one half its size. For dense models with 61 layers and widths of 512, 640, and 768, trained at a fixed tokens-per-parameter ratio, the scaled-down batch sizes were 1.4M, 1.7M, and 2.1M tokens, respectively. We also evaluated batch sizes approximately half as large. For each model and batch-size setting, we selected the best-performing learning rate from nine candidates. Halving the batch size reduced the final training loss by 2.7%, 5.1%, and 3.4% at widths of 512, 640, and 768, respectively. These losses were smoothed using an exponentially weighted moving average with a half-life of 50 steps. The validation gain was smaller, 1.1% and 0.4% at widths 512 and 768, and reversed at width 640, where the loss was 0.1% higher. Based on these results, we halved the target for the 375B model to 4k sequences of 8k tokens (34M tokens; Table 2).

3.1.2 Final Pre-Training Recipe

We combined the small-scale results with the training budgets and execution constraints of each model to select the final recipes. Table 2 summarizes the production optimizer settings and global batch sizes.

Model / phase	Batch (tokens)	Steps	Tokens (T)	Peak LR	Warmup steps	Weight decay
0.9B	8,388,608	600,000	5.03	8×10^{-4}	3,000	0.10
3.7B	19,922,944	1,100,000	21.92	4×10^{-4}	3,000	0.06
7B	19,922,944	1,100,000	21.92	3×10^{-4}	3,000	0.06
32B	19,922,944	1,100,000	21.92	1.5×10^{-4}	3,000	0.06
36B-A4B	19,922,944	1,100,000	21.92	4×10^{-4}	3,000	0.06
375B-A23B / phase 1	33,554,432	210,000	7.05	1.4×10^{-4}	4,500	0.073
375B-A23B / phase 2	33,554,432	240,000	8.05	1.4×10^{-4}	1,000	0.073

Table 2: Pre-training recipe. All models use AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.95$, gradient-norm clipping at 1.0, and an 8k-token sequence length. Batch sizes are global token counts. Steps denote checkpoint endpoints, with phase-local counts for the 375B model; tokens are batch size times steps, rounded to two decimal places in trillions. The 375B pre-training data mixture was updated after phase 1 (§3.3.1), with a 1,000-step re-warmup at the phase boundary to help the model stabilize on the new data. The two 375B phases sum to approximately 15.10T tokens at the listed endpoints. Warmup values are configured optimizer-step counts, including the phase-2 re-warmup for 375B.

All listed configurations start warmup at 10^{-4} times the peak learning rate. The 0.9B model uses cosine decay to 10% of its peak rate. The 3.7B, 7B, 32B, and 36B models retain the peak learning rate after

Model	Stages	Sequence Length	Batch (tokens)	RoPE Base	Steps	Tokens (T)	Peak LR	Warmup steps	Weight decay
0.9B†	Stage 1	40k	5,242,880	1 million	75,000	0.393	2×10^{-4}	200	0.1
	Stage 2	128k	4,194,304	1 million	47,684	0.2	5×10^{-5}	200	0.06
3.7B	Stage 1	32k	19,922,944	1 million	55,000	1.1	4×10^{-4}	1,250	0.06
	Stage 2	128k		10 million	25,000	0.498	4×10^{-5}	500	
	Stage 3	512k		10 million	5,500	0.11	4×10^{-5}	200	
	Stage 4	512k		10 million	10,000	0.199	4×10^{-5}	200	
7B	Stage 1	32k	19,922,944	1 million	55,000	1.1	3×10^{-4}	1,250	0.06
	Stage 2	128k		10 million	25,000	0.498	3×10^{-5}	500	
	Stage 3	512k		10 million	5,500	0.11	3×10^{-5}	200	
	Stage 4	512k		10 million	10,000	0.199	3×10^{-5}	200	
32B	Stage 1	32k	19,922,944	1 million	55,000	1.1	1.5×10^{-4}	1,250	0.06
	Stage 2	128k		10 million	25,000	0.498	1.5×10^{-5}	500	
	Stage 3	512k		10 million	5,500	0.11	1.5×10^{-5}	200	
	Stage 4	512k		10 million	10,000	0.199	1.5×10^{-5}	200	
36B-A4B	Stage 1	32k	19,922,944	1 million	55,000	1.1	4×10^{-4}	1,250	0.06
	Stage 2	128k		10 million	25,000	0.498	4×10^{-5}	500	
	Stage 3	512k		10 million	5,500	0.11	4×10^{-5}	200	
	Stage 4	512k		10 million	10,000	0.199	4×10^{-5}	200	
375B-A23B	Stage 1	32k	33,554,432	1 million	32,500	1.09	1.4×10^{-4}	1,250	0.073
	Stage 2	128k		10 million	15,000	0.503	1.4×10^{-5}	300	
	Stage 3	512k		10 million	3,500	0.117	1.4×10^{-5}	150	
	Stage 4	512k		10 million	6,000	0.201	1.4×10^{-5}	150	

Table 3: Mid-training recipe. All models use AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.95$, and gradient-norm clipping at 1.0. For each stage, the table lists the context length, the global batch size in tokens, the number of optimizer steps, the resulting token count in trillions, the peak learning rate, the number of warmup steps, and the weight decay. Weight decay is set once per model and shared across its stages, except the 0.9B model. Mid-training starts from the average of the final pre-training checkpoints: four spaced 1,500 steps apart for the 375B model (phase-2 steps 235,500 to 240,000), four spaced 4,000 steps apart for the 3.7B and 7B models (steps 1,088,000 to 1,100,000), and six spaced 5,000 steps apart for the 32B and 36B models (steps 1,075,000 to 1,100,000); the 0.9B model uses no averaging. Sequence lengths use k = 1,024 tokens. †Before stage 1, the 0.9B model runs a brief context-extension phase of about 2,000 steps on separate data, which is not listed in the table.

warmup through the listed pre-training endpoints. The 375B model retains its peak learning rate after warmup within each listed phase. All pre-training runs except the 0.9B's therefore end at their peak learning rate, and the decay is realized instead by averaging the final checkpoints (Tian et al., 2026) (§3.1.3).

3.1.3 Mid-Training and Long-Context Recipe

Mid-training has three jobs: to realize the learning-rate decay that pre-training postponed, to extend the context window from 8k to 512k tokens, and to shift the data toward long documents, reasoning, and agentic trajectories (§5). It reuses each model's pre-training batch size, weight decay, and peak learning rate, so no new sweep is needed (Table 3).

Implicit decay via merging. Because pre-training ends at the peak learning rate (§3.1.2), mid-training starts from an average of the last few pre-training checkpoints rather than from a single one, which acts as an implicit decay (Tian et al., 2026). Stage 1 then re-warms the learning rate to the pre-training peak and decays it linearly to 10% of the peak. Stage 1 carries most of the mid-training budget, about 1.1T of roughly 1.9T tokens, so this is where the explicit decay happens.

Extending the context. Context then grows in three steps, from 32k tokens in stage 1 to 128k in stage 2 and 512k in stages 3 and 4, and these longer-context stages use far fewer tokens than stage 1. As in K2-V2 (K2 Team, Institute of Foundation Models, 2025), the learning rate stays at the 10% floor through stages 2–4 to keep optimization stable as the context grows, and the RoPE base rises with the context

length, from 500,000 in pre-training to 1 million in stage 1 and 10 million from stage 2 onward, so that rotary positions stay distinguishable over longer distances (Xiong et al., 2023).

The 0.9B model. The 0.9B differs because its pre-training already ended with a cosine decay, so it skips checkpoint averaging. After a brief context-extension phase of about 2,000 steps on separate data, it runs two stages at 40k and 128k tokens of context, each with a short warmup and a cosine decay to 1% of the stage's peak. It stops at 128k to keep its memory footprint within the budget of the phones and other memory-constrained devices it targets. Its first stage uses 40k rather than 32k tokens because its smaller vocabulary turns the same text into more tokens, so data prepared for a 32k context would otherwise be truncated.

3.2 Infrastructure

This section describes the system that ran the training. §3.2.1 covers the cluster, its storage and checkpointing, the tooling that keeps jobs running through failures, and the loss monitor that restarts or rolls back a run when its loss misbehaves. §3.2.2 covers the xLLM training framework and the parallel layout chosen for each model.

3.2.1 Clusters, Storage, and Reliability

Compute environment. All models in the `K2-Horizon` family were trained on servers with the same node configuration as the K2-V2 cluster (K2 Team, Institute of Foundation Models, 2025, Sec. 3.4.1). Each node hosts eight NVIDIA H200 GPUs (141 GB HBM, 700 W TDP) and two Intel Xeon Platinum 8468 CPUs with approximately 1.6 TB of system memory. For inter-node communication, each node is equipped with eight 400 Gb/s InfiniBand NDR host channel adapters, one per GPU, for an aggregate of 3.2 Tb/s of injection bandwidth per node. All nodes are connected through a fully non-blocking fabric. Jobs are scheduled with Slurm. The fleet is shared with other workloads, and training runs are submitted under dedicated accounts and quality-of-service (QoS) levels that govern priority and preemption.

Storage and checkpointing. Checkpoints are distributed and sharded: all ranks cooperatively save and reload model shards, while optimizer shards are written once per non-replicated shard by designated ranks. This enables parallel checkpoint I/O and resumption. We write a checkpoint about every 250 training steps to make potential rollbacks convenient. To bound storage use, older checkpoints are rotated out over time, while a subset is retained (milestone checkpoints, usually every 5000 steps) for evaluation. For durability beyond the cluster filesystem, a scheduled upload job copies completed checkpoints to Amazon S3.

Reliability tooling. Runs are managed through an in-house submitter that wraps Slurm and exposes three lifecycle operations. Submit launches a run, terminate stops it cleanly, and recover relaunches it from the most recent valid checkpoint. At the scale of our runs, hardware faults and transient job failures are expected. We therefore treat automatic restarts as routine operation and distinguish them from interventions, meaning deliberate human changes to a run's configuration, data, or code. §3.3.2 reports the spikes detected during the run and the rollbacks they led to.

Loss monitor. Alongside the fault-driven restarts above, a separate monitor watches the training loss itself. Every 30 minutes it reads the most recent 35 minutes of per-step loss and applies two tests. The restart test flags a window of 10 consecutive steps in which every loss exceeds 0.97 and the largest exceeds 1.2; requiring the whole window to stay high ignores isolated noisy steps, while the second condition requires a large excursion. A flag triggers a rollback, and the run is relaunched through the submitter from the last checkpoint on the 250-step grid at or before the first flagged step. The alert test is looser (windows of 5 steps, with thresholds 0.95 and 0.99) and only sends a notification. Both sets of thresholds are absolute loss values, so they encode the loss range of the run and must be retuned if it shifts. The monitor reads

the training loss only, and it has no guard against resuming repeatedly from the same checkpoint. §3.3.2 counts restarts and rollbacks by how far the run was rewound, whichever mechanism triggered them. The design follows the monitor used for K2-V2 (K2 Team, Institute of Foundation Models, 2025).

3.2.2 Training Infrastructure and Parallelism

Framework. All models are trained with xLLM, our in-house training framework, which we open source alongside this report. xLLM is designed to be an efficient, production-tested infrastructure that combines large-scale performance with research flexibility. Architectures, data mixtures, training stages, and objectives can be changed without rebuilding the surrounding stack. It covers the full model lifecycle: training, evaluation, checkpoint conversion, bridges to serving systems, and runtime kernels. xLLM was already the mid-training framework for K2-V2, though it was not named in that report (K2 Team, Institute of Foundation Models, 2025). For `K2-Horizon` we extended it to pre-training, so a single codebase now spans multiple training stages.

Parallelism trade-offs. Each parallelism axis trades memory savings for communication cost. We summarize the axes we use, following K2 Team, Institute of Foundation Models (2025, Sec. 3.4.2).

Data parallelism (DP) splits each global batch across workers, and each worker processes its own shard of the batch. In its plain form, every worker holds a full model replica, and gradients are synchronized with an all-reduce after each step (Li et al., 2020). This scales throughput almost linearly but saves no memory. xLLM instead implements DP with PyTorch FSDP in full-shard mode, which is equivalent to ZeRO-3 (Rajbhandari et al., 2020; Zhao et al., 2023). Parameters, gradients, and optimizer states are sharded across the DP group of each tensor-parallel rank. Parameters are all-gathered one layer at a time for compute and resharded after the forward pass. Gradients are reduce-scattered in fp32, not all-reduced. Each GPU therefore holds roughly a $1/(\mathrm{TP} \cdot \mathrm{DP})$ share of the model and optimizer state. The price is an extra parameter all-gather in both the forward and backward passes, and FSDP overlaps it with computation.

Tensor parallelism (TP) shards individual weight matrices by rows or columns, so each device computes a slice of every layer (Shoeybi et al., 2019). This cuts per-device parameter and activation memory, but it needs collectives inside every layer, in both the forward and backward passes. Only high-bandwidth intra-node links (NVLink) can absorb that traffic, so we restrict TP to 8-GPU groups within a node.

Expert parallelism (EP), which K2-V2 did not need, distributes the experts of a mixture-of-experts layer across devices. Tokens are routed to their assigned experts through all-to-all communication (Lepikhin et al., 2021; Fedus et al., 2022). In xLLM, EP and TP are tied: experts are sharded across the same 8-GPU group used for tensor parallelism. Both axes therefore share one intra-node communication domain, and the all-to-all traffic never crosses the InfiniBand fabric.

Context parallelism (CP) partitions activations along the sequence dimension, and its benefit grows with context length: activation memory scales with sequence length while parameter memory does not. xLLM has no separate sequence-parallel (SP) axis (Korthikanti et al., 2023). Between layers, tensor parallelism already shards activations along the hidden dimension. Our CP uses the all-gather formulation (Grattafiori et al., 2024). Each CP rank holds a contiguous chunk of the sequence. For attention, it all-gathers keys and values from the other ranks and keeps only the causal prefix its queries need. CP ranks share parameters, so FSDP switches to hybrid sharding: parameters are sharded across DP and replicated across CP. We use no CP during 8k pre-training. CP is added in mid-training as context grows, and it reaches CP=4 at 512k context for the 375B flagship.

Pipeline parallelism (PP) assigns contiguous groups of layers to different devices. xLLM has no pipeline axis. Pipelining introduces idle "bubbles" unless the per-device batch is large enough to fill the pipeline.

K2-V2 showed that, at its operating point, the bubble overhead outweighed the memory benefit (K2 Team, Institute of Foundation Models, 2025, Sec. 3.4.2), and the same reasoning applies at ours.

Chosen strategy. For the 375B-A23B, these heuristics yield a two-axis layout. Tied TP≡EP=8 runs within each node, and fully sharded (ZeRO-3) data parallelism spans all remaining devices. We use no context parallelism during 8k-context pre-training and defer it to the mid-training configuration. Master weights are kept in fp32 and sharded by FSDP. Forward and backward compute runs in bf16, and gradient reduction is performed in fp32. To save activation memory, we use selective recomputation: the gated-FFN intermediate activations are recomputed in the backward pass, while the output logits are kept. Attention uses FlashAttention. Smaller models reuse the same framework at smaller allocations (Table 4) and different recomputation controls to satisfy memory requirements.

		CP				
Model	TP/EP	Pre-training	Mid-1	Mid-2	Mid-3	Mid-4
0.9B	1	1	1	1	–	–
3.7B	1	1	1	2	8	8
7B	1	1	1	4	16	16
32B	2	1	1	4	16	16
36B-A4B	2	1	1	2	8	8
375B-A23B	8	1	1	1	4	4

Table 4: Pre-training and mid-training compute and parallelism setup. Mid-k denotes mid-training stage k. TP and EP are tied in our MoE and MoVA models.

3.3 The Training Runs

Loss spikes persist throughout training

- Spikes occur throughout the run, not only early. Counting episodes on erased branches, as K2-V2 does, we find that 43% to 67% of the spike episodes in the six runs that held a constant peak learning rate fall in the first 40% of the run, compared with the vast majority in K2-V2; only the 0.9B run, whose learning rate decays on a cosine schedule, is front-loaded (91%; Table 5). A likely reason is that these runs hold the peak learning rate for most of training, whereas the 0.9B run's decaying learning rate suppresses later spikes.
- Many of the flagship's spike episodes occurred on training branches that were later discarded when the run resumed from an earlier checkpoint, so they never entered the shipped model: 69% of the phase-1 episodes and 35% of the phase-2 episodes.
- Most spikes are brief, and the gradient norm gives little warning: it carries information only in the last ten steps before a spike.

This subsection describes how the training runs went, focusing on the 375B flagship and covering the smaller models more briefly. §3.3.1 describes the phase transition, which divided the flagship pre-training run into two regimes with separate step counters. §3.3.2 analyzes the loss spikes detected across both regimes, the restarts and rollbacks used to resolve them, and their cost in discarded training (Table 5). §3.3.3 covers the mid-training run. The loss monitor used on the flagship run is described in §3.2.1.

3.3.1 The Phase Transition

The flagship was pre-trained in two phases that share one architecture and one optimizer but differ in their data. Phase 1 contributed 210,000 steps, or 7.05T tokens, on the initial data mixture. Phase 2 continued

Chain	Steps	Episodes (accepted)	In first 40%	Episodes (erased)	Erased share	Rollbacks	Deepest rollback
0.9B	0.6M	65	91%	2	3%	3	6,005
3.7B	1.1M	225	46%	1	0.4%	1	453
7B	1.1M	403	43%	106	21%	43	22,124
32B	1.1M	100	52%	5	5%	13	2,747
36B-A4B	1.1M	30	67%	3	9%	5	2,072
375B phase 1	0.21M	83	49%	184	69%	29	4,217
375B phase 2	0.24M	71	62%	38	35%	11	3,761

Table 5: Spike episodes by training chain, counted at full cadence on the accepted trajectory and on branches later erased when the run resumed from a checkpoint. An episode runs from the first step at which the loss z-score exceeds 5 until it stays below 2 for three steps. The z-score is the loss's distance above its rolling median in units of the rolling median absolute deviation (Equation 12). The column "In first 40%" is the share of all episodes, accepted and erased, that start in the first 40% of the run's schedule. A rollback rewinds the run by more than the 250-step checkpoint interval, and the deepest rollback is in steps. Step counts are rounded.

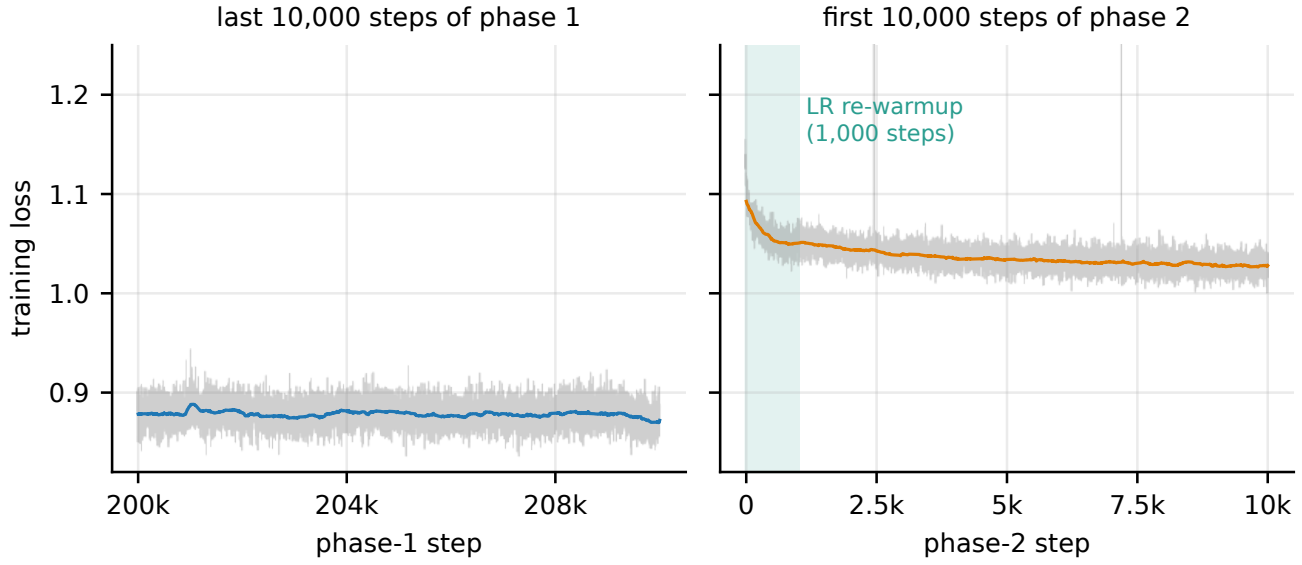


Figure 3: Training loss across the phase boundary. Left: the last 10,000 steps of phase 1. Right: the first 10,000 steps of phase 2, which restarts the step counter, discards the optimizer state and re-warms the learning rate over 1,000 steps (shaded). Grey: loss at every step; colored: running median over 201 steps. The two phases use different data, so their loss levels are not directly comparable.

from the phase-1 weights on an updated mixture (§4) for a further 240,000 steps, or 8.05T tokens, for a total of 450,000 steps, or 15.10T tokens. Each phase restarts its own step counter at one, so step numbers in this section and in Table 2 refer to the phase in which they occur.

Phase 2 started from model weights only: the optimizer state was discarded, and the learning rate was re-warmed linearly over 1,000 steps, from 10^{-4} times the peak back to the same peak of 1.4×10^{-4} (phase 1 used a 4,500-step warmup). The learning rate then stayed at that peak for the rest of the run, since neither phase used a decay (§3.1.2). The updated mixture raised the web share and moved weight from source code to code-reasoning data (§4.9); it replaced the phase-1 mixture.

Figure 3 shows the loss around the boundary. Over the last 2,000 steps of phase 1 the loss averages 0.88. Over the first 100 steps of phase 2 it averages 1.10, and it falls to 1.01 by step 50,000 and to 0.98 over the final 1,000 steps.

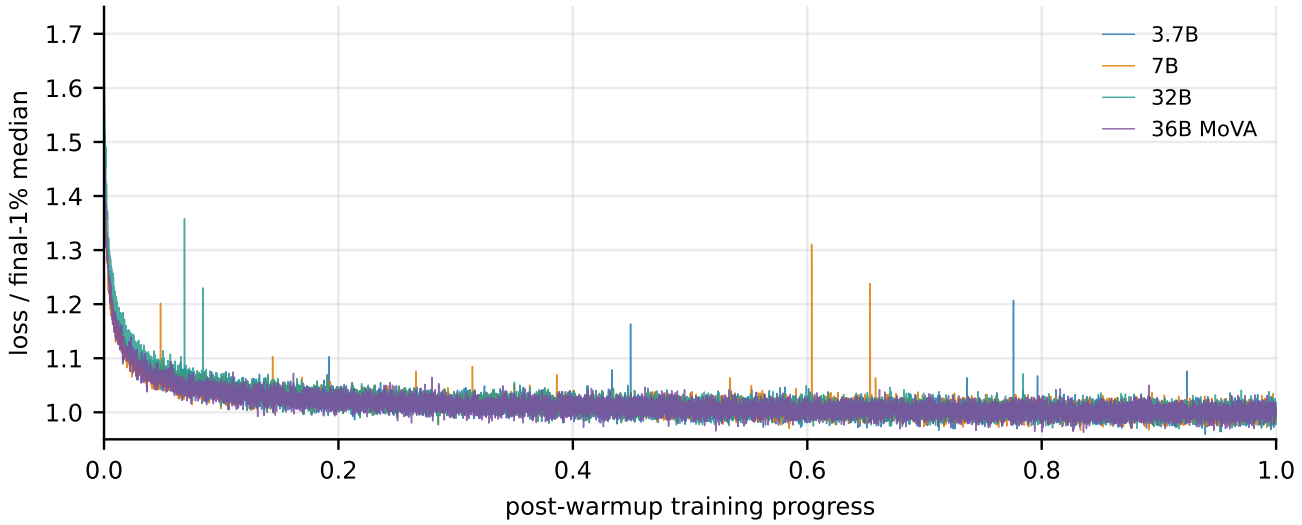


Figure 4: Normalized pre-training loss of the four models trained on the same 22T-token corpus, on the accepted trajectory. Each model's loss is divided by its median over the final 1% of steps, and progress is counted after the 3,000-step warmup. Every 100th step is shown, without smoothing.

3.3.2 Pre-Training Dynamics & Stability

This subsection describes how the pre-training runs behaved, with a focus on the 375B flagship: how the loss evolved across models, how we detect and count loss spikes, whether the gradient norm warns of them, and how restarts and rollbacks were used to handle them.

Loss across scales. The 3.7B, 7B, 32B, and 36B-A4B models trained on the same 22T-token corpus, in two data orders: the 3.7B and 36B saw the same batches at every step, as did the 7B and 32B. Their normalized losses follow nearly the same trajectory (Figure 4). Each model's loss is divided by its median over the final 1% of steps and plotted against progress after warmup. The normalization aligns the endpoints by construction; it does not force the earlier part of the curves to agree. The agreement spans dense and sparse models and nearly an order of magnitude in parameters, and is consistent with Bergsma et al. (2026), who find that loss curves collapse when hyperparameters are scaled together with the training budget, as ours are (§3.1.1).

Loss spikes. Loss spikes — sharp, transient jumps in the training loss that interrupt an otherwise smoothly decaying curve — are a recurrent instability of large language model pre-training (Molybog et al., 2023; Takase et al., 2025). They have been linked to aggressive learning-rate schedules and to a lagging second-moment estimate in Adam-style optimizers (Shazeer and Stern, 2018; Wortsman et al., 2023). Unlike K2-V2, whose learning rate followed a cosine decay from the start (K2 Team, Institute of Foundation Models, 2025), our pre-training runs held the peak learning rate for the whole run, except for the 0.9B, whose cosine schedule decays to 10% of the peak. This difference may matter for when spikes occur (see below). To quantify spikes across the complete pre-training history of every `K2-Horizon` run on a common scale, we use the robust local z-score of K2 Team, Institute of Foundation Models (2025), which measures how abnormal the current loss is relative to its recent history. For loss values y_t over training steps t, we compute a rolling median m_t over a trailing window $W(t)$ spanning 1% of the full training duration; to estimate the local variability of a curve whose level falls throughout training, we take, within

each window, the median of the absolute deviations from the rolling median:

$$\mathrm{MAD}_t \;=\; \mathrm{median}_{s\in W(t)}\big|\, y_s - m_s \,\big|, \qquad z_t \;=\; \frac{y_t - m_t}{1.4826\ \mathrm{MAD}_t}. \tag{12}$$

We flag a loss spike wherever $z_t > 5$. One spike episode runs from the first step with $z_t > 5$ until z_t stays below 2 for three consecutive steps, and a later excursion starts a new episode. Intuitively, z_t reports how many typical fluctuations a given step lies above the local baseline, adapting to the local noise level as the loss falls. A run is often resumed from a checkpoint: from the latest one (a restart, rewinding at most the 250-step checkpoint interval) or from an earlier one (a rollback, rewinding further). A run that is resumed repeatedly leaves two histories: the accepted trajectory, the chain of checkpoint-resumed segments that led to the final model, and the branches abandoned along the way, whose rows on superseded segments cover step ranges that were later replayed. We apply the definition to both histories (K2 Team, Institute of Foundation Models, 2025), treating the spikes on abandoned branches as candidate failure events. Spikes on both histories are scored against the robust statistics (m_t, MAD_t) of the accepted trajectory alone, so an excursion has one consistent meaning across all runs. Table 5 lists, for each run, its length and its counts of spike episodes and rollbacks.

Spikes across the fleet. Table 5 applies the same detector to every pre-training run. Four runs trained stably: the 0.9B, 3.7B, 32B, and 36B models were rolled back at most 1.2 times per 100,000 steps. The 7B (3.9) and the flagship (13.8 in phase 1, 4.6 in phase 2) were not. The flagship's spikes are brief but heavy-tailed: half of its phase-1 episodes last four steps or fewer, yet the median phase-1 episode peaks at $z \approx 17$ and the largest at $z \approx 950$ (Figure 5). Spikes also persist late into training (Figure 7b). In K2-V2, the vast majority of spikes fall in the first 40% of training (K2 Team, Institute of Foundation Models, 2025), For our 32B model, the closest counterpart to K2-V2's dense architecture, only 52% fall in the first 40% when counted the same way (including episodes on erased branches). We hypothesize that the long constant-learning-rate phase is the reason.

Gradient norm. The gradient norm gives little warning of a loss spike. We modeled the per-step probability that a spike begins as a function of the gradient norm in the preceding steps, controlling for the time since the last spike, the number of recent spikes, and the time since the last restart. In both phases, the only window with a clear signal is the last ten steps: each standard deviation of elevation in the log gradient norm over those steps multiplies the odds of a spike by 1.9 in phase 1 and 1.8 in phase 2. Excursions 11 to 100 steps earlier add nothing measurable, and their confidence intervals span one in both phases. This rise is close enough to the onset that it may be part of the spike itself, so we do not read it as an independent warning. The reported norm is measured before clipping, which at a threshold of 1.0 engaged on about 1.5% of executed steps in phase 1 and 0.5% in phase 2 (Figure 6).

Findings and interventions. On the 375B flagship, a large share of the instability never reached the final model, because it occurred on branches that were later discarded when training resumed from a checkpoint; the share is larger in phase 1 than in phase 2. Counting spike episodes at every logged step, phase 1 has 83 episodes on the accepted trajectory and 184 on erased branches; phase 2 has 71 and 38, respectively. Erased branches thus account for 69% of phase-1 episodes and 35% of phase-2 episodes (Figure 7a; Table 5).

Our standard response to a spike was to resume from the latest checkpoint. In 40 cases we instead rolled back to an earlier checkpoint: 29 times in phase 1 and 11 times in phase 2. Of these 40 rollbacks, 35 followed a spike episode, and the deepest rewound 4,217 steps (Figure 7a).

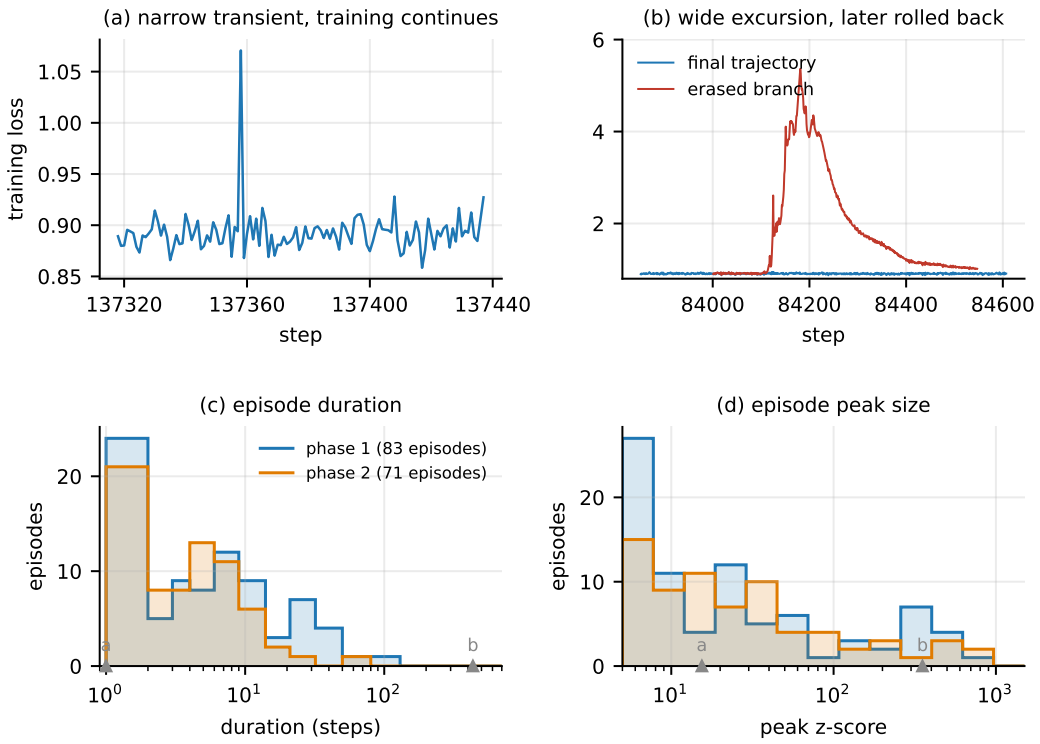


Figure 5: Spike shapes in the flagship. (a) A single-step transient in phase 1, peaking at about 15 times the usual fluctuation above the baseline (loss 1.07 against 0.89); training continues unaffected. (b) A wide excursion in phase 1 that begins at step 84,114 on a branch later erased by rollback: the loss rises from about 0.9 to 5.4, stays above 2 for 151 steps and is still above baseline when the branch ends at step 84,547. The run then resumes from step 82,001, and the final trajectory over the same steps (blue) shows no excursion. (c, d) Distributions of episode duration and of peak z-score (Equation 12) on the accepted trajectory, for the two phases; triangles mark the examples in (a) and (b), and the wide excursion lies on an erased branch and is not counted.

Rollbacks in practice. The loss monitor (§3.2.1) answers a spike by restarting from a checkpoint. Engineers add manual rollbacks when needed. The following example shows how automatic restarts and manual rollbacks interact. Near step 199,555 the loss jumped from about 0.9 to about 18, and the run was resumed from the checkpoint at step 199,500. The replay spiked again at nearly the same place: across ten resumes, five of the six attempts that reached step 199,538 spiked there, with onsets between steps 199,538 and 199,542. Only a deeper rollback, to step 196,000, broke the pattern; its replay passed this point without a spike, as did four later resumes at step 199,500. The trouble then moved forward instead of disappearing. Spikes recurred near steps 199,860 and 199,900 and again near 200,520, where the run was resumed from the checkpoint at step 200,500 more than forty times, and training moved past this stretch only after a rollback to step 200,000. This stretch shows that the monitor can resume repeatedly from the same checkpoint when a spike replays at the same step (Figure 8). We leave this unguarded on purpose: repeated failures at one step flag the checkpoint for an engineer, who then decides how far to roll back.

3.3.3 The Mid-Training Run

Mid-training, which ran at a lower learning rate, was more stable than pre-training. It ran in four stages, each continuing from the end of the previous one: stage 1 at 32k context, stage 2 at 128k, and stages 3 and 4 at 512k. The learning-rate schedule and token budgets are given in §3.1.3, and the parallel layout, which changes only by adding context parallelism at 512k, in §3.2.2.

Interruptions were far milder than in pre-training. We call a resume from the latest checkpoint a restart

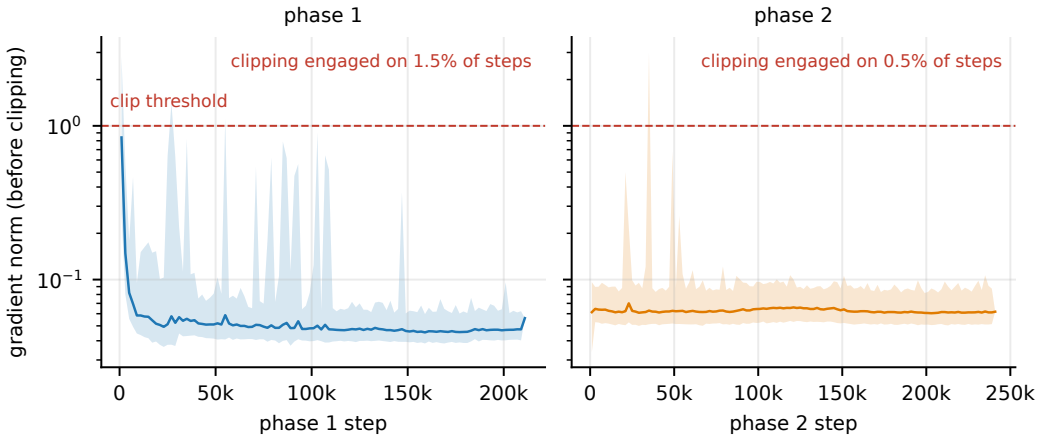


Figure 6: Gradient norm before clipping for the 375B flagship, in bins of 2,000 steps over all executed steps of each phase: median (line) and 5th to 95th percentile (band), on a log scale. The dashed line is the clipping threshold of 1.0, which engaged on 1.5% of steps in phase 1 and 0.5% in phase 2.

Stage	Context	Steps	Restarts	Rollbacks	Deepest rollback	Spike episodes	Loss (first / last 100)
Stage 1	32k	32,500	15	2	1,249	2	0.81 / 0.65
Stage 2	128k	15,000	3	1	628	3	0.51 / 0.46
Stage 3	512k	3,500	5	0	—	7	0.47 / 0.45
Stage 4	512k	6,000	7	0	—	0	0.45 / 0.41

Table 6: Mid-training run statistics for the 375B model. A restart resumes from the latest checkpoint and a rollback from an earlier one; the deepest rollback is in steps. Spike episodes are counted on the accepted trajectory. Loss is the mean over the first and last 100 steps of each stage.

and a resume from an earlier checkpoint a rollback. Over the four stages, the run restarted 30 times and was rolled back three times: twice in stage 1, rewinding at most 1,249 steps, and once in stage 2, rewinding 628 steps (Table 6). The deepest of these is less than a third of the flagship's deepest pre-training rollback (4,217 steps), and in stages 3 and 4 every interruption resumed from the latest checkpoint.

Spikes were rare and mostly marginal. Under the detector of §3.3.2, the accepted trajectory contains two spike episodes in stage 1, three in stage 2, seven in stage 3, and none in stage 4. The episodes in stages 2 and 3 are marginal (z between 5.0 and 5.8); only stage 1 has two larger excursions (z between 10 and 20). Within each stage the loss declines (Figure 9), for example from an average of 0.81 over the first 100 steps of stage 1 to 0.65 over its last 100. Loss values are not comparable across stages, because the data mixture and context length can change from one stage to the next.

3.4 Base Model Evaluation

We evaluate `K2-Horizon` at every model size after each stage of mid-training. Our curriculum introduces agentic, instruction-following, and conversational data early (§5), so intermediate checkpoints are not directly comparable to external models: they have an advantage over base models and a disadvantage against fully post-trained ones. We therefore report these results as reference points for how capabilities evolve across pre-training and mid-training.

Benchmarks. Our mid-training evaluation suite has the following 10 metrics: MMLU (Hendrycks et al., 2021) (0-shot, accuracy avg @ 1), MMLU-Pro (Wang et al., 2024a) (0-shot, accuracy avg @ 1), BBH (Suzgun et al., 2022) (0-shot, accuracy avg @ 1), GSM8K (Cobbe et al., 2021) (0-shot, accuracy avg @ 1),

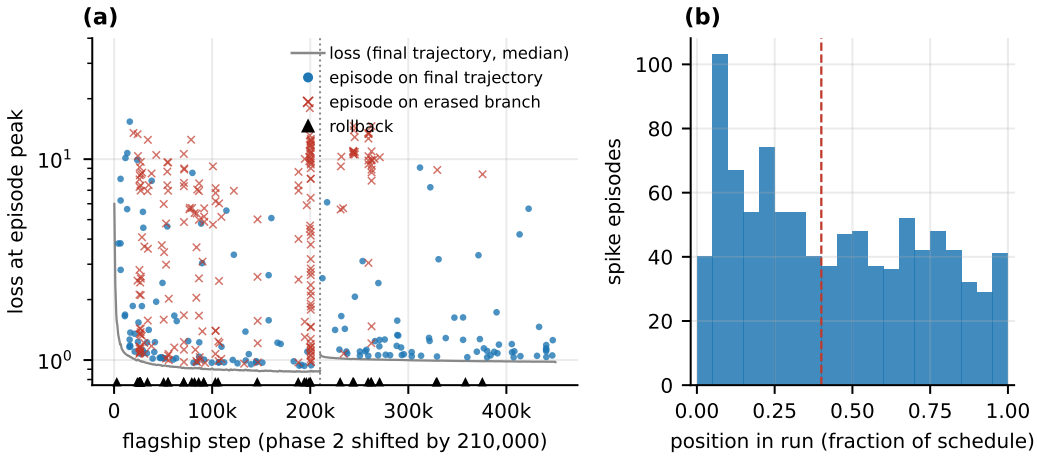


Figure 7: Spike episodes. (a) Flagship loss at the peak of each episode, on the final trajectory (blue dots) and on branches later erased when the run resumed from a checkpoint (red crosses; a replay of the same steps counts again), with rollbacks of more than 250 steps (triangles). The grey line is the median loss of the final trajectory; phase 2 is shifted by 210,000 steps. (b) Position of each episode within its run, pooled over the two flagship phases and the main runs of the five smaller models (977 episodes on final trajectories). Half of the episodes (49.7%) fall in the first 40% of a run (dashed line).

GPQA Diamond (Rein et al., 2024) (accuracy avg @ 16), AIME 2026 (accuracy avg @ 32), MBPP (Austin et al., 2021) (0-shot, pass@1), and IFEval (Zhou et al., 2023b) (0-shot) evaluated across three distinct strict adherence metrics: Strict Average Accuracy, Strict Prompt Accuracy, and Strict Instruction Accuracy.

Benchmark scores are sensitive to implementation details, so we run all evaluations, including those of external models, with a single open-source harness, Eval360[2], for consistency and reproducibility.

Flagship results. Table 7 compares the performance of K2-Horizon 375B-A23B after each mid-training stage against open-weight models in the same weight class: GLM-5.1 (GLM-5 Team et al., 2026), GLM-4.7 (GLM Team et al., 2025), MiniMax-M2.7 (Chen et al., 2026a), and Qwen3.5-397B-A17B (Qwen Team, 2026).

Overall, the K2-Horizon flagship base model achieves empirical performance competitive with leading post-trained systems. While trailing slightly on broad general-knowledge benchmarks, it leads on AIME 2026 and GSM8K and is within 3 points of the best model on GPQA Diamond. Furthermore, the early incorporation of instruction data within our mid-training mixture enables the model to attain IFEval compliance rates typically observed only in fully post-trained models.

[2] https://github.com/ifm-ai/Eval360

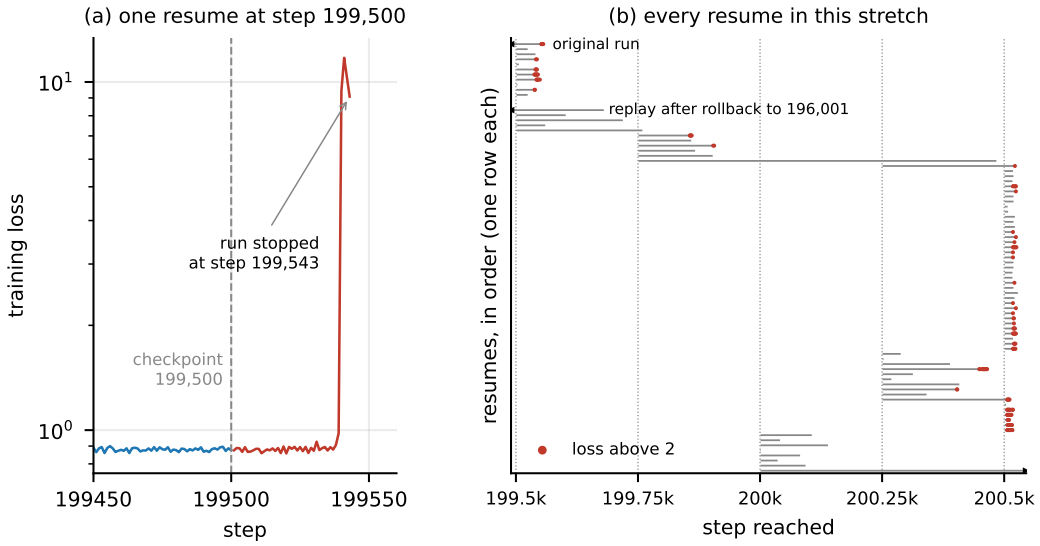


Figure 8: Resumes near step 199,500 in phase 1. (a) One resume from the checkpoint at step 199,500: the loss rises from about 0.9 to above 9 at step 199,540, and the run is stopped at step 199,543. (b) The original run (top row) and every later resume, one row each in order, from its starting step to the furthest step reached; red marks steps with loss above 2. The replay after the rollback to step 196,000 starts left of the axis, and the last resumes, from step 200,000, run past its right edge (arrowheads).

Benchmark	**K2-Horizon Mid 4**	K2-Horizon Mid 3	K2-Horizon Mid 2	K2-Horizon Mid 1	GLM 5.1	GLM 4.7	MiniMax M2.7	Qwen 3.5
# Total params	375B	375B	375B	375B	754B	355B	230B	397B
# Active params	23B	23B	23B	23B	40B	32B	10B	17B
General Knowledge & Core Reasoning								
MMLU	89.5	89.3	89.1	89.1	92.5	91.1	89.7	**92.8**
MMLU-Pro	82.6	81.7	81.3	80.9	84.4	82.0	82.3	**88.5**
BBH	87.5	87.9	87.5	87.3	84.2	90.2	84.4	**92.0**
Math & STEM Tasks								
GPQA Diamond	84.9	78.1	78.0	70.7	85.5	85.7	**87.9**	81.6
GSM8K	95.1	95.3	95.5	**95.7**	91.5	92.9	90.0	91.4
AIME 2026	**92.5**	91.9	89.2	81.3	90.0	91.8	85.8	90.0
Coding Tasks								
MBPP	93.8	87.6	92.0	85.8	**96.4**	95.6	91.2	93.0
Instruction Following								
IFEval Avg	92.1	88.4	90.0	85.2	**94.3**	91.4	91.5	91.5
IFEval Prompt	90.6	86.3	88.7	85.2	**93.2**	89.7	90.0	90.4
IFEval Inst	93.6	90.4	91.2	89.6	**95.4**	93.2	92.9	92.7

Table 7: Mid-training performance of K2-Horizon 375B-A23B across the four mid-training stages compared to leading open-weight models in the same weight class. We highlight the best result per row in **bold**, and underline the second best.

Mid-range and lightweight results. Table 9 compares the final mid-training checkpoints for all remaining K2-Horizon variants. For the 7B, 3.7B, and 0.9B, we directly benchmark against both base and post-trained Qwen3.5 (Qwen Team, 2026) models in the same weight class.

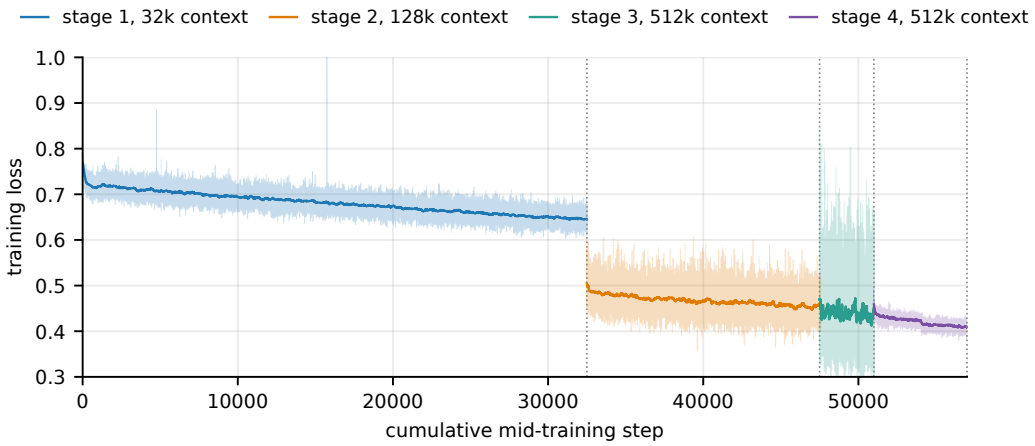


Figure 9: Training loss across the four mid-training stages (context lengths 32k, 128k, 512k and 512k). Dotted lines mark stage boundaries. Light: loss at every step; solid: running median. The stages differ in data mixture and context length, so loss levels are not comparable across them.

We find that our 36B MoVA variant performs comparably to the 32B dense model on several benchmarks, despite using about one-seventh of the active parameters and a similar total parameter count. Although they do not broadly match the scores of the corresponding post-trained Qwen3.5 models, the 7B and 3.7B variants demonstrate robust capabilities for base models, notably outperforming their post-trained counterparts on MBPP. Finally, our smallest 0.9B variant consistently outperforms the post-trained Qwen3.5-0.8B across all mathematics and coding benchmarks, with particularly significant margins on AIME, GPQA Diamond, and MBPP.

Long context. To assess the evolution of extended sequence modeling and retrieval capabilities over long context, we evaluate the 7B and 375B models' performance on the RULER benchmark (Hsieh et al., 2024), encompassing standard Needle-In-A-Haystack (NIAH) (Nelson et al., 2024) tasks and advanced context-dependent challenges.

Table 8 details the average RULER accuracy across context lengths ranging from 4k to 128k tokens. We evaluate the 7B model at the conclusion mid-training Stage 2 and Stage 3, and the 375B model at the end of mid-training Stage 1, Stage 2 and Stage 3. Because the mid-training data mixture progressively expands the maximum sequence length as training advances, we only evaluate models up to their seen context horizon at each respective stage.

Model Checkpoint	4k	8k	16k	32k	64k	128k
K2-Horizon 7B (Mid 2)	96.13	94.11	93.00	89.96	82.52	–
K2-Horizon 7B (Mid 3)	95.46	94.64	93.81	91.07	82.71	76.90
K2-Horizon 375B (Mid 1)	98.37	97.62	97.19	–	–	–
K2-Horizon 375B (Mid 2)	98.24	97.61	97.32	97.21	95.22	–
K2-Horizon 375B (Mid 3)	98.43	97.55	97.04	96.54	94.22	90.65

Table 8: Average RULER accuracy across expanding context lengths during mid-training for the 7B and 375B models. Evaluation is constrained to the context length observed by the model up to that stage.

	K2-H 32B	K2-H 36B-A4B	K2-H 7B	Qwen3.5 9B	Qwen3.5 9B Base	K2-H 3.7B	Qwen3.5 4B	Qwen3.5 4B Base	K2-H 0.9B	Qwen3.5 0.8B	Qwen3.5 0.8B Base
# Total Backbone Params	32.2B	36.2B	6.9B	7.4B	7.4B	3.77B	3.9B	3.9B	0.88B	0.60B	0.60B
# Active Backbone Params	32.2B	4.7B	6.9B	7.4B	7.4B	3.77B	3.9B	3.9B	0.88B	0.60B	0.60B
Post-trained				✓			✓			✓	
Seen agentic & chat data	✓	✓	✓	✓		✓	✓		✓	✓	
General Knowledge & Core Reasoning											
MMLU (0-shot)	88.1	86.9	83.3	**88.1**	87.4	80.7	**85.2**	84.3	44.4	**53.2**	52.9
MMLU-Pro	80.3	79.5	75.7	**82.3**	80.7	73.0	**79.4**	77.2	40.3	**42.3**	36.5
BBH (0-shot)	87.6	88.9	87.8	**91.6**	81.0	84.4	**89.7**	78.0	**43.8**	37.8	23.0
Math & STEM Tasks											
GPQA Diamond	83.7	81.3	73.9	**80.5**	75.9	73.3	**74.6**	68.8	**27.8**	11.6	27.6
GSM8K (0-shot)	95.8	94.9	95.1	**95.5**	93.9	92.3	**95.2**	92.5	**53.1**	48.6	50.7
AIME 2026	92.2	92.1	87.6	**87.9**	85.1	**88.2**	81.2	76.5	**12.3**	0.2	1.0
Coding Tasks											
MBPP (0-shot)	91.2	92.8	**91.0**	80.2	89.2	**86.4**	77.2	85.2	**53.2**	32.8	15.8
Instruction Following											
IFEval Avg	89.3	90.7	**89.9**	**89.9**	80.4	85.7	**89.7**	78.5	43.1	**61.0**	38.9
IFEval Prompt	87.4	89.3	88.2	**88.4**	76.7	83.5	**88.0**	75.0	36.6	**56.7**	33.6
IFEval Inst	91.1	92.2	**91.6**	91.4	84.1	87.9	**91.5**	82.0	49.5	**65.8**	44.1

Table 9: Final mid-training performance of K2-Horizon mid-range and lightweight models compared to base and post-trained Qwen3.5 models. Bold marks the best result within each size group.

Evolution of Capabilities. To understand how K2-Horizon acquires and refines its capabilities, we track its performance trajectory continuously across both the pre-training and mid-training phases. We open-source the W&B[3] runs for all training stages and model sizes, which include regular benchmark evaluations conducted throughout each phase to offer high resolution performance data. Figure 10 shows the evolution of performance on a few selected benchmarks for the 375B-A23B variant, across phase 2 of pre-training and mid-training stages 1-4. These benchmarks were not measured during Phase 1 pre-training.

Interestingly, BBH and MBPP exhibit sharp but transient performance drops during Mid 4. Both recover at the following checkpoint, and neither drop coincides with a noticeable change in training loss. This behavior is even more pronounced during mid-training stage 1 of the 3.7B model. Figure 11 shows the evolution of MMLU-Pro, MBPP, and GSM8K throughout pre-training and mid-training, together with training loss sampled at the same checkpoints. At 20,000 steps into Mid 1, MMLU-Pro and GSM8K decline by approximately 28 and 15 percentage points, respectively, without a corresponding loss spike. Both benchmarks recover by the next evaluated checkpoint and exceed their performance prior to the drop.

4 Pre-Training Data

Our 3.7B, 7B, and 32B dense models and the 36B-A4B model were pre-trained on 22T tokens, and the 375B MoE model on about 15T tokens in two phases (Table 2). The natural data portion builds on TxT360, the pre-training corpus of K2-V2 (K2 Team, Institute of Foundation Models, 2025), which we extend with two more web crawls, ClueWeb and HPLT (§4.1), with public datasets for web text, code, and math (§4.6), and with multilingual corpora covering 17 languages (§4.7). The bigger change from K2-V2 is synthetic data: about 10T of the tokens we train on are synthetic, and nearly 17% of the corpus consists of problem-solving trajectories with explicit reasoning (Sections 4.2, 4.3, and 4.4).

[3] https://forge.coreweave.com/wandb/llm360/projects

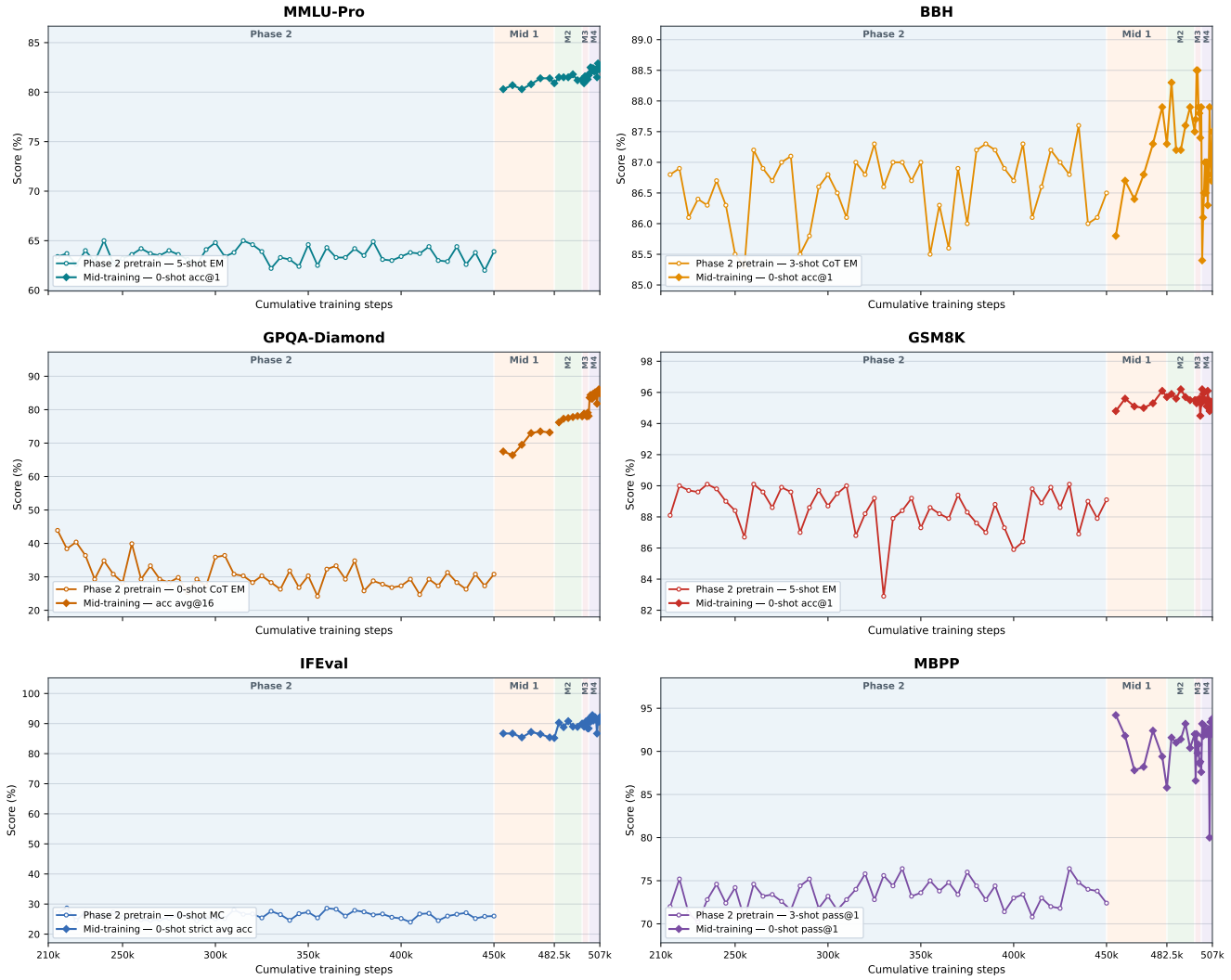


Figure 10: Evaluation performance of the 375B-A23B model during Phase 2 pre-training and four mid-training stages. Shaded regions indicate the training stages. Phase 2 and mid-training results use distinct markers because their evaluation protocols differ, so discontinuities between the two curves should not be interpreted as direct performance gains.

Maximizing distribution support. Our goal for synthetic data is to expand the model's support beyond natural data. We synthesize data both to re-balance the training distribution toward target domains and to add content that is scarce or absent in natural data, such as explicit reasoning. This includes reasoning traces for 250M math problems (§4.2) and 259M synthesized programming tasks (§4.3), reasoning on instructions with verifiable constraints (§4.4), and texts that exhibit behaviors such as planning, weighing alternatives, or counterfactual reasoning (§4.5). Our hypothesis is that a base model with broader support is better prepared for post-training. This is motivated by our K2-V2 (K2 Team, Institute of Foundation Models, 2025) work where we introduced reasoning in mid-training and by prior work on the relation between base-model support and RL (Yue et al., 2025; Wang et al., 2025; Gandhi et al., 2025).

Diversity first. Broad support requires diverse data. At trillions of tokens, synthetic data easily collapses into a few repetitive patterns, making diversity the main constraint in our data design. We address this with two techniques. First, we introduce "diversity knobs": independent attributes of a prompt that are sampled and combined, so that the number of distinct prompts grows combinatorially with the number of

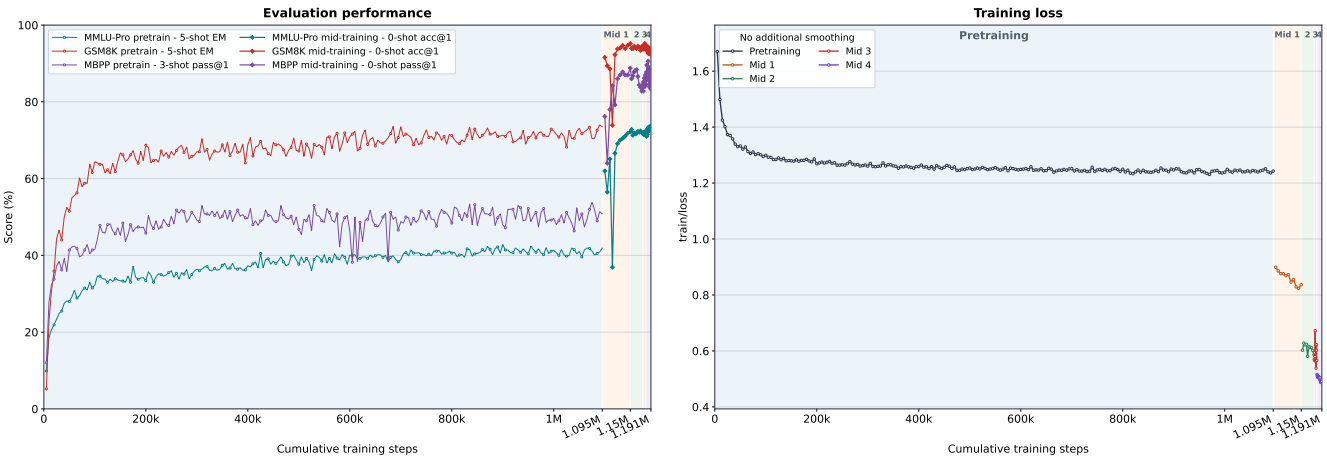


Figure 11: Evaluation performance and training loss of the 3.7B model across pre-training and four mid-training stages. The left panel reports MMLU-Pro, GSM8K, and MBPP at their available checkpoints; distinct markers denote the different evaluation protocols used during pre-training and mid-training. The right panel shows train loss, sampled at the same checkpoints.

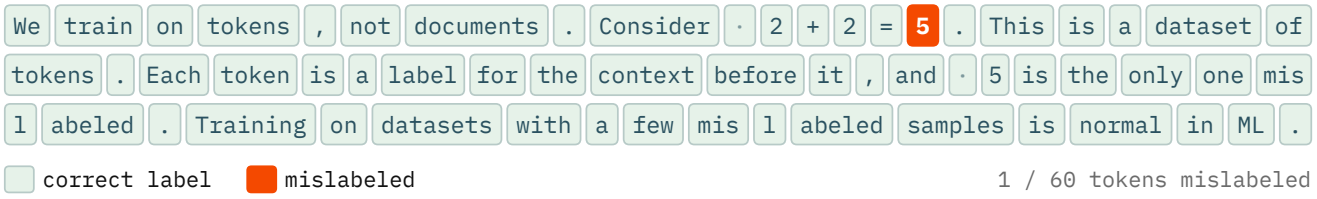


Figure 12: A document is a dataset of tokens: each token is a label for the context before it. An error such as $2 + 2 = 5$ is one mislabeled token, and the tokens around it remain correct labels. This is why we do not discard synthetic trajectories that contain mistakes.

knobs. This combinatorial explosion yields tens of billions of distinct prompts (e.g., 6.6×10^{10} concept–knob combinations for programming tasks; §4.3). Second, we ground synthesis in documents retrieved by Search360[4], our search index over 994M documents from our web corpus (§4.5). To verify that this works, we measure diversity by compression with WZIP, our own compressor: a less diverse corpus compresses better, and our final pipelines closely track the compression trend of natural web and code data (Figure 14).

No verification in synthetic pre-training data. Most of our synthetic data is not verified: math solutions are not checked against reference answers, and generated code is not executed. We view each document as a dataset of tokens, in which every token is a label for the context before it (Figure 12). A trajectory that ends with a wrong answer still contains thousands of tokens of valid steps, notation, and techniques, and its few mislabeled tokens are ordinary label noise. Strict verification-based filtering would discard all of these tokens and reduce data diversity. We do filter defects that corrupt many tokens at once, such as truncated generations, leaked special tokens, and degenerate loops.

Mixing for the target scale. Which mix is best depends on scale. Math and SFT-style data help small models quickly but saturate, while web text pays off as model capacity grows. We therefore chose the mix by fitting scaling laws on small models and predicting benchmark scores at production scale (§4.9). The prediction reversed our decision: the mix that was best in 1.5B runs is predicted to be the worst of three at 32B (Table 17).

[4] https://github.com/ifm-ai/search360

Diversity beats mixing

The largest gains came from new and diverse data, mostly synthetic pipelines with strict diversity control. Tuning the mix mattered less: at 32B, the predicted gap between our best and worst candidate mix is about two points on the benchmark average (Table 17).

4.1 Web data

Our web-data collection builds on the source mixture and curation methodology used for K2-V2 (K2 Team, Institute of Foundation Models, 2025), combining large-scale web crawls with curated domain-specific sources. For K2-Horizon, we extend this collection with ClueWeb (Overwijk et al., 2022) and HPLT (Burchell et al., 2025).

Data sources. The collection comprises the following sources, listed in descending order of cross-source deduplication priority: MegaMath, Wikipedia, HackerNews, StackExchange, PG-19, Ubuntu IRC, FreeLaw, Europarl, Europarl-Aligned, USPTO, arXiv, PhilPapers, S2ORC abstracts, S2ORC full text, PubMed abstracts, PubMed Central, Common Crawl, ClueWeb, and HPLT.

Processing and deduplication. For sources shared with K2-V2, we follow the corresponding source-specific preprocessing and filtering procedures described in that report. The newly added ClueWeb (Overwijk et al., 2022) contributes an additional large-scale web crawl, while HPLT (Burchell et al., 2025) provides web text originating from Common Crawl and the Internet Archive. Both additions undergo the same processing and filtering pipeline used for Common Crawl in K2-V2, including language identification, URL filtering, line-level cleaning, document-level quality filtering, and removal of personally identifiable information. We apply global deduplication across the combined collection, retaining documents from higher-priority sources when duplicates are identified across sources.

4.1.1 Web Data Annotation, Curation, and Shuffling

Annotation. We annotated 6.754 trillion tokens across 7.751 billion web documents, excluding the `txt360-qa` subset of TxT360 (Tang et al., 2024), whose QA-annotated passages duplicate web text already in the corpus. Following the ensemble bucketing approach of Nemotron-CC (Su et al., 2024), we combined FineWeb-Edu, fastText ELI5, and fastText PreSelect scores. Each score was mapped to one of 20 ordered buckets using fixed thresholds; the maximum bucket index determined the document's quality tier (Table 10). This retains documents rated highly by any classifier. We additionally assigned topic and format labels with WebOrganizer's TopicClassifier and FormatClassifier, which each distinguish 24 categories (Wettig et al., 2025).

Filtering and curation. We converted heterogeneous inputs, primarily JSONL, to typed Parquet for compact columnar storage and selective reads. The resulting annotations supported construction of the `web-high` and `web-high-medium` training subsets. Lazy queries in Polars enabled filtering by quality, topic, format, and document length, and aggregation of counts and scores to compare candidate data mixtures at scale. We used `party` for document inspection. We release `party` and other tools we developed for working with pre-training data.

Quality tier	Final bucket	Tokens (billions)	Documents (millions)
High	19	979.675	947.110
Medium-High	18	915.456	770.590
Medium	12–17	3,277.655	3,323.464
Medium-Low	7–11	1,140.956	1,699.017
Low	0–6	439.786	1,010.585
Total		6,753.529	7,750.767

Table 10: Annotated corpus statistics. Counts are rounded; no documents were unclassified.
Counts are rounded; no documents were unclassified.

Global shuffling. We used a distributed two-pass shuffle following the algorithm described by Hardin (Hardin, 2018). The first pass streamed Parquet records and assigned each document uniformly at random to one of M compressed temporary partitions. The second pass combined each partition's records across workers, shuffled them in memory, and emitted JSONL training shards. Choosing M so that each partition fits in memory allows corpus-wide randomization without loading the entire dataset. Fixed seeds make runs reproducible for a fixed input and execution configuration.

Abbreviated document.

```
text: "Microsoft Outlook 2021/365 Complete
      provides the basic and advanced
      concepts and skills ..."
subset: commoncrawl
FormatClassifier: 16
TopicClassifier: 20
fineweb_edu_classifier_score: 1.758
fasttext_eli5_score: 0.079
fasttext_preselect_score: 0.988
jais_token_count: 134
final_bucket: 19
quality_category: High
```

Text is excerpted and scores rounded. Token count refers to the original document.

Example filter and summary.

```
import polars as pl

ldf = pl.scan_parquet("web/**/*.parquet")
summary = (
    ldf.filter(
        (pl.col("FormatClassifier") == 1)
        & (pl.col("TopicClassifier") == 14)
    )
    .group_by("subset")
    .agg(pl.len().alias("documents"))
    .collect(engine="streaming")
)
```

Illustrative selection; these category IDs are query examples, not final mixture criteria.

Inspecting the data is essential

We found that directly inspecting documents was one of the most valuable practices in curating our pre-training corpus. To make this easy, we developed `party`, a command-line tool that randomly samples and displays documents from corpus files or directories across multiple formats, including JSON, JSONL, Parquet, and tokenized datasets. This simple workflow helped us uncover many issues in our data pipelines.

Decompressing the web. A web document is a compressed record: it states its content but leaves implicit the many facts a careful reader could derive from it, such as which words recur, which entities it concerns, or how often a letter occurs. We make part of this latent content explicit by appending to each `web-high` document a few questions about the document itself, with answers computed programmatically rather than generated by a model. The augmentation adds no information that is not already in the document; it spells out the computation needed to extract it, so every answer is exact by construction and

the augmentation costs no GPU time. Each document was analyzed with spaCy (`en_core_web_sm`) for tokenization, part-of-speech tags, sentence boundaries, and named entities, where a word is a lowercased alphabetic token. From this analysis we instantiated 11 question templates in four families:

- Frequency: words that appear exactly three times, the longest word that appears more than twice, and the number of occurrences of a random letter.
- Contextual filtering: the most frequent verbs excluding auxiliaries, words longer than six characters, and capitalized words that do not start a sentence.
- Threshold and ranking: words that appear between five and ten times, and the three least common words that appear more than once.
- Semantic grouping: the most frequent action verbs and state verbs (separated by a fixed list of state verbs), and the most frequent people, places, and organizations.

For each document we sampled between one and five of the applicable questions uniformly, shuffled their order, and appended them after a separator and a fixed instruction to answer using only the text above. Answers are short sentences and include counts where the question asks for a ranking. The pipeline was applied to all `web-high` documents longer than 300 words.

```
<document text>
---
Read the document above. Using only that text, compute word
frequencies [...] Then answer the following questions precisely
and concisely.
Question: How many times does the letter 'g' occur in the above text?
Answer: The letter 'g' occurs 28 times in the text.
Question: Which words appear exactly 3 times in this document?
Answer: The words that appear exactly 3 times are: not, it, they,
product, can, is, with, and as.
```

This data corresponds to TxT360-v2 in our release.

4.2 Math reasoning

Our math reasoning data starts from about 250 million unique problems (after exact deduplication) from a variety of sources. These sources are drawn from permissively licensed mathematical datasets studied in the OpenThoughts project (Guha et al., 2025)[5]. To reduce the risk of contamination, we use only training splits where applicable. The resulting collection spans a broad range of problem types and difficulty levels, including templated arithmetic and numerical tasks (e.g., determining whether a large number is prime), verification of mathematical statements, and other mathematical reasoning problems. We previously used this problem set to create mid-training data for K2-V2 (K2 Team, Institute of Foundation Models, 2025).

Solutions with reasoning. We generate solutions for each of the problems with Qwen3-32B with thinking (Yang et al., 2025) and GPT-OSS-120B with high reasoning effort (Agarwal et al., 2025). Resulting pre-training documents are a simple concatenation of problem statement, reasoning trace, and answer after removing chat template special tokens. We filter out samples that did not finish within a 15.5k-token budget or contained unexpected special tokens (e.g., `commentary` or extra `analysis` channels for GPT-OSS). Resulting data consisted of 800B tokens for problems with Qwen3 generations and 424B for problems with GPT-OSS generations.

[5] https://www.open-thoughts.ai/blog/thinkagain

Socratic teacher–student dialogues. Solution traces show a single, uninterrupted line of reasoning. To also show how a solver responds to feedback (spotting an error, revising an attempt, confirming a result), we simulate tutoring dialogues between two LLM agents (Figure 13). Unlike the dialogue re-writes below, which recast an existing reasoning trace, these dialogues are generated interactively, and the student never sees the answer. We start from the hardest problems in our collection: those whose GPT-OSS-120B solution exceeded 8,192 tokens and that have a reference answer, 8.1M problems in total. More than half of them (4.5M) ask whether a large number is prime or composite (see the "Is it prime?" takeaway below), so we remove every problem that mentions either word, leaving 3.6M problems.

Teacher prompt	Effort (student / teacher)	Dialogues	Student tokens	Total tokens
Socratic	low / high	963,798	3.74B	4.69B
Socratic	medium / high	2,019,252	8.96B	10.48B
Persona	low / high	1,139,475	3.27B	3.87B
Persona	medium / high	1,990,900	8.95B	9.73B
Persona	medium / medium	999,853	1.54B	4.82B
Persona	medium / low	1,000,000	1.00B	4.77B
Total		**8,113,278**	**27.45B**	**38.37B**

Table 11: Socratic teacher–student dialogues by teacher prompt and reasoning effort. All configurations draw from the same pool of 3.6M problems. Student tokens (reasoning and answers) are exact; total tokens add the teacher's messages, estimated from their character count at the student's characters-per-token ratio.

Both agents are GPT-OSS-120B. The student receives only the problem and the system prompt "You are an expert mathematician." The teacher receives the problem and its reference answer, which is usually a short final answer rather than a worked solution, and is instructed never to reveal it. The student opens with a solution attempt. On each turn the teacher evaluates the latest attempt: if it is correct, the teacher explains why and ends the dialogue with an `<END_CONVERSATION>` token; otherwise it asks probing questions that break the problem into smaller steps or point at the faulty step, without correcting it directly. Each agent sees only the other's messages, not its reasoning. We use two teacher prompts. The first is a plain Socratic tutor. The second assigns the teacher a random persona, covering a profession, skills, hobbies, and location; the persona shapes the teacher's voice and choice of move (a targeted question, a micro-hint, an analogy, a check), and the teacher makes one move per turn and accepts any valid method, not only the one in the reference. As a second diversity knob, we vary the reasoning effort of both agents: low or medium for the student, and low, medium, or high for the teacher. We keep only dialogues that the teacher closed within 100 turns without errors. Each one becomes a pre-training document with the student as the assistant: the student's reasoning is kept, the teacher's reasoning is dropped, and the teacher's messages become user turns. The result is 8.1M dialogues with about 38B tokens, 27B of them from the student (Table 11).

Right on the first try

Although we chose problems whose GPT-OSS solutions are long, the student's first attempt is accepted in roughly 50–80% of dialogues (in a sample of 5,000 per configuration), and the median dialogue has a single student turn. Lowering the teacher's effort produces more back-and-forth: with a medium-effort student, the average number of student turns rises from 1.4–1.9 with a high-effort teacher to 2.9 with a medium-effort and 3.3 with a low-effort teacher, and the teacher's messages then outweigh the student's by roughly 2–4× in tokens. We have not checked whether these extra turns reflect genuine student mistakes or the teacher failing to recognize correct attempts.

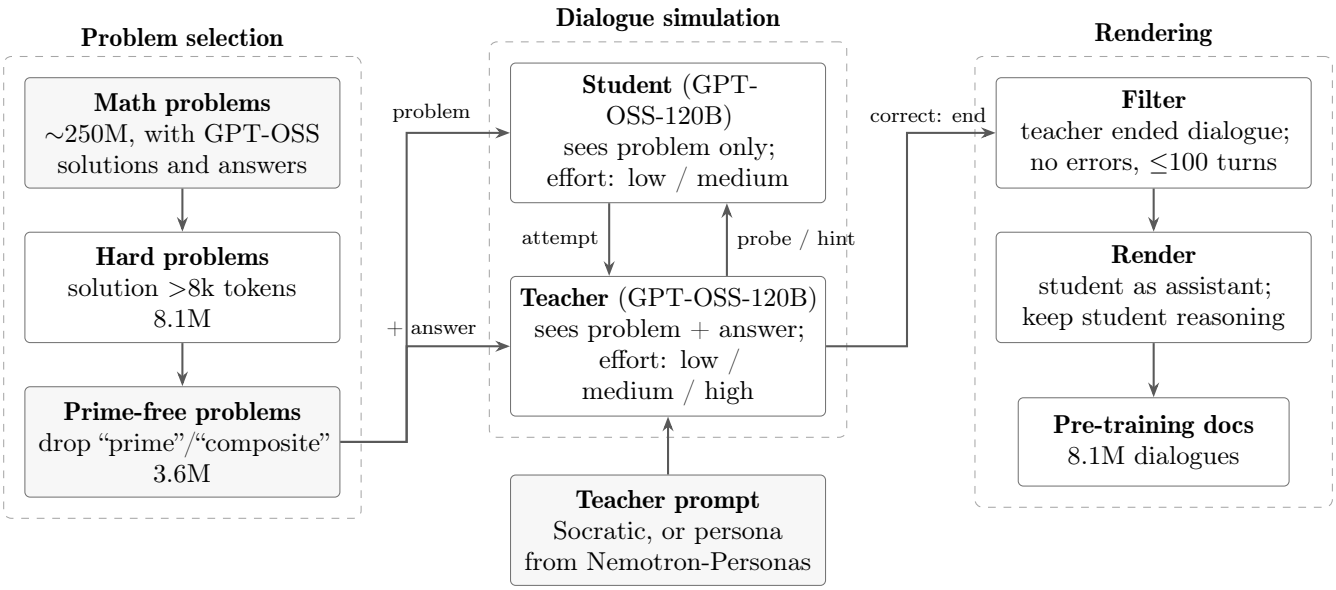


Figure 13: Socratic teacher–student dialogue generation. Hard, prime-free problems are posed to a student agent that sees only the problem. A teacher agent that also sees the reference answer checks each attempt and either probes the student's reasoning or, once the attempt is correct, ends the dialogue. The teacher prompt (plain Socratic or a sampled persona) and both agents' reasoning effort serve as diversity knobs. Completed dialogues are rendered as pre-training documents from the student's perspective.

Reasoning re-writes. LLM reasoning traces often contain detailed analysis, derivations, and references to relevant mathematical results. However, the "style" of LLM reasoning lacks diversity and can often appear unnatural. To convert knowledge and mathematical techniques from the reasoning traces into more natural text, we developed two re-writing pipelines that we applied to problems with Qwen3 reasoning and answers:

- Study guide re-write where we instruct GPT-OSS to "Transform the following problem-solving trace into educational content that teaches the concepts and methods used." resulting in 566B tokens.
- Dialogue re-write inspired by MIND (Akter et al., 2025). We adopt five styles: `debate`, `interview`, `problem_solving`, `teacher_student`, and `two_students`. For each, we create our own prompt and use GPT-OSS to re-write Qwen3 reasoning and answers into the corresponding dialogue style resulting in 1.5T tokens.

We release the full Math Reasoning data collection.

Is it prime?

When we examined the problems with the longest reasoning, we expected to find the hardest ones, but realized that most of them asked to check whether a large number is prime or to find its prime factors. Such questions make up only 4% of our problems, yet account for 59% of Qwen3 and 70% of GPT-OSS generations that exceeded the token budget. More broadly, over 80% of the collected problems are procedurally generated, and we think that the diversity and quality of the problems can be substantially improved. However, we found that training on reasoning and solutions for these tasks is already quite effective, judging by the strong performance of our pre-trained model on math benchmarks.

4.3 Code reasoning

Unlike math, we did not have a large collection of programming problems to start from, so we had to synthesize the tasks ourselves. The key challenge is diversity: simply asking an LLM to write a programming problem over and over tends to produce similar ones. To address this, we seed every task with a random combination of a programming concept, a snippet of real-world code, and a set of diversity knobs.

Task synthesis. As programming concepts, we choose 1,304 reference implementations of algorithms and data structures in 66 categories (e.g., graphs, dynamic programming, string processing) from the TheAlgorithms/Python[6] and keon/algorithms[7] repositories. We pair each concept with a random snippet of 20–30 lines of Python code from RefineCode (Huang et al., 2025), which serves as inspiration for the setting and style of the task. Diversity knobs then control how precisely the task is specified (from a vague user request to a detailed competitive programming statement), its difficulty, framing (algorithmic, practical, academic, debugging, or optimization), constraint style, application domain (one of 90), and twelve binary switches such as input/output examples, starter code, multi-part structure, type hints, or test-driven development. The knobs alone give 50.3M distinct configurations, or 6.6×10^{10} concept–knob combinations, before accounting for the code snippets. We made five passes over the 57M Python files in RefineCode, synthesizing one task per file with GPT-OSS-120B with high reasoning effort: three passes use a restricted set of knobs where every task is phrased as a vague user request, and two passes use all knobs. After filtering out generations that were truncated or exceeded 7.5k tokens, we obtained 259M tasks. Even tasks sharing the same concept rarely overlap, with only 1% of 5-grams in common on average.

Solutions with reasoning. We generate solutions with GPT-OSS-120B with high reasoning effort and a 32k token budget. Similarly to math, pre-training documents are a concatenation of the task, reasoning trace, and final answer. Code solutions tend to be long (median 8.6k tokens, and 11% exhaust the budget), so we split them by length: solutions within 15.5k tokens resulted in 100.7M documents (761B tokens), while longer ones (up to 31k tokens) resulted in 23.7M documents (543B tokens) that we reserved for mid-training. In both cases, we filter out solutions with stray channel markers. Tasks synthesized with all knobs tend to be more detailed and require longer solutions (median 10.5k vs. 7.7k tokens), so we release them as separate "hard" subsets. In addition, we release a version without reasoning (task and final answer only; 210M documents, 790B tokens), an earlier version of the reasoning data (82M documents), and the task synthesis traces from one of the all-knobs passes (49M documents, 286B tokens).

Finally, we reproduced the CodeI/O pipeline (Li et al., 2025c), which turns code into reasoning about predicting a function's inputs and outputs, resulting in 23B tokens.

We released the full Code Reasoning data collection.

Good luck!

When writing a programming task, GPT-OSS can hardly resist wishing the solver well: 73% of synthesized tasks end with "Good luck!" (often with a rocket emoji). We strip the sign-off when processing the tasks.

4.4 SFT reasoning

[6] https://github.com/TheAlgorithms/Python
[7] https://github.com/keon/algorithms

4.4.1 Instruction-following (IF) reasoning

Real instructions often come with hard requirements on the form of the answer, such as a word limit, a keyword that must appear, or a fixed number of sections. Such requirements can be verified, which makes them a source of reasoning data whose answers are verified without a judge. We synthesized prompts with such constraints and used the responses, including the reasoning, in pre-training.

Prompts. Each prompt pairs a writing task (e.g., an essay, a tutorial, a professional email, or a report on one of 15 topics) with one to five constraints drawn from 37 constraint types in 15 categories, following the taxonomy of verifiable instructions introduced by IFEval (Zhou et al., 2023a). The types cover keyword frequencies and exclusions, letter counts, the number of paragraphs, sentences, words, bullet points, and highlighted sections, the first and last word of the response or of a paragraph, postscripts and placeholders, copying and repeating spans, the response language, and capitalization. Parameters such as keywords, counts, and positions are sampled for each prompt, and 13 incompatible pairs of types (e.g., two different requirements on the first word) are never combined. The number of constraints is skewed toward fewer constraints, from 41.5% of prompts with one to 1.9% with five. The writing tasks are our own templates, and no prompts are taken from evaluation sets.

Responses and verification. We generated responses with GPT-OSS-120B (Agarwal et al., 2025) at low, medium, and high reasoning effort. Every constraint type has a code-based verifier built on regular expressions and string operations, and we kept a response only if it satisfied all constraints. We then removed exact duplicates of the user prompt. Pre-training documents concatenate the prompt, the reasoning trace, and the response, and we removed GPT-OSS-specific artifacts as for our other GPT-OSS data.

The result is 8.6M documents with 44B tokens. The reasoning dominates: its median length is 3.6k tokens, compared with 465 tokens for the response. Deduplication also reshaped the mix: single-constraint prompts have few distinct instantiations, so their share fell from about half of the verified responses to 22%. This data is released as the `instruction-following` subset of SFT-Reasoning.

Counting is the hard part

Writing to a specification is easy but checking it is not. Constraints that make the model count its own output dominate the reasoning: a requirement such as "the letter z should appear 32 times" occurs in 10% of prompts but in 35% of the longest tenth of reasoning traces, and "all lowercase words should appear at most N times" raises the median reasoning length from 3.6k to 9.6k tokens. Including a keyword or avoiding a word takes 2–3k.

4.4.2 SFT data as pre-training text

TxT360-3efforts, the SFT data we released with K2-V2 (K2 Team, Institute of Foundation Models, 2025), contains conversations in math, code, chat, STEM, instruction following, tool use and safety, with answers mostly generated by GPT-OSS-120B (Agarwal et al., 2025) at low, medium and high reasoning effort. It is a 9.5M-conversation mix sampled from a larger pool in which most questions are answered at all three effort levels. We reuse this pool in pre-training by rendering each conversation, including the reasoning, as plain text, so the same problem appears with reasoning of different lengths.

Format diversity. Rendering tens of millions of conversations with a single template would make the template one of the most repeated patterns in the data. We therefore draw a format for every conversation. In 43% of conversations (68% of those with tools), the turns are marked with role labels drawn from 21

sets (e.g., `User:`/`Assistant:`, `Human:`/`AI:`, Markdown headers, bracketed or bold labels, and one-letter prefixes), combined with one of 7 turn separators and 7 wrappers around the conversation. The remaining conversations use one of eight other layouts: turns separated only by a delimiter, a document with sections (background, query, analysis, response), question–answer pairs, instruction and response blocks, named personas, an interview transcript, a screenplay, and a compact format with one line per turn. Conversations with tools use only the four layouts that render tool definitions, calls and results, and show the list of tools in one of four styles. In every layout the reasoning precedes the response, in some under its own heading.

The result is 32.2M documents with 79.5B tokens, released as the `3efforts-pretrain` subset of SFT-Reasoning.

4.5 Pre-training Behaviors

The question augmentation of `web-high` makes explicit what a single document leaves implicit. Knowledge about a topic, however, is spread across many documents, and web text rarely shows the behaviors we want a model to acquire, such as explaining a concept to a novice, planning under constraints, weighing alternatives, or reasoning from evidence to a conclusion. We decompress this knowledge across documents: for each question, we retrieve the relevant documents from our pre-training corpus and ask a model to write a text that exhibits a chosen behavior, grounded in what was retrieved. This data corresponds to Pretrain-Behaviors in our release.

Search index. We built Search360[8], a dense retrieval service over the corpus itself. It indexes `web-high` together with the High-Quality and Medium-High-Quality partitions of Nemotron-CC-v2.1 (NVIDIA, 2025a): 994M documents split into 4.0B passages of 128 tokens, where a final passage shorter than 64 tokens is merged into the one before it. Passages are embedded with Contriever-MSMARCO (768 dimensions) (Izacard et al., 2021) and indexed with FAISS as an IVFPQ index with 4,096 centroids and 16 sub-quantizers of 8 bits, trained on 10M sampled embeddings and searched with 256 probes. The index stores no text. Arrays map each passage to its document and to a byte offset in the original JSONL shards, so a passage or a full document is read with a single seek. A FastAPI service returns either passages or full documents, ranked by the number of retrieved passages they contain, with optional re-ranking by jina-reranker-v3. Behind a round-robin load balancer with health checks, one 8-GPU node answered 1,000 queries in 53 s at 64–128 concurrent requests without re-ranking, or about 19 queries per second.

Questions. We started from Quora questions and Natural Questions and asked an LLM to write five deeper follow-up questions for each, which gave about 6M questions. For each question we retrieved the top eight documents in full.

Behaviors. Each prompt template specifies a behavior and receives the question and the retrieved documents, which the generated text must draw on. The templates fall into four families (Table 12). The general family has 58 genres, from research abstracts and legal memoranda to lesson plans, documentary scripts, and steelmanned critiques. The reasoning family has 47 modes of reasoning, such as counterfactual, first-principles, and game-theoretic reasoning. The planning family covers plans with risks and mitigations, timelines and dependencies, and fixed resource budgets, and the data science family covers machine learning system design, experimentation, and data analysis. We generated the texts with GPT-OSS-120B (Agarwal et al., 2025) at high reasoning effort and kept only the final text, without the reasoning.

Diversity knobs. In a document-grounded variant, each prompt also received three attributes sampled from 22, including mood, education level, writing style, perspective, narrative structure, audience, and

[8] https://github.com/ifm-ai/search360

factuality level, each with 3 to 21 values. The attributes are appended as constraints that the text must satisfy without naming them, so the same document yields, for example, a speculative text for children or a formal, first-person text for experts. In the main run, diversity comes from crossing the templates with the questions and their retrieved documents.

Family	Example behaviors	Documents	Tokens
General	legal memorandum, lesson plan, documentary script	207M	630B
Reasoning	counterfactual, first principles, game theory	502M	1,324B
Planning	risks and mitigations, timelines, resource budgets	290M	898B
Data Science	ML system design, A/B testing, forecasting	123M	335B
Total		1.1B	3.18T

Table 12: Search-grounded behavior data by template family.

Measuring diversity. When synthesizing data at trillion-token scale, it is easy to miss repetitive patterns that the model may then learn to reproduce. To quantify diversity, we developed a custom lossless compressor, WZIP. Its key distinction from Zlib/GZIP and Zstandard is the explicit coupling of **match-length-dependent sliding windows, dedicated hash indexes, and length-conditioned distance coding**. Short matches use compact local histories and specialized distance models, while larger histories use longer, more selective hash seeds to identify distant repetitions. This design controls short-match coding costs and limits unnecessary candidate scanning while retaining access to long-range repetitions. The decoded match length selects the distance model without an additional per-match selector. Together with bounded searches and Huffman coding, this organization targets high compression throughput on large datasets. Under consistent evaluation settings, we use the compression ratio (input bytes per compressed byte) as a quantitative proxy for textual redundancy to assess how our search and generation controls affect corpus diversity, complementing assessments of semantic coverage and data quality.

In Figure 14, we present WZIP compression ratios for the reasoning pre-training behaviors, comparing them to an earlier version of this pipeline without search and diversity knobs, and to natural code and web data. Code and web data have different baseline compression ratios, but for both the ratio grows slowly with the number of documents due to natural data diversity. Our final pipeline nearly matches the compression ratio trend of natural data, while our earlier version lacks sufficient diversity. Compression ratio was among the key metrics guiding our synthetic data generation development.

4.6 Public sources

Alongside the corpora we build and release, our pre-training mixture includes public datasets for web text, code, mathematics, and instruction-style data (Table 13). Public multilingual corpora are described in §4.7.

Web. We use Nemotron-CC-v2 (Su et al., 2024; NVIDIA, 2025b) and Nemotron-CC-v2.1 (NVIDIA, 2025a), discarding every partition below medium-high quality. About three quarters of the retained tokens are synthetic: rephrasings, translations into English, and question-answer pairs that the dataset authors generated from web documents.

Code. As in K2-V2 (K2 Team, Institute of Foundation Models, 2025), our base source-code corpus is RefineCode (Huang et al., 2025). We add Stack-Edu (Allal et al., 2025), Nemotron-CC-Code-v1, and the synthetic code data of Nemotron-Pretraining-Code v1 and v2 (NVIDIA, 2025b,a).

Math. We use the 3plus partition of Nemotron-CC-Math-v1 (Mahabadi et al., 2025), in addition to the MegaMath data in our web collection (§4.1).

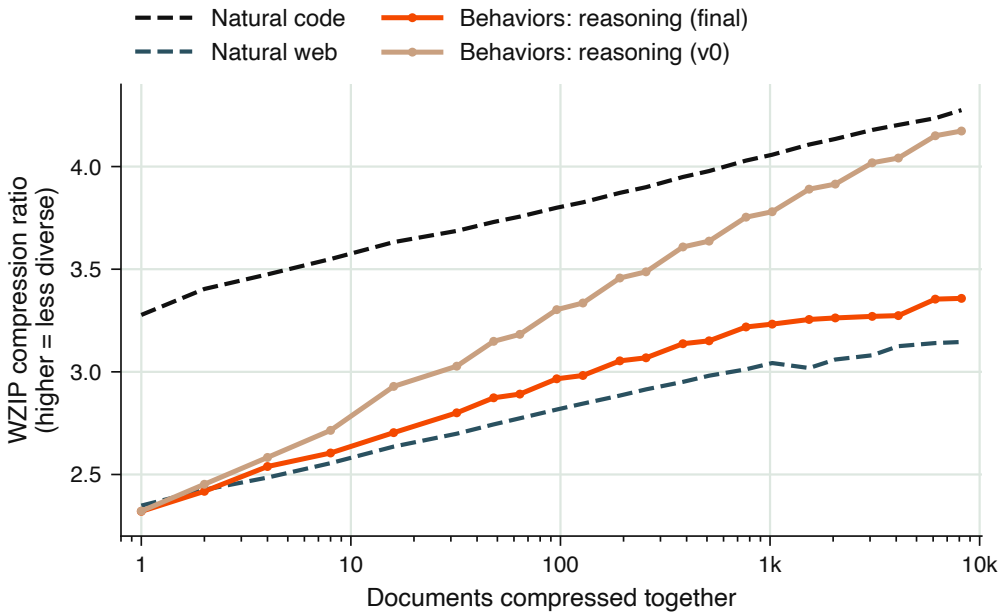


Figure 14: WZIP compression ratio versus the number of documents compressed together (log scale); a higher ratio means less diverse text. Dashed lines are natural code and web data. v0 is an earlier version of our pipeline without search and diversity knobs.

Instruction-style data. We include question-answer data in plain pre-training format from Nemotron-Pretraining-SFT-v1 and Nemotron-Pretraining-Specialized-v1. From Nemotron-Pretraining-SFT-v1, we remove every sample whose recorded generating models include a 72B-parameter Qwen2.5 model due to a license restriction, which removes 39.9% of its samples; Nemotron-Pretraining-Specialized-v1 is used in full. We also include Nemotron-Post-Training-Dataset-v1 (Nathawani et al., 2025) and the Llama-Nemotron post-training data (Bercovich et al., 2025), after removing samples generated by Llama models and non-English samples. We delete the `<think>` and `</think>` tags from these samples and keep the reasoning text between them.

We use these datasets under their original licenses and do not redistribute them.

4.7 Multilingual data

Multilingual text makes up roughly 7% of our pre-training mixture and covers 17 languages (Table 14). Three languages have a dedicated corpus: Arabic, French, and Hindi. For the remaining languages, we use public web data and translated question-answer data.

Arabic, French, and Hindi. As in K2-V2 (K2 Team, Institute of Foundation Models, 2025), our Arabic data is the corpus assembled for the Jais family of models (Sengupta et al., 2023). It contains Arabic web text from Common Crawl, public corpora such as ArabicWeb, OSCAR, CC100, Wikipedia, books and news. French is web text from RedPajama (Weber et al., 2024). Hindi is compiled from public corpora, primarily Sangraha (Khan et al., 2024), HPLT (Burchell et al., 2025), Common Crawl, FineWeb2, C4, and CulturaX, together with smaller community datasets. The three corpora contain 902M, 645M, and 256M documents, respectively.

Broader language coverage. We use FineWeb2-HQ (Messmer et al., 2025), which keeps the top 10% of FineWeb2 (Penedo et al., 2025) documents in each language according to a model-based quality score, for twelve languages: Russian, Chinese, German, Spanish, French, Japanese, Italian, Portuguese, Indonesian,

Category	Dataset	Partitions used	Tokens (B)
Web	Nemotron-CC-v2	High-Quality	650
		Medium-High-Quality	550
		High-Quality-Synthetic	1,257
		Diverse-QA	693
Web	Nemotron-CC-v2.1	All except Medium-Quality and Medium-High-Quality-Synthetic	369
		Medium-High-Quality-Synthetic	2,123
Code	RefineCode	Full	1,774
Code	Stack-Edu	Full	125
Code	Nemotron-CC-Code-v1	Full	428
Code	Nemotron-Pretraining-Code v1, v2	Synthetic-Code	698
Math	Nemotron-CC-Math-v1	3plus	133
Instruction	Nemotron-Pretraining-SFT-v1, Specialized-v1	72B Qwen2.5 outputs removed from SFT-v1	477
Instruction	Nemotron-Post-Training-v1, Llama-Nemotron	Llama outputs and non-English removed	150

Table 13: Public datasets in the pre-training mixture. Token counts are approximate and give the prepared inventory available for mixing, not the number of tokens seen in training. The Nemotron datasets are available at `nvidia` and Stack-Edu at `datasets/HuggingFaceTB/stack-edu`.

Turkish, Persian, and Arabic. We also train on Bengali, Urdu, Malayalam, and Filipino. FineWeb2-HQ does not include these four languages, so we take them from FineWeb2 itself.

Translated question-answer data. We include the translated Diverse-QA portion of Nemotron-CC-v2 (NVIDIA, 2025b), in which question-answer pairs generated from English web documents are translated into other languages. Of its fifteen languages, we keep the nine that our tokenizer supports: Spanish, French, German, Italian, Japanese, Portuguese, Russian, Arabic, and Chinese.

Our web collection also contains non-English text through Europarl and Europarl-Aligned (§4.1). We do not redistribute the multilingual data.

Language	Sources	Tokens (B)
French	RedPajama, FineWeb2-HQ, translated QA	753
Arabic	Jais Arabic data, FineWeb2-HQ, translated QA	512
Russian	FineWeb2-HQ, translated QA	100
Hindi	Compiled public corpora	84
Chinese	FineWeb2-HQ, translated QA	78
German	FineWeb2-HQ, translated QA	77
Spanish	FineWeb2-HQ, translated QA	72
Japanese	FineWeb2-HQ, translated QA	52
Italian	FineWeb2-HQ, translated QA	51
Portuguese	FineWeb2-HQ, translated QA	43
Indonesian, Turkish, Persian	FineWeb2-HQ	16
Bengali, Urdu, Malayalam, Filipino	FineWeb2	17
Total		1,855

Table 14: Multilingual pre-training data by language. Token counts give the prepared inventory available for mixing, not the number of tokens seen in training. They are estimated from character counts, using a characters-per-token ratio measured for each language with our tokenizer on 1,000 sampled documents.

4.8 Benchmark decontamination

Benchmark contamination makes evaluation scores look better than the actual model performance. In a corpus of our size, benchmark items can come in through web pages that quote them, through synthetic data built from benchmark problems, or through public datasets that include evaluation splits. We therefore scan all our training data for overlap with the benchmarks and remove the documents that match. A benchmark item that was paraphrased or translated will not be found.

Approach. We use the AllenAI decon tool (Allen Institute for AI, 2025) with a few changes. We added Python bindings and a stage that removes contaminated documents. We also use 8-token n-grams for questions, instead of the default 5, so that a match needs a longer overlap. Each benchmark item is split into a question, answer, and an optional passage. This way a document that copies only the question scores lower than one that copies the whole item. To find matches, we build an inverted index of hashed n-grams from the benchmark items and look for a matching 8-gram in each training document. When one is found, the match is extended token by token. The overlap is then scored so that rare n-grams count more than common ones, and short items have to match more closely than long ones. We remove documents that score 0.85 or higher.

Benchmarks. We prepared evaluation data for the 208 benchmarks in our registry and scanned every data source against all of them. After looking at the results for each benchmark and each dataset, we removed contamination for the 18 benchmarks we report results on. These make up 27 entries in the registry. For ARC and GSM8K, we only count matches with the test split.

Removal. Each match records the file and line of the training document. Removal therefore only rewrites the files that have matches, and every removed line is logged. We remove the whole document and not just the matched text. Contamination is rare in all of our data. The highest rate, 0.026% of documents, is in the SFT reasoning data; the next highest, 0.017%, is in the pre-training behaviors data. In 35 of the 37 datasets with any match, the rate is 0.01% or lower.

Table 15 shows what we removed: 197,149 matches in 88,625 documents from 30 datasets. Only 23 of the 27 registry entries had any match. The table does not include the SFT reasoning data and the Arabic corpus, whose removal had not finished when we collected these numbers. They add another 12,616 matches in 8,462 documents. Most removed matches are near-exact copies, and 172k of them score 0.95 or higher. MMLU and MMLU-Pro account for 166k matches, which is 84.6% of the total. About two thirds of the removed documents come from four datasets: the data science behaviors, the public instruction-style data, and two web datasets.

4.9 Data mix

The corpora above can be mixed in many ways, and the mix that looks best in a small ablation need not be the mix that is best at the scale we actually train. The two ends of our design space make this concrete. SFT-style and mathematical data are dense and immediately useful, but they are a small share of the available tokens and their benefit saturates quickly. Web text is broad and high-entropy, and we expect it to pay off as model capacity grows. We therefore chose the pre-training mix by comparing candidate mixes on their *predicted* benchmark scores at the target scale rather than on their measured scores at small scale.

Candidate mixes. We compared three candidates built from the same sources (Table 16): a web-heavy mix, a math-heavy mix, and an SFT-heavy mix. They share the code sources and the multilingual budget, and differ mainly in how they divide the remaining budget between web text on one side and mathematical

Benchmark	Matches	Docs.	Benchmark	Matches	Docs.
MMLU	117,227	66,296	MultiPL-E (Go)	148	133
MMLU-Pro	49,468	32,728	MBPP+	137	24
ARC-Easy	8,417	4,516	MultiPL-E (C++)	129	120
ARC-Challenge	5,852	3,818	BBH formal fallacies	36	7
GSM8K	5,218	966	BBH date understanding	29	23
Codeforces	3,826	3,730	BBH disambiguation QA	23	8
DeepSeek-LeetCode	2,767	2,241	BBH logical deduction	5	3
MATH-500	1,619	1,462	BigCodeBench	4	4
MultiPL-E (JS)	1,144	934	CRUXEval	4	3
BBH causal judgement	565	32	BigCodeBench-Hard	3	3
HumanEval+	353	341	MultiPL-E (Java)	2	2
BBEH	173	30			

Table 15: Contamination removed for each benchmark, ordered by the number of matches. The table lists the 23 registry entries that had at least one match with a score of 0.85 or higher. AIME 2025-I, AIME 2025-II, BBH boolean expressions, and GPQA Diamond had no matches. A match is one pair of a training document and a benchmark item, and the **Matches** column adds up to 197,149. **Docs.** is the number of documents removed for each benchmark. One document can match several benchmarks, so this column adds up to more than the 88,625 documents removed. BBH rows are single BIG-Bench Hard tasks, and BBH logical deduction is the three-object version. MultiPL-E rows are HumanEval translated into the named language. The SFT reasoning data and the Arabic corpus are not included.

and SFT-style data on the other. The web-heavy mix gives 36.8% of its tokens to web text and 4.1% to SFT-style data; the SFT-heavy mix gives 23.0% and 13.1%.

Category	Web-heavy	Math-heavy	SFT-heavy
Web (§4.1)	**36.8**	24.5	23.0
Code	13.4	11.5	10.0
Code reasoning	11.6	14.0	9.9
Math reasoning	12.7	**23.9**	20.0
SFT and instruction data	4.1	8.6	**13.1**
Pre-training behaviors	16.4	12.5	19.0
Multilingual	5.0	5.0	5.0

Table 16: Category shares (% of training tokens) of the three candidate mixes. *Web* includes our web collection, the Nemotron-CC web partitions, and web-derived question–answer and rewrite data. *Code* is source code (RefineCode, Stack-Edu, Nemotron code), and *code reasoning* is our synthetic code data with and without reasoning. *SFT and instruction data* covers the public instruction-style datasets and SFT data rendered as pre-training text. *Pre-training behaviors* are the synthetic reasoning, planning, and general-behavior sets. Bold marks the category each mix emphasizes. Columns sum to 100 up to rounding.

Predicting benchmark scores at scale. We predict a large model's benchmark scores in two stages (Figure 15), each fitted only on runs that are cheap enough to repeat many times. Chaining the two stages yields a predicted score for every combination of mix, model size, and benchmark.

Stage 1: validation-loss scaling laws. For each candidate mix, we trained a grid of 33 small models spanning 36M to 1.5B parameters at 10 to 60 tokens per parameter, which gives 0.36B to 59B training tokens. We measured each checkpoint's loss on a battery of held-out validation sets drawn from our data categories. Losses are measured in bits per byte, $\log_2(\text{PPL})$ divided by the tokenizer's bytes per token, so they are

comparable across tokenizers. For each mix and validation set we fit a scaling law in parameters P and training tokens N (Hoffmann et al., 2022; Li et al., 2025b). The fitted form is log-linear or additive in $\log P$ and $\log N$, chosen per validation set. We then evaluated each law at the planned production configurations of 3.6B, 7.0B, and 31.2B parameters trained on 20T tokens. These targets lie about 20× beyond the largest fitted model and about 340× beyond the largest fitted token count.

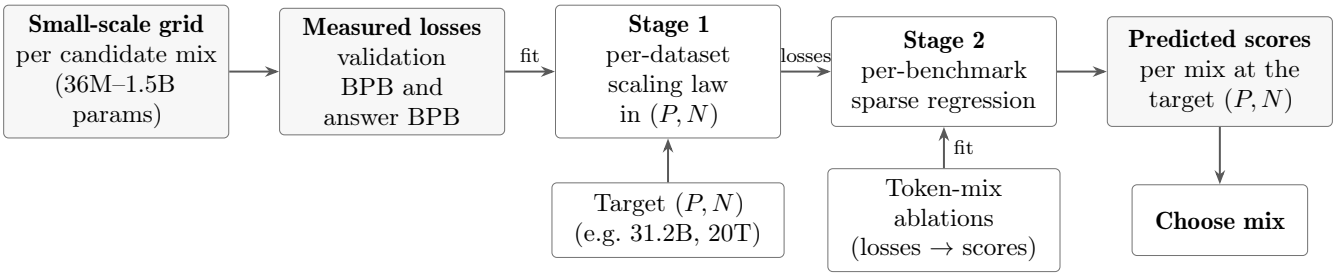


Figure 15: Predicting benchmark scores of large models from small-scale runs. Stage 1 fits a scaling law to each validation set's loss over a grid of small models and evaluates it at the target model size P and token budget N. Stage 2 maps losses to each benchmark's score with a sparse regression fitted on separate ablation runs, and is applied to the extrapolated losses. Shaded boxes are data; unshaded boxes are fitted models or inputs to them.

Stage 2: from losses to benchmark scores. For each of 13 benchmarks (AIME 2025, ARC-Challenge, BBH, GSM8K, HellaSwag, HumanEval, MATH-500, MBPP, MMLU, MMLU-Pro, PIQA, TruthfulQA, and WinoGrande), we fit a sparse linear model, LARS-based Lasso with a cross-validated penalty, that maps checkpoint losses to the benchmark score. The features are the losses on ten validation sets plus the benchmark's *answer loss*: the model's loss on the answers it generates for that benchmark's questions. The answer loss measures how surprised the model is by its own outputs and is a cheap proxy for whether it knows the answer. The Stage 2 models are fitted on checkpoints from our earlier token-mix ablations of up to 7B parameters, which exclude the three candidates, and are then applied to the Stage 1 extrapolated losses. The approach follows prior work that predicts downstream performance through intermediate losses (Gadre et al., 2024; Bhagia et al., 2024).

Measured and predicted rankings disagree. Table 17 contrasts the two views of the candidates. Measured on extended 1.5B training runs, the SFT-heavy mix is the best of the three on the 13-benchmark mean and the web-heavy mix is the worst. The SFT-heavy mix leads at every evaluated checkpoint of those runs, so its lead is not a single-checkpoint artifact. Had we chosen based on measured behavior at the largest scale we had run, we would have picked the SFT-heavy mix. The predictions reverse this ordering. At every predicted size, the web-heavy mix ranks first and the SFT-heavy mix last, and the predicted curves are close to parallel, so the ordering does not depend on which target size we compare at. At the 31.2B configuration, the web-heavy mix is predicted to be best on 11 of the 13 benchmarks. The math-heavy mix is predicted to be best on the other two, MATH-500 and AIME 2025, and the SFT-heavy mix is predicted to be worst on every benchmark except those two. We followed the prediction and adopted the web-heavy mix.

	Measured	**Predicted** (20T tokens)		
Mix	1.5B	3.6B	7.0B	31.2B
Web-heavy	36.6	**51.8**	**53.6**	**57.4**
Math-heavy	38.8	51.3	53.1	56.7
SFT-heavy	**40.5**	49.6	51.3	55.1

Table 17: Mean score over 13 benchmarks per candidate mix. *Measured*: final checkpoint of a 1.5B run per mix. *Predicted*: the two-stage procedure in Figure 15 at the planned production configurations. Token budgets differ, so compare orderings, not levels, across the two column groups. Bold marks the best mix per column.

Production mix. Our production mixes descend from the web-heavy candidate (Figure 16). The mix for the dense 3.7B, 7B, and 32B models and the 36B-A4B model keeps the web-heavy candidate's category balance. It moves about four points of the web share to multilingual and math data. It also broadens the multilingual budget beyond Arabic to French, Hindi, further languages, and translated question–answer data (§4.7). The phase-2 mix of the 375B model raises the web share further and moves weight from source code to code-reasoning data. The phase-2 mix was chosen so that the overall mix seen by the 375B matches the final mix used for the other models.[9]

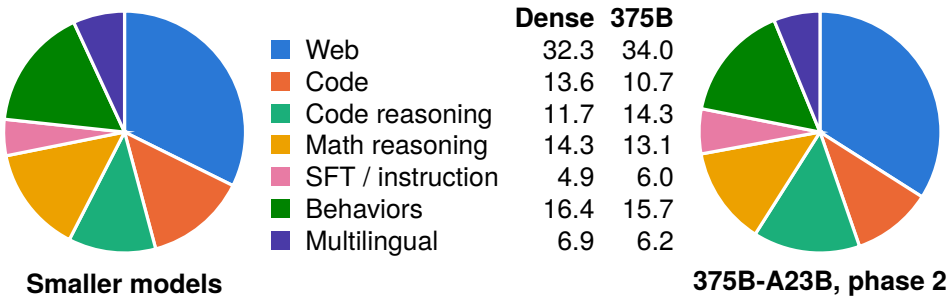


Figure 16: Production pre-training mixes by category (% of training tokens), using the categories of Table 16. Slices run clockwise from the top in legend order. Left: the mix used for the dense 3.7B, 7B, and 32B models and the 36B-A4B model. Right: the second-phase mix of the 375B-A23B model. Shares sum to 100 up to rounding.

Choose the mix for the scale you train

The mix that was best in our 1.5B ablations is predicted to be the worst of the three at 32B. Comparing mixes on predicted rather than measured scores reversed the decision.

4.10 Tokenizer

The model uses a standard 256K byte-level BPE tokenizer (64K for 0.9B model) trained from scratch on a custom multilingual and code-focused corpus of approximately 25.2 GB. The training mixture contains about 9.9 GB of English, 5.9 GB of Arabic, 4.2 GB of code and structured text, and additional data spanning Hindi, French, Bengali, Chinese, German, Persian, Filipino, Indonesian, Italian, Japanese, Malayalam, Portuguese, Russian, Spanish, Turkish, and Urdu. The comparatively large Arabic allocation, roughly 24% of the on-disk corpus before including Persian and Urdu, was designed to provide substantially stronger Arabic coverage than general-purpose multilingual tokenizers. Code data includes general source code, JSON, Python, SQL, shell, HTML, and TeX. When tuning the tokenizer training mix, we measured fertility on our own code, TeX files from prior work, and synthesized examples of downstream use cases. This led us to up-weight JSON and Python data, where our first tokenizer was less efficient than other open-model tokenizers.

Text is NFC-normalized and processed with a Qwen-style Unicode-aware pre-tokenizer followed by byte-level BPE, including segmentation of long numerical strings into groups of up to three digits. The complete byte alphabet provides open-vocabulary coverage without an unknown token. The tokenizer includes BOS/EOS tokens plus markers for fill-in-the-middle coding, repository and notebook metadata, and tables. The 256K BPE model (64K for 0.9B model) is followed by 256 special-token IDs, comprising 50 functional markers and 206 reserved slots.

[9]The 375B run started before all data pipelines were finalized.

5 Mid- and Post-Training data

Our mid- and post-training data targets long context and agentic ability. Unlike common practice, we do not separate these stages explicitly: agentic data enters as soon as context extension begins (§3.1.3), mixed with long-context pre-training data, agentic data in both pre-training and chat formats, and additional instruction-following and reasoning data. We developed many task generation pipelines and collected public post-training datasets (§5.18). Together, they provide 106M unique documents and trajectories with 1.17T tokens, with 75% of the trajectories coming from our own pipelines. The data carries over the philosophy of our pre-training data (§4): diversity first, grounding in real data, and no verification in many pipelines.

Grounding in real sources. Beyond Search360 (§4.5), we use the Keenable web search API[10] to find relevant sources such as repositories, office files, and papers, with about 5M requests supporting task synthesis. We also curate specialized collections of code and papers, which we query with simple keyword search when creating tasks. Our main curated source is PRism, a collection of GitHub pull requests (§5.1). Besides fueling four pipelines for agentic coding data (Figure 17), PRism turned out to be a rich source of office files: its repositories contain 1.3M PDF, Word, Excel, and PowerPoint files, which we use to synthesize knowledge-work and file manipulation tasks (§5.9.2).

Diversity and strong teachers. Diversity remains our primary focus, but a strong teacher is also essential. In SFT (§6.6), we used the then newly released Kimi K3 (Kimi Team, 2026b) and DeepSeek-V4-Flash (DeepSeek-AI, 2026) for both task synthesis and trajectory generation, which improved the performance of our model. A teacher is limited by its own abilities, but the training data we generate does not have to be. We developed generation techniques that produce trajectories beyond what the teacher can do in a single rollout (§5.16). Multi-turn PR chains replay the PRs of a repository as consecutive requests in one coding session: the teacher handles one PR at a time, while the training trajectory captures the entire session and reaches hundreds of thousands of tokens (§5.16.3). A privileged advisor uses the reference solution and hidden tests to find the solver's mistakes and inserts targeted cues into its reasoning, without revealing the solution. Rerunning the tasks the solver failed with an advisor raises the share of tasks with a successful trajectory from 54% to 73% (§5.16.4). These techniques require extra compute for each trajectory, so we reserve them for the hardest tasks.

AI for Science. A particular focus of our data is AI for Science (§5.12): tasks and environments built from research papers, a simulated research community, Kaggle competitions, scientific simulators, and proofs from machine learning theory.

Quality without verification. As in pre-training, many of these pipelines forgo verifying the final answer (Figure 12). Instead, every trajectory must pass strict programmatic checks of format and behavior (§5.17). Below, we describe our task generation pipelines organized by capability, followed by how we ensure the quality of the training trajectories.

Behaviors before correctness

Much of the data we used is unverified. We think it still teaches useful behavioral patterns, such as exploring a repository, planning, and recovering from errors. Throughout mid-training and SFT, we trained on all trajectories, with only occasional correctness filtering, and replayed the correct ones during the final training phase.

[10] https://keenable.ai/

5.1 PRism Data Processing

We construct a corpus of GitHub pull requests (PRs) that combines development discussions with the corresponding code changes and repository context. The pipeline comprises three components: PRism-curator reconstructs and filters PR records, PRism-annotator adds structured semantic labels, and PRism-synthesis transforms the resulting data into synthetic reasoning documents, coding tasks, executable tasks for reinforcement learning, and multi-turn agent trajectories. This section describes curation and annotation; synthesis is covered separately. Figure 17 summarizes the data flow.

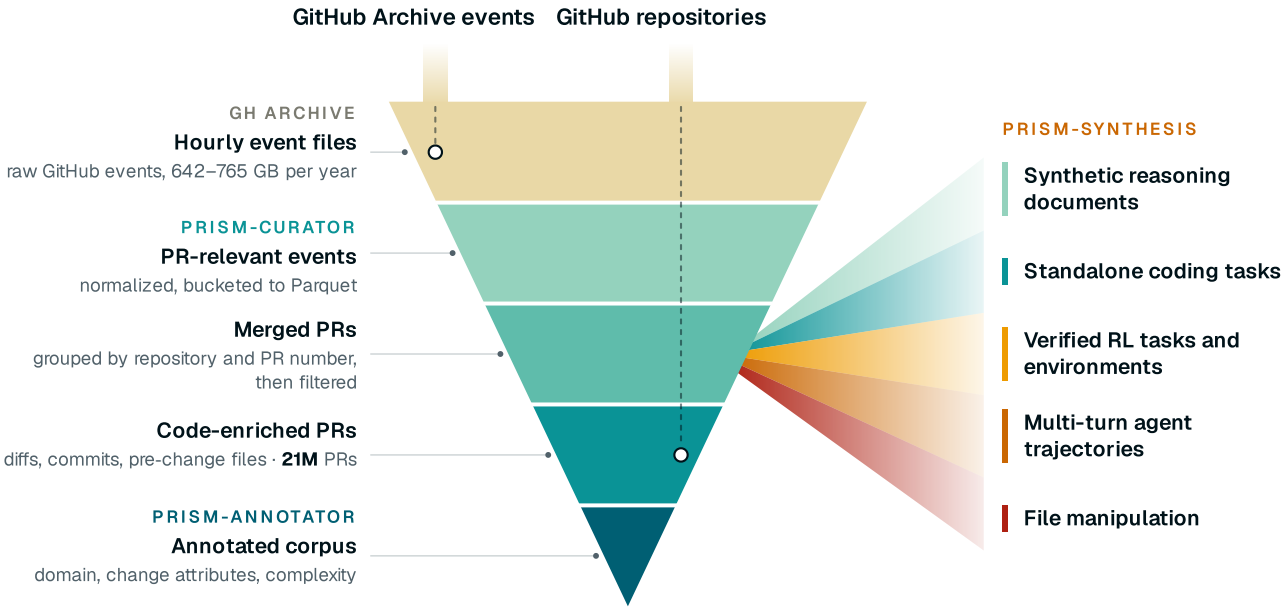


Figure 17: Overview of PRism data processing. Curation combines archived events with repository code, and annotation adds semantic labels. The resulting corpus provides inputs for five synthesis branches. The prism is schematic; band widths are not to scale.

5.2 PRism Data Curation

Event collection and reconstruction. We process GitHub Archive events and retain information relevant to PRs, including PR updates, comments, reviews, pushes, and continuous integration events. Events are normalized and partitioned by repository identifier for parallel processing. We then group records by repository identifier and PR number, associating discussions and related events through PR references and commit identifiers. Each reconstructed record preserves the PR metadata, participants, available description revisions, and associated events.

Selection criteria. We retain merged PRs targeting the `main` or `master` branch, with descriptions longer than ten characters after whitespace removal and changes to between one and fifteen files. PR descriptions may contain at most one distinct issue-style reference of the form `#123`. A keyword filter excludes PRs whose titles or descriptions indicate dependency or version updates. We also require an identifiable landed commit, obtained from the recorded merge commit or inferred from push events. These criteria select changes with bounded scope and accompanying textual context.

Code enrichment. For each retained PR, we retrieve the repository history and resolve the base and head revisions. Where available, the merge base defines the starting revision for extracting the change. We

collect the combined diff, individual commit patches, changed file paths, repository file list, and pre-change contents of the affected files. Extraction is subject to configurable limits on file counts and content size; truncation and retrieval errors are recorded. The enriched records are stored in Parquet format, preserving the original PR fields and adding the extracted code context. This representation links the description and discussion of a change to its implementation and initial repository state.

5.3 PRism Annotation

Annotation dimensions. We use a language model with a fixed rubric to annotate each PR along three dimensions. The domain label identifies the principal technical area, such as data engineering, security, or compiler development. Change attributes provide multiple labels for properties such as bug fixes, feature additions, refactoring, and the inclusion of tests. The complexity label describes the predominant reasoning requirement: knowledge of frameworks or conventions, multi-step logical reasoning, or recognition of a subtle behavior or edge case. These categories describe the nature of the task rather than an ordinal difficulty level.

Evidence and label assignment. The annotation prompt includes PR metadata, discussion, repository information, and extracted code context. The rubric prioritizes code evidence, followed by PR text and labels, repository metadata, and discussion or continuous integration metadata. The model is instructed to score each candidate label against explicit criteria and return the selected labels, scores, and short rationales. Domain and complexity annotations allow a `NONE` label when evidence is insufficient, while change attributes may be empty. The prompt also specifies consistency rules for conflicting attributes.

Structured output and provenance. Annotations are generated through vLLM with temperature zero and a JSON schema constraint. Long inputs are truncated to fit the available context, and the original and retained prompt lengths are recorded. PRs with code-retrieval errors remain in the dataset but are left unannotated. The output preserves all input columns and appends the annotations, parsing status, and prompt truncation metadata. The resulting corpus retains the source evidence alongside its semantic labels, supporting subsequent analysis and preparation of inputs for the synthesis pipelines.

Example record. Figure 18 shows an abbreviated annotated PR record. The PR fixes game behavior and adds sound effects; its annotations identify both bug fixes and feature additions. The complexity label is `NONE`, indicating insufficient evidence for a specific reasoning category.

5.4 Developer thoughts synthesis

PR data records the detailed history of a repository but not the reasoning behind each change. To mitigate this gap, we turn human-written code changes into long-context agentic data in text format by asking an LLM to synthesize only the missing thoughts, while all the code comes from the PR itself.

Synthesis. We use about 21M curated PRs from 2020 to 2024 (§5.2). For every PR, we show its metadata, discussion, files before the change, and commit diffs to Qwen3.5-397B-A17B (Qwen Team, 2026) and ask it to write three kinds of prose: a task description, the developer's reasoning before each commit (or before each file within a commit), and a closing summary. We skip PRs with code-retrieval errors or whose prompt would exceed 32k tokens. We then splice the prose around the real repository tree, files before the change, and commit diffs. The result reads like an agentic coding session in text format: a task, the relevant files, and a sequence of reasoning steps, each followed by an actual code edit. All code in the document is real; only the thoughts are synthetic.

```
repo_id: 847236596
pr_number: 33
pull_request:
  title: "Fix: qol"
  merged: true
  base.ref: main
  changed_files: 8
  commits: 6
code:
  repo_full_name: dice-2004/Union-Hack-2024
  status: ok
  touched_files: [src/battle.py, src/enemy.py, ...]
  compare_diff: |
    --- a/src/battle.py
    +++ b/src/battle.py
    ...
    -                pl.lvup_check()
    +                pl.lvup_check(sounds)
annotation:
  domain.label: GAMES_SIMULATION_MISC_CRAFT
  attributes.labels: [BUGFIX, FEATURE_ADD]
  complexity.label: NONE
prompt_was_truncated: false
```

Figure 18: Abbreviated annotated PR record. Selected fields are shown in a compact notation; ellipses mark omitted paths and diff context. Full records also contain base files, commit patches, events, annotation scores, and rationales.

Diversity knobs. Applying a single prompt and a single document template to millions of PRs would produce highly repetitive data. We therefore draw knobs for every PR that control both what the LLM writes and how the final document is rendered. Prompt knobs set the style of the task description (GitHub issue, direct instruction, conversational, or specification), its level of detail, whether it includes a plan or names the files to change, and whether reasoning is written per commit or per file. Rendering knobs set the diff format (unified, search/replace, or full file), what is shown at the end (combined diff, files after the change, or both), line numbers and their style, how files are delimited, truncation of long files, whether files are shown upfront or right before the commit that touches them, whether and how the repository tree is shown, where the repository is mentioned, and the connector phrases between sections. Since rendering knobs are applied after generation, the same LLM output can be turned into many different documents.

Long context. Documents over 128k tokens are re-rendered more compactly (no files after the change, harder truncation of long files), and documents over 512k tokens are dropped. The resulting 14.6M documents (181B tokens) cover a wide range of lengths: 8.8M documents are under 8k tokens, while over 1M documents are between 32k and 512k tokens.

5.5 PRism task synthesis

PRs are a natural source of realistic coding tasks for agents. We turn curated PRs into standalone tasks that can be solved on the repository before the change, wrap them into instructions for a coding agent, and collect agent trajectories solving them.

Task synthesis. Starting from the curated PRs (§5.2), we give GPT-OSS-120B with high reasoning effort (Agarwal et al., 2025) the PR metadata, commit subjects, combined diff, files before the change, and, optionally, the discussion comments, and ask it to write the task that the PR solves as an independent assignment, without referencing the PR or its diff. We skip PRs whose touched files exceed 5,000 lines

in total, as well as prompts over 32k tokens. The task is then combined with general instructions for an agent working in the repository, including language-specific setup hints based on the primary programming language of the repository.

Difficulty control. We do not verify the solutions produced by agents on these tasks, so controlling task difficulty is important: tasks that are too hard are likely to result in incorrect solutions, while tasks that are too easy teach little. The main knob here is the amount of hints, with five levels: bare (a one- to three-sentence request, as typed in a chat), concise (a title and a short description), standard (adds pointers to environment setup files and a high-level plan), guided (adds a step-by-step plan and testing hints), and detailed (adds pointers to the files, functions, and design decisions). We sample the level based on the size of the PR, measured by the number of commits, diff lines, and touched files. Small PRs mostly get few hints (bare or concise 65% of the time), while large PRs mostly get many (guided or detailed 65% of the time). Every level appears at every PR size, but overall the distribution is biased towards balancing difficulty: large changes become more approachable with more guidance, while small changes stay realistic and underspecified. The agent instructions follow the opposite pattern: the fewer hints a task contains, the more detailed the instructions and the stricter the requested verification.

Diversity knobs. Other knobs vary the style of the task (direct instruction, GitHub issue, or specification), the persona (direct request or a project maintainer guiding a contributor), the scope (the minimal targeted change or open to restructuring related code), whether the PR discussion is shown to the LLM, and how the task is rendered (Markdown, plain text, or conversational, with varied headers, separators, and section names). The agent instructions vary in verbosity, workflow (phases, checklist, or free-form), verification strictness, guidance on honestly reporting problems, and response format.

Agent trajectories. We run Qwen3.5-397B-A17B with thinking (Qwen Team, 2026) in the OpenCode agent harness on the synthesized tasks, each in a sandbox with the repository at its state before the change. The agent explores the repository, sets up the environment, edits code, and runs tests using tools such as `bash`, `read`, and `edit`. The resulting 2.3M trajectories (70B tokens) are used in mid-training.

Don't forget sandbox diversity

We put a lot of effort into task diversity but overlooked the sandboxes: in every trajectory, the repository was located at `/workspace`. The model learned this too well. In evaluations where the repository was elsewhere, it kept looking for `/workspace`; tasks that did not state the working directory took twice as many commands, and the pass rate fell by 40%. Starting from the fourth mid-training stage, we use an augmented version of the trajectories with varied working directories and a programmatically inserted turn imitating an agent checking where it is.

5.6 PRism RL task synthesis

We use the same PR corpus as mentioned in §5.2 to create agentic coding tasks for RL training. Each task needs an environment in which the agent's solution can be run and verified. To build them, we filter the corpus, rewrite each remaining PR into a task, synthesize an environment for it, and filter the environments with a judge that inspects solver trajectories (Figure 19).

Task design. A task has four parts: the repository at the commit before the PR, an instruction that describes the change without revealing the solution, a sandbox in which the project's tests run, and a verifier that decides whether the agent succeeded. The PR also gives us a reference solution (fix patch)

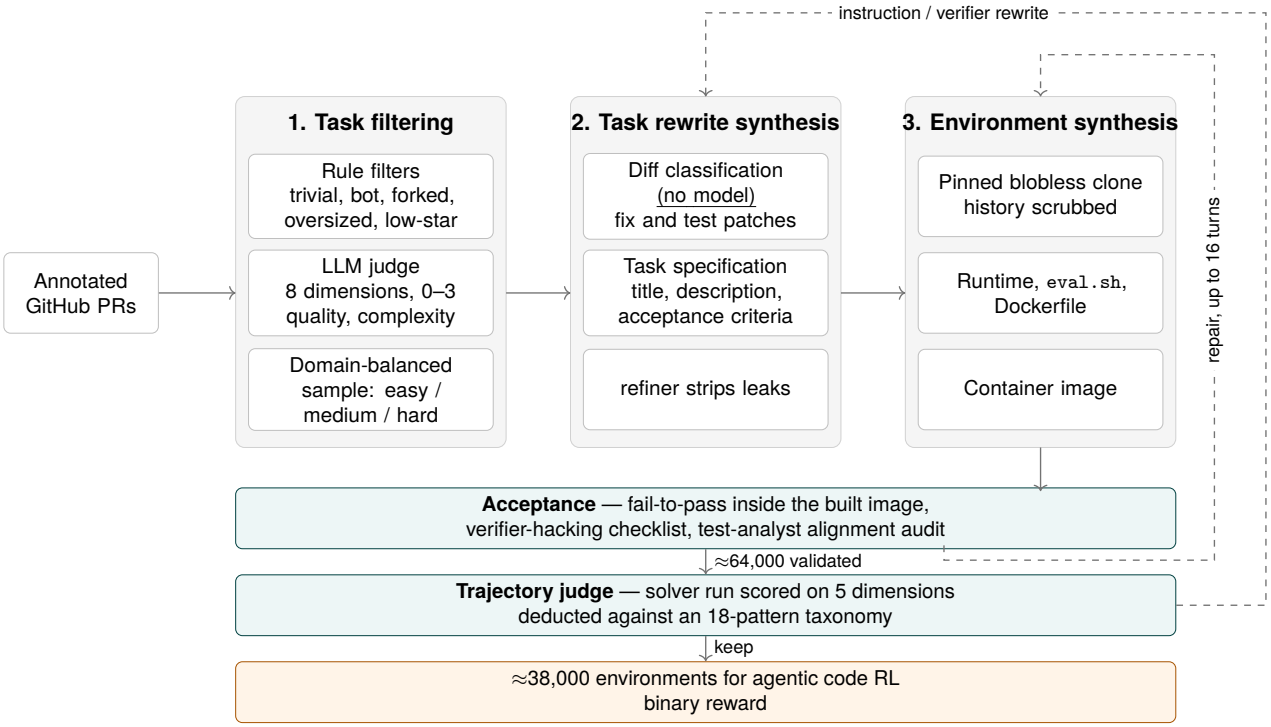


Figure 19: Building RL coding environments from GitHub pull requests. Filtering reduces the annotated PR corpus with deterministic rules, then scores survivors with an LLM judge and draws a domain-balanced sample stratified by difficulty. Rewriting derives the reference patches from the diff without a model and uses the model only for the task specification, the predicted environment, and the test commands. Environment synthesis is an agent loop that produces a container image. The two checks below it are distinct: acceptance is mechanical, whereas the trajectory judge runs a solver against an already-accepted environment and decides whether it is worth training on. Solid arrows carry artifacts and dashed arrows carry feedback: acceptance failures return to the responsible sub-agent, and judged misalignment returns to the rewrite stage. Filter thresholds, the eight scoring dimensions of the filtering judge, the checklist items, and the selection rule are given in the text.

which we use to check the verifier but never show to the agent. A good task needs a verifier that fails on the original code and passes with the reference solution. The task also has to be solvable, so that a failed attempt reflects its difficulty rather than a broken environment. The first property can be checked deterministically; the second needs an LLM judge and static checks.

Task filtering and rewriting. Most PRs cannot be turned into a usable task, so we drop PRs annotated as documentation-only, version bumps, reformatting, code regeneration, or refactoring, unless they also contain a functional change. We also drop bot PRs, diffs over 200k characters, binary-only diffs, PRs against forks, PR bodies under 100 characters, and repositories with fewer than six stars or no package manifest. GPT-OSS-120B (Agarwal et al., 2025) then scores the remaining PRs on eight dimensions, and the quality score weights content strength 0.60 and environment feasibility 0.40. Finally, we sample a task set balanced across 17 domains and label each PR as easy, medium, or hard relative to PRs with the same star bucket, domain, and language.

A merged PR already contains its answer, so we rewrite it into a specification of what to implement, with the level of detail set by the difficulty knobs. The reference patches come directly from the diff against the merge base which we split by file into a `fix_patch` and a `test_patch`. An LLM writes the title, description, acceptance criteria, and test commands, and a second pass compares the draft with the test patch to remove identifiers leaked from the fix and claims the tests do not check.

Diff against the merge base

The base commit recorded on a PR is not always the commit the change was written against. The commit that matches the diff is the merge base, the common ancestor of the PR's base and head branches. The two differ in about 15% of PRs because of branch drift.

Environment synthesis. A task also needs an environment to run and a verifier we can trust. An agent using the OpenCode harness based on Qwen3.5-397B-A17B (Qwen Team, 2026) builds both, delegating to sub-agents that explore the repository, set up the runtime, write and audit the evaluation script, and build the Dockerfile. It works in a blobless clone at the merge base with all git history and references removed. It installs dependencies with the project's own package manager and checks with a dry run that the tests can be collected; it has at most 16 turns to finish. Hidden tests stay outside the working tree. An environment is accepted only if it is fail-to-pass: the evaluation script fails on the original repository and passes with the reference patch applied, both run inside the built image. Because the same agent writes the environment and its verifier, a test-analyst sub-agent and static checks audit the verifier before acceptance. Accepted environments are exported in Harbor format (`task.toml`, `instruction.md`, `environment/`, `tests/`, and `solution/`) with a binary reward.

Judge filtering. Passing fail-to-pass does not mean that a task is useful for training. We therefore run a solver on each accepted environment and judge its trajectory on five dimensions, including environment readiness, dependency completeness, and verifier integrity. The judge first identifies which of 18 failure patterns occur (e.g., a runtime that never starts, a missing package, tests that never run, or a solver that searches without editing), and lowers a dimension's score only when it has identified a matching pattern. A deterministic rule then maps the scores to quality buckets like keep, downweight, or reject. When the judge flags a mismatch between the instruction and the verifier, we repair the task instead of discarding it: a faulty verifier is realigned with the instruction, and an unclear instruction is rewritten from the solver trajectory, with up to two candidates that are checked for leakage and accepted only if a new solver run scores higher. With this approach, we created roughly 64k environments that passed validation, of which 38k were rated high quality and used for RL training.

Watch out for verifier hacking

When an agent writes both the environment and the grader, passing fail-to-pass tests becomes deceptively easy. A script that exits 0 is much cheaper to write than one that runs a real test suite, and the synthesis agent often took that shortcut in our initial runs. We therefore added regex checks that catch verifiers that only check that the patch files exist instead of running tests, grepping the reference patches, using `git diff` as an oracle, or throwing away the test command's exit code. Agents get around these static checks with inline scripts and custom commands. We therefore add a test-analyst LLM judge that reads the verifier source and its execution trace. It checks that the script calls a recognized test runner and its result comes from real assertions, and that it gives different results on the clean and the patched tree.

5.7 Subagent delegation

Long coding tasks require an agent to spend context on repository exploration, file contents, command output, and test results. To study whether delegation can improve this process, we collect trajectories in which a parent agent may assign one bounded piece of work to a child agent. Each child runs in a

fresh OpenCode session and returns a final report to the parent, which then checks the report and decides whether to use it. The complete parent and child sessions are retained separately.

Tasks. We start from Harbor coding environments derived from merged pull requests. The original sandbox and verifier are left unchanged. Kimi K3 is used for task-instruction synthesis. Given a digest of the original change, it writes a short description of the required behavior, the shared interface between the workstreams, and several pieces of work that can be developed and checked independently. A fixed template then asks the solver to validate these workstreams and integrate the result before finishing. The instruction synthesizer produces prose only. We reject drafts that mention files from the reference patch, copy patch text, or explicitly instruct the solver to delegate or use subagents. This results in 908 tasks.

Rollouts. We run Kimi K3 with thinking enabled at maximum reasoning effort in the OpenCode harness (Anomaly, 2026), orchestrated by our rollout engine, with Harbor executing the sandbox and verifier. The parent agent has the usual repository tools and a `task` tool that can launch one of two specialists in a fresh context. The investigator traces repository behavior and answers a focused question; the implementer performs one bounded change. The parent prompt describes these roles and asks the parent to verify load-bearing details in a child's report before relying on them. Delegation is optional: the task instruction does not require a child call, and recursive delegation is disabled. Parent and child tool calls are executed within the same Harbor task environment, while their OpenCode sessions and trajectories remain separate.

We retain valid four-hour rollouts together with eight additional valid trials from the two-hour run. Of the 908 tasks, 850 produced at least one valid trajectory, while 58 failed both attempts. Harbor evaluates each trial with the original task verifier.

Training records. Each parent and child session is serialized as a separate training trajectory. The parent trajectory contains the `task` call and the report returned by the child; the child document preserves the detailed work performed in the child session. A trial whose Harbor verifier returns reward 1 is labeled correct. Child labels are inherited from the parent trial and therefore are parent-level labels rather than independent child-verifier judgments.

The four sources then pass through the shared SFT validator (§5.17). Correct documents allow five consecutive tool errors, while other documents allow three. The resulting collection contains 3,041 documents and 131.6M tokens (Table 18).

Source	Documents	Tokens
Parent, verified	618	45.5M
Parent, other	290	21.6M
Child, parent-verified	1,349	37.9M
Child, parent-other	784	26.5M
Total	3,041	131.6M

Table 18: Parent and child trajectories retained for SFT. Child labels are inherited from the parent trial's verifier outcome, not from independent child-level judgments.

5.8 SWE Efficiency

Our SWE Efficiency pipeline, inspired by Ma et al. (2025), transforms real-world, performance-oriented pull requests into reproducible optimization tasks and agentic engineering trajectories. Curation identifies suitable changes, prepares executable environments, and establishes correctness checks and representative performance workloads. During generation, coding agents investigate the repository, analyze behavioral invariants and compatibility requirements, develop optimizations, and iteratively validate their changes through tool execution and feedback. The emphasis is on evidence-grounded engineering: proposed improvements must preserve required behavior, and performance claims must be supported by measurements. We retain interaction histories and execution evidence to enable traceability, quality assessment, and subsequent auditing. Reviews consider functional

correctness, measured performance, and consistency between the agent's explanations and observed results, with accepted trajectories maintained separately from lower-confidence records. With this pipeline, we synthesized 700 SFT-format agentic traces.

5.9 File Manipulation

We built two pipelines producing teacher trajectories for office work with files such as spreadsheets, documents, slide decks, PDFs, HTML pages and technical drawings. They are inspired by GDPval (Patwardhan et al., 2025). A task consists of optional input files and an instruction, usually an e-mail from a manager or colleague or an assignment brief, that asks for one or more files to be created or edited, or for a written answer drawn from the input files.

Both pipelines start each task from real files. They differ in where the files come from: the first collects public documents from the web (§5.9.1), and the second mines office files from the repositories of our PR dataset (§5.9.2).

5.9.1 Tasks from Public Documents

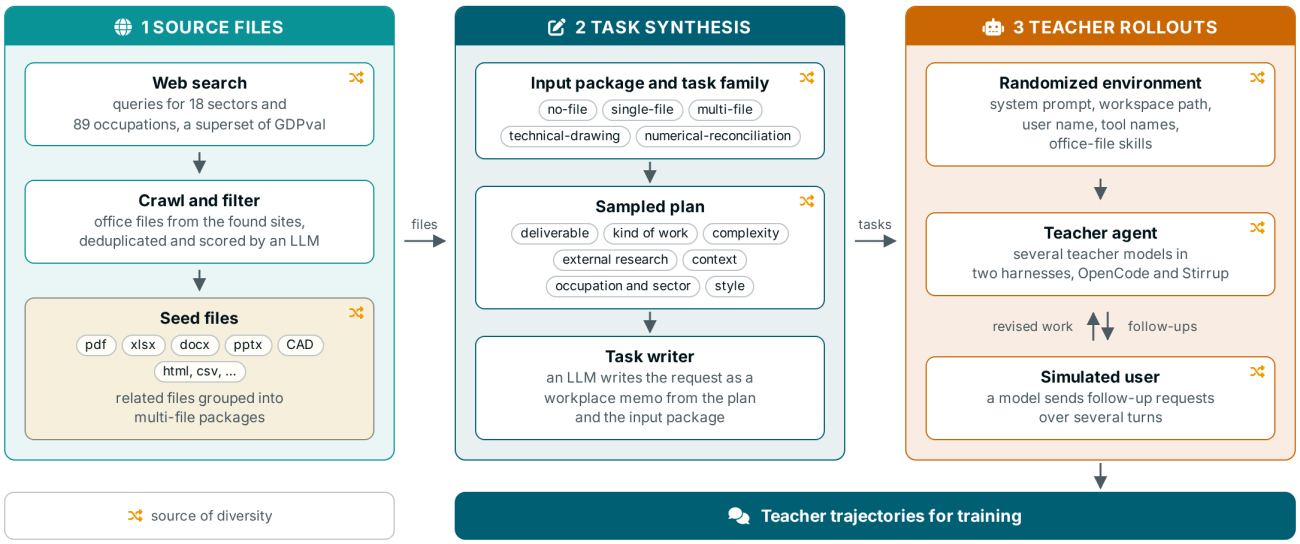


Figure 20: File manipulation pipeline from public documents. Source files are found by web searches over sectors and occupations, then crawled and filtered. Each task combines an input package, a task family and a sampled plan, from which an LLM writes the request. Teacher models solve the tasks in two harnesses, with a randomized environment and, in some rollouts, a simulated user.

Source files. The source files are a pool of documents scraped from public sources. We used 18 NAICS sectors and 89 O*NET occupations that form a superset of those included in GDPval. We then create search queries and use them to find domains that would publish documents related to each sector/occupation (e.g., "a trucking association publishing freight rate tables"). We then crawled such domains for files with office-like extensions (e.g., pptx, xlsx, docx, and pdf). After basic filtering and deduplication, we used a Qwen3.6-27B model to score each file across 6 weighted criteria, including whether the file could be the input of a realistic professional task and how well it fits the selected sectors and occupations. The same model also extracts metadata (identifiers such as case numbers, entities such as project names, organizations, and topics from a generated taxonomy), which we use to group related files into multi-file packages. Finally, we use the scores to filter files based on a threshold, leaving approximately 115k files.

File type	Seed files	Share
pdf	66,395	57.7%
xlsx	19,597	17.0%
docx	12,872	11.2%
pptx	3,596	3.1%
CAD (STEP)	≈2,300	2.0%
Other (HTML, XML, CSV, plain text)	≈10,240	8.9%
Total	115,000	100%

Table 19: Public files used as seeds, by file type.

Task synthesis. We roughly follow these four steps to synthesize a task from the pool of documents. First, we sample an input package containing either a single, multiple, or no files. Second, we sample a plan along the diversity dimensions described below. Third, an agent using DeepSeek-V4-Flash receives this plan and the input package to produce a plain-prose request written as an internal memo addressed to a worker mentioning the situation, the inputs and a sentence listing every file to be delivered. We then create a task folder with the input package, the generated request and the drawn plan.

To diversify the tasks, we sample each plan along the following dimensions:

- **Deliverable:** the file format and the number of files.
- **Kind of work:** produce, analyze, decide, verify or transform.
- **Complexity:** the reasoning depth and the quantitative level.
- **External research:** how much outside research the task needs.
- **Context:** the stakes, the audience and the register of the request.
- **Occupation and sector:** drawn from a list of 358 titles, favoring under-covered sectors.
- **Style:** how the request opens, the kind of employer, and a naming tradition for any people in it.

Each task also belongs to one of five families. No-file, single-file, and multi-file tasks come with zero, one, or several input files. Technical-drawing tasks list a physical inventory in full (e.g., equipment, rooms, or wiring) and ask for a plan, schematic, or layout as a PDF, optionally with SVG or STEP files. Numerical-reconciliation tasks ask for named values such as totals, variances, or converted amounts, computed from 3–6 figures that code extracts from the input file or places in the request. Each value's direction, unit, and rounding are stated so that it has exactly one correct answer.

Teacher rollouts. We solve the tasks with several teacher models, including Kimi K3/2.6 and DeepSeek-V4-Flash, in different agent harnesses, such as OpenCode and Stirrup, the open-source harness behind GDPval-AA. We also randomize the environment of each rollout: the system prompt is rewritten or paraphrased, the workspace path, the user name, and the tool and argument names vary across trajectories, and some rollouts give the teacher a set of office-file skills. In some rollouts, a model plays the user and sends follow-up requests, so the teacher must revise its work over several turns.

Teacher	Traj.	Tokens
Kimi K2.6	63.6k	3.5B
Kimi K3	220.0k	17.7B
DeepSeek-V4-Flash	45.6k	7.0B
Total	329.2k	28.2B

Table 20: Trajectories and tokens per teacher model.

5.9.2 Tasks from Repository Files

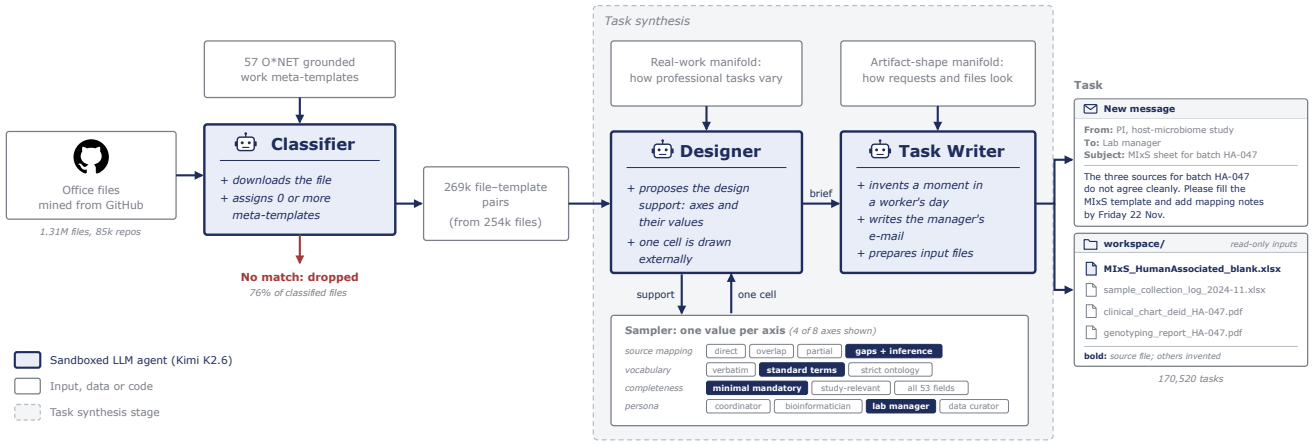


Figure 21: File manipulation pipeline from GitHub repositories: from a file to a task.

Source files. We mine input files from the repositories of our PR dataset (§5.2), which yields 1.3M office files (PDF, Word, Excel and PowerPoint) across 85k repositories. An agent downloads each file and matches it against a taxonomy of 57 work meta-templates grounded on O*NET occupations. Each template describes a cluster of economically valuable office tasks and the input files they need. Of the 1.29M files we downloaded and the 1.08M we classified, 254k (23.6%) match at least one meta-template, giving 269k file–template pairs, of which 170,520 became tasks.

Task synthesis. The main objective during task synthesis was to create a distribution as wide as possible while keeping economically valuable office work at its center. To achieve this, we split task synthesis into two agent runs: one designs the task, and the other writes it. To avoid diversity collapse on each step we anchor the agents with long form description files that avoid details and examples, and we use an external sampler to draw an instantiation of a variety of knobs proposed by the agents themselves.

The anchors are two manifold documents, long prose files generated from a representative set of economically valuable tasks. They describe office work at the population level and contain no example tasks or details. The first, real-work manifold, describes how real tasks vary: how the deliverable relates to the input files, how much of the answer is already in the inputs and how much must be constructed, where ambiguity lives, and whether multiple inputs agree or conflict. The second, artifact-shape manifold, describes what requests and files look like, from how managers phrase instructions to the operational mess of real files.

We used Kimi K2.6 for both agent runs. In the design run, the agent receives the source file, the meta-template, the real-work manifold and proposes a design space: the axes along which a task built from this file could vary, where we encourage behavioral diversity over cosmetic diversity. An external sampler is used to draw an instantiation from the proposed design space, so the agent defines the support but does not choose the draw (Gu et al., 2026). With the draw, the agent then writes a brief describing what the draw means for this file. In the writing run, a new agent reads the file, the meta-template, the design brief and the artifact-shape manifold, and creates the task.

Synthesized tasks take the shape of a manager's e-mail plus the agent's starting workspace: the source file (potentially modified) and, in a third of the tasks, synthesized companion files to create multi-file tasks.

Teacher rollouts. We solve each synthesized task once with Kimi K2.6 in our fork of the forge harness. Before the rollout, for 95% of the tasks, a short-lived agent proposes several system-prompt styles, an external sampler picks one, and the agent then writes the system prompt in that style. We also include the traces from the task synthesis stage as part of the training corpus. Table 21 summarizes the counts created from this pipeline.

Trajectories	Count	Tokens
Rollouts	154,888	11.70B
Task writing	165,657	11.85B
Total	320,545	23.55B

Table 21: Kimi K2.6 trajectories from GitHub repositories, after the checks of §5.17.

Infrastructure. Every agent in this pipeline, from downloading files to the rollouts, runs in our fork of the open-source forge harness,[11] which records a trace of each run and can load its system prompt from a file. Each agent runs inside cowbox,[12] a lightweight sandbox that puts a disposable copy-on-write layer over the machine's file system. The agent can install whatever it needs and cannot see other tasks, and only its own task folder is written back when it exits. Kimi K2.6 is served on our GPU cluster, and the agents run on worker VMs that pull work from a single coordinator and reach the model through SSH tunnels. We release the code as a reference implementation.[13]

5.10 Search

5.10.1 Open-ended research conversations

Many search requests have no single short answer: a user wants to understand a topic, compare options, or see how something works, and expects an account assembled from several sources. To teach this behavior, we generated multi-turn research conversations in which an agent searches our pre-training corpus through Search360 (§5.10), reads documents, and writes an answer grounded in what it found. We call this data Agentic Semantic Packing: length-controlled, search-grounded conversations with tool calls.[14]

Queries. We started from the 1.01M training queries of MS MARCO (Nguyen et al., 2016) and asked an LLM to rewrite 565k of them (good candidates for deep research) in five directions: a follow-up about consequences, a more specific question, a comparison with a related concept, a question about the underlying mechanism, and a question about practical application. This gave 3.83M queries, which we split into disjoint sets, one per target length, so that no query appears at two lengths.

Agent. The agent runs a ReAct loop with two tools: `search`, which returns the top five passages with their document IDs, and `get_document`, which returns a full document truncated to 30k characters. Results with low relevance scores are kept but marked with a note suggesting other search terms, rather than silently dropped. Before a call is executed, a validator rejects repeated queries, near-duplicates of earlier queries (word Jaccard similarity of at least 0.85), and documents that were already read, and returns the reason, so the trace shows the agent trying a different angle instead of looping. Each conversation samples a user persona along four axes with six values each: role (e.g., expert researcher, curious student, skeptical reviewer), interaction style (e.g., formal, Socratic, time-pressed), behavioral

Target length	Trajectories	Tokens
32k	1,322k	38.1B
128k	436k	47.2B
512k	80k	30.8B
Total	1,838k	116.1B

Table 22: Search-based research conversations used in mid-training, by target length.

[11] https://github.com/LLM360/forgecode/tree/981e7ac35d31d03690156d76a3850f526e75a737, the commit pinned in the release.
[12] https://github.com/LLM360/cowbox
[13] https://github.com/ifm-ai/file-manipulation-pipeline
[14] https://github.com/ifm-ai/agentic_semantic_packing

trait (e.g., concise, detail-oriented, big-picture), and research approach (e.g., depth-first, hypothesis-driven, comparative), for 1,296 combinations. The persona shapes both the system prompt and the wording of the user turns. Between research steps, a simulated user writes follow-up messages that refer to specific findings the agent has reported, and occasional nudges (to read a full document, reflect on what has been learned, or rephrase a failed search) are drawn from pools that include the option to say nothing. Each conversation runs until it reaches a target length and ends with a consolidation of the key findings and a final synthesis (§5.16.2).

Filtering. We converted the traces to chat format with tool definitions and removed those with leftover special tokens or channel markers, malformed role sequences, or lengths over budget. A second filter removed traces with garbled tokens (stray non-Latin characters in English prose), empty assistant turns alongside tool calls, refusals, and tool-call markup leaking into assistant text. In our largest generation run, 668,591 of 673,040 traces (99.3%) passed these checks.

Data. We used the conversations in mid-training. They comprise 1.84M trajectories with 116B tokens at three target lengths up to 512k tokens (Table 22).

5.10.2 Multi-hop complex search data

A portion of our search tasks is generated by an automated synthesis pipeline over a local Wikipedia snapshot. Inspired by Gao et al. (2025), we use an e5-base-v2 encoder and a FAISS index of approximately 26 million Wikipedia passages for dense retrieval. We retain only questions that can be answered through the same retriever interface later available to the agent. We first compile the snapshot's inter-article links into an entity graph, then sample multi-hop chains by walking it, biasing each hop towards a more widely linked entity so that the walk terminates at a recognizable anchor. Each edge of a sampled chain is grounded before it is used: we fetch the source article, locate the passage that mentions the next entity, and extract the relation together with a supporting quotation from that passage, so no relation is proposed by a model out of context, and a chain is discarded outright if any one of its edges cannot be grounded this way. The chain is then composed into a question in reverse, from the anchor towards the hidden answer, with every intermediate entity referred to only by its relation to the preceding one, which forces the chain to be resolved sequentially rather than in parallel.

Two further stages harden the item. The first enforces answer uniqueness: we enumerate the entities that satisfy the final relation. Where more than one does, we add attributes that hold for the intended answer and no alternative, while each remaining generic enough to describe many other entities, so the question is pinned down without becoming a fingerprint of its answer. The second replaces the name of each intermediate entity with a description assembled from facts taken from that entity's own article, under a validator that rejects any rewrite in which the name survives. Difficulty is then set by two independent knobs: the number of hops and the number of entities so described. We synthesize five bands, from two hops with no obfuscated entities to six hops with four obfuscated entities, allocating progressively more items to the harder bands. Finally, every candidate passes a two-sided difficulty gate, measured rather than assumed. It is discarded unless a strong model answering from parametric memory alone fails all eight attempts, which establishes that retrieval is necessary, and an agent given the retriever succeeds at least once in eight independent trials while its success rate over the first four trials stays below a per-band ceiling, establishing that the item is solvable but not easy. Items that survive are annotated with the number of tool calls their successful solutions required.

5.11 Tool use

General tool-use data synthesis. To supplement scarce real agentic data, we built a fully automated pipeline that turns a bare domain name (e.g., flight booking, incident triage) into an executable environment

together with verified multi-step tasks. For each domain, a model first designs a relational SQLite schema and populates it layer by layer along foreign-key topology, then writes a Python tool API over that world, so that the agent sees only function signatures and never SQL. The tools are analyzed into a tool-dependency graph whose edges encode data flow (one tool's return field feeds another's required parameter) and state dependency, and tasks are sampled as paths through this graph rather than written free-form, which makes hallucinated tool names and unsatisfiable calls structurally impossible. Each sampled chain is rendered into an executable reference solution. The initial database state is reverse-constructed so that the chain is feasible by design, distractor entities are injected, and a model jointly writes the natural-language instruction and a programmatic verifier. Every candidate must then survive a triple-check (the solution runs, the verifier accepts the post-state, and the verifier rejects the initial state), a battery of deterministic static guards covering invented literals, instruction-to-verifier literal coverage, chain-order consistency and untouched-row negative checks, a rubric-based QC review, and finally a pass@k rollout filter that discards unsolvable items and parks each task at its solvable difficulty frontier.

O*NET-grounded tool-use data synthesis. To ground tool-use data in economically meaningful work, we adapt the scenario generator of Somerstep et al. (2026), which turns O*NET occupational tasks into tool-mediated workplace requests (Figure 22). The tool ecosystem is built from the Smithery registry of Model Context Protocol (MCP) servers: we crawl every listed server, have an LLM write an analysis of its purpose and tools, and keep servers an agent can connect to without enterprise purchases or private API keys, then probe each tool with realistic inputs and retain those that return real, non-error data. Each O*NET task is matched to the servers most likely to help accomplish it by embedding the task statement and the server analyses with a Qwen3 embedding model and thresholding their cosine similarity. A scenario is generated from an occupation together with several matched (task, server) pairs: the generator LLM is given the occupation and task descriptions, web search results about them to supply realistic constraints and context, and the servers' tool schemas. It then writes a self-contained workplace request that needs a minimum number of tools across more than one server to complete, a cross-tool workflow whose links are labeled by dependency type (parameter dependency, conditional routing, cross-validation, or aggregation), and an answer key of the required tool calls with schema-conformant arguments and expected outputs. The request must contain every concrete value the agent needs (IDs, names, dates, and so on), must not name the tools or servers, and must not be answerable without calling tools. For multi-turn data, we also generate information-withheld variants that leave out one to three key parameters, so the agent has to ask clarifying questions. A simulated user answers them via role inversion: it runs in its own session, where it believes it is the assistant, receives the agent's replies as user messages, and never sees the agent's reasoning or tool calls. Because live MCP servers are rate-limited and unstable at scale, rollouts use virtual tools: an LLM that sees the server and tool documentation, the input schema and the running simulation history returns realistic tool outputs, after checking that required arguments are present, no hallucinated arguments were passed, and argument types match the schema. The resulting single- and multi-turn trajectories are used as mid-training data.

Automatic pipeline for skill-use data synthesis. Skills equip LLM agents with professional knowledge and guidance to complete long-horizon and complex tasks. To teach and improve LLMs' skill-use capability, we start from public skills, and use an automatic pipeline to construct tasks where skills provide advantages in completion. We first crawl 184k skills from GitHub and skills.sh, deduplicate and filter them. Around 12k skills that are complete, clearly written, and executable offline in a container are kept. Then, we use a builder-reviewer agentic pipeline to build tasks. The pipeline first writes initial Docker files for skills, where basic dependencies are installed. Then, starting with these initial environments, the builder is instructed to build multiple types of tasks, including multi-hop, abductive diagnosis, constraint satisfaction, and

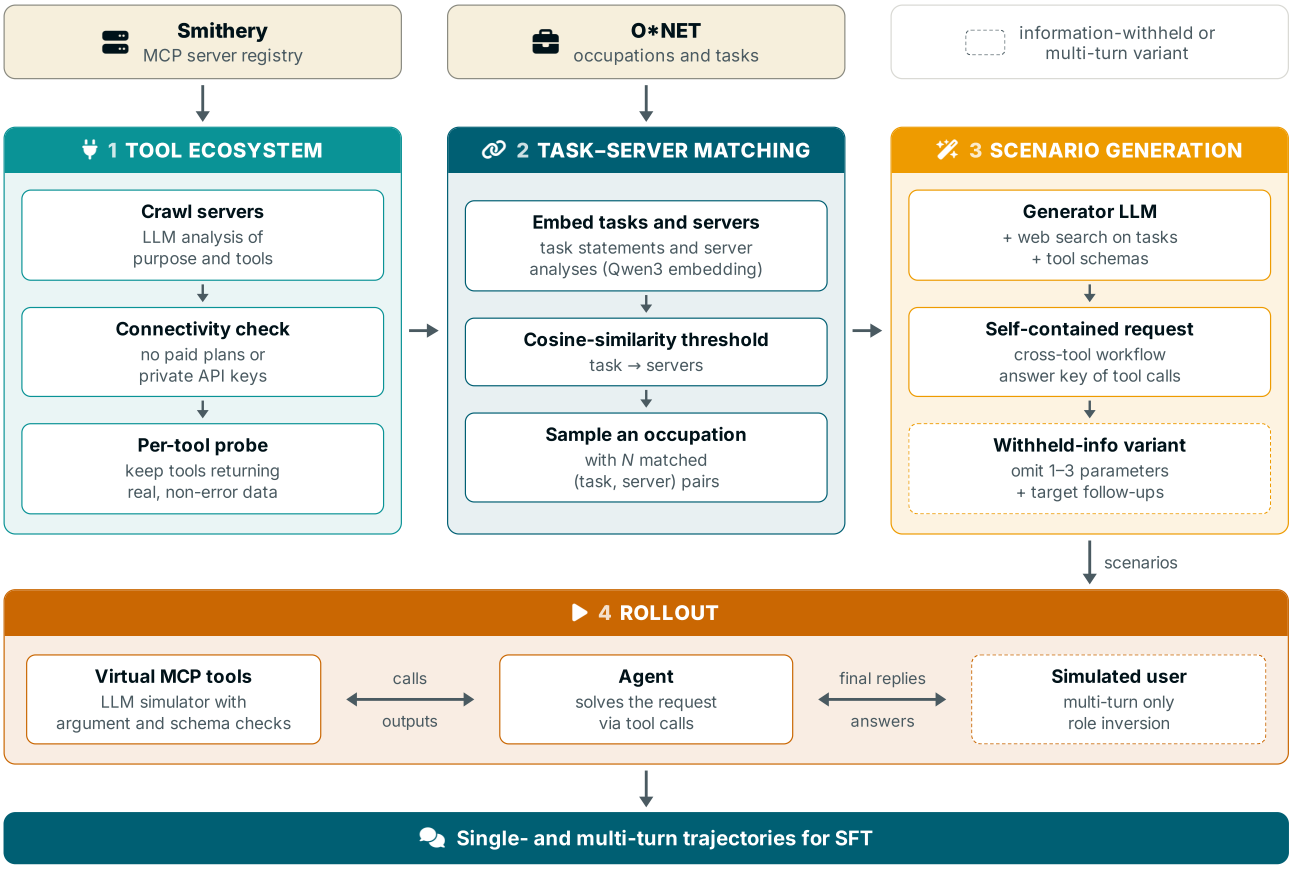


Figure 22: O*NET-grounded tool-use data synthesis, adapted from Somerstep et al. (2026). The tool ecosystem keeps only Smithery MCP servers that an agent can reach and whose tools return real data. Each O*NET task is matched to servers by embedding similarity, and an occupation with several matched (task, server) pairs is expanded by an LLM into a self-contained workplace request, a cross-tool workflow and an answer key; information-withheld variants (dashed) omit key parameters so the agent must ask for them. Rollouts run against LLM-simulated tools and, for multi-turn data, a role-inverted simulated user that sees only the agent's final replies.

partial-ordering. Each type is based on one profile, specifying the detailed steps for building such tasks. For example, multi-hop asks the builder to explore the workspace step-by-step and form the corresponding task with these sequential steps, where the input of one step is the output of the previous. Details of the other task types are in Wang et al. (2026). Each task comes with an instruction, verifier, and corresponding solution. The pipeline produced skill-related tasks at a large scale.

Skill-driven tool-use data synthesis. `Skills` provide general-purpose agents with procedural guidance for specialized tasks, including when and how to use tools (Zhang et al., 2025a; Li et al., 2026b). In this pipeline we view skills as examples of realistic LLM uses and synthesize corresponding tasks. Given a selected set of skills, we construct an environment supporting their tools and workflows, then generate contextualized tasks that require agents to apply them. The previous pipeline gives the agent a skill and builds tasks where using it helps. This pipeline treats skills as descriptions of real workflows and builds tasks that require carrying out those workflows, with or without the skill file.

We collected approximately 400k skill records from sources such as GitHub and ClawHub, normalized them, removed exact and near-duplicates (via MinHash LSH), and translated non-English skills, yielding approximately 160k skills. Each skill is annotated with domain, capability, and quality to guide sampling

toward under-represented domains and high-quality workflows, and we construct a directed skill graph based on input–output compatibility (Fan et al., 2026) so that subgraphs of complementary skills can be composed into more complex workflows. For each sampled skill (or subgraph), an LLM derives a representative workflow specifying each step's inputs, outputs, and required tools; an LLM with harness then produces executable tool simulations and materializes the data they operate on, using web search to ground textual and structured data in public sources, media generators, and document renderers to produce artifacts such as images, videos, and PDFs. Finally, given the workflow and generated data, the agent curates a contextualized task with instructions, deliverables, an oracle solution, and verification scripts, which we package with the simulated tools and initial environment state to form a complete task instance.

Skills are noisy and stateless

Skill files are diverse but noisy: sampling them at random or by embedding similarity yields largely incompatible combinations. The graph greatly helps yield coherent and composable workflows. They also carry no state or data, and LLMs asked to simulate it directly produce one-line, simplistic states; grounding generation in external tools such as web search yields richer, higher-quality states.

5.12 AI for Science

5.12.1 Task factory

Question–answer sites and papers contain many hard, checkable science problems, but a problem is not yet a training environment. Reinforcement learning needs a sandbox, a deliverable the agent must produce, a verifier whose reward can be trusted, and some knowledge of how hard the task is. We built a task factory[15] that turns candidate problems into Harbor tasks, each with an instruction, a container environment, a programmatic verifier, and a reference solution that the verifier accepts, each audited for reward hacking and calibrated against a real agent. The factory is itself an agent program in the spirit of autoresearch (Karpathy, 2026): a Markdown file that a coding agent receives as its instructions, together with a small set of skills for the individual checks, and that it follows in a loop without supervision (Figure 23). Several of us ran it in parallel campaigns over Stack Exchange and arXiv, and it produced 9,220 tasks from 15 sources (Table 23). Every task was audited with the gates that reinforcement learning needs: verifier probes, a reward-hacking audit, and review by a second model.

The loop. For each task, the agent reads an append-only ledger of previous attempts to find under-covered areas and recurring failures, mines candidate problems from the source, and picks one with a finite, checkable answer; proofs, opinions, and prose answers are rejected. It computes the answer with a script before writing any assertion, writes the task, and runs the reference solution through the verifier, abandoning the task after three failed attempts. It then calibrates the task with pass@4 against an open model and triages every trial into a failure class, such as a correct answer rejected by the verifier, an ambiguous instruction, a timeout, a loop, or a tooling error. Patterns across trials are read as evidence about the task rather than the agent: if every trial misreads the same clause, the instruction is ambiguous; if every trial gives the same "wrong" answer, the verifier may be wrong; if every passing trial brute-forced, the task is easy whatever its score. An ambiguous instruction or a wrong verifier is fixed and recalibrated once; genuine difficulty is kept, and infrastructure failures are never counted as difficulty. Tasks are published with a difficulty label from their pass rate, and overall 65% of the tasks are labeled hard, 31% medium, and 4% easy. When several agents mined the same source in parallel, they converged on the same few problem

[15] https://github.com/ifm-ai/task-factory

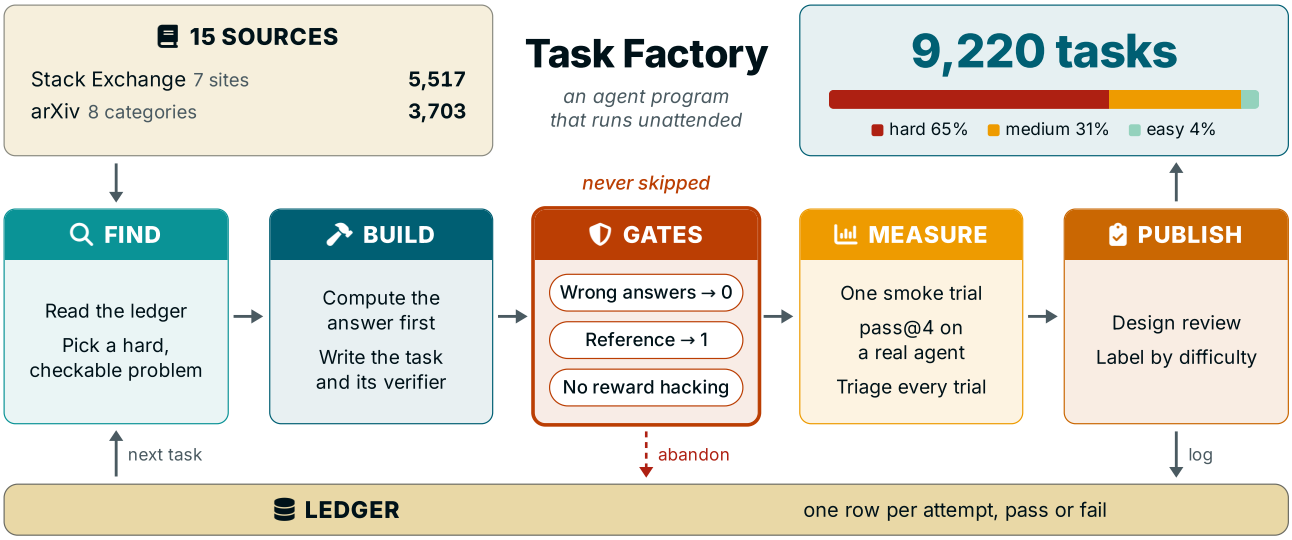


Figure 23: Task factory. A coding agent runs this loop without supervision: it finds a problem with a checkable answer, grounds it as a Harbor task, passes it through three gates that are never skipped, measures its difficulty against a real agent, and logs every attempt, pass or fail, to a ledger that it reads before building the next task.

types; in materials science, for example, most tasks drifted toward elastic tensors and Debye temperatures until we assigned each agent its own subfield.

Grounding rules. A few rules each cost us tasks before they were written down. The instruction never names its source, since an agent will look it up. The reference solution and the verifier share no derivation code, and a third, independent check substitutes the expected answer back into the original problem: when the same wrong formula appears on both sides, the oracle passes while correct agents fail (a concordant bug; see the box below). The verifier follows an output contract under which a capable agent that follows the instruction always receives reward 1: it parses JSON, compares numbers as numbers, and ignores order where order carries no meaning. When a correct answer is rejected, the instruction is fixed first and the format handling second, and the domain bar is never lowered.

Gates for reinforcement learning. A task that passes its oracle and scores 1/4 can still be useless for RL, so every task also passes three gates that are never skipped. The reference solution must pass. A probe battery submits empty, wrong, malformed, truncated, and partially correct answers, and every one must score 0, since a verifier that has only seen the reference answer has not been tested. A reward-hacking audit then asks whether the agent can score 1 without doing the work: whether the verifier or expected answer is visible or reachable in the sandbox, whether ground truth is recomputed from files the agent can edit, and whether tests or the reward file are writable. A second model with no shared context reviews each finished task for contamination, brute-forceability, ambiguity, and verifier fitness, since the agent that wrote a task is the wrong one to judge it.

Hardening the reward channel. The largest class of exploits we found was not in the tasks but in where they ran. The agent originally ran as root in the container in which the verifier ran afterwards, so it could replace the Python interpreter, write the reward file in advance, or leave a process behind that rewrote the reward after grading. We now run the agent as an unprivileged user, and the verifier clears and seals the reward directory, stops leftover agent processes, and fails closed if any of these steps did not take effect; after this change, the class disappeared from later audits. A second class of exploits came from verifiers

that graded a finite sample of cases: a lookup table with 25 entries, or a fingerprint of a seeded random generator, was enough to score 1. We added a probe that replays memorized outputs, and the rule to grade the contract stated in the instruction rather than a sample of it. Prohibitions in the instruction ("do not use tool X") that the verifier did not check were bypassed as well, for example, with a small C wrapper that evaded a lexical blocklist. Finally, some sandbox backends cannot close network access; on these backends, every contamination verdict is provisional, which is one more reason why instructions never name their source.

What makes a task hard. Across the first several hundred tasks, the separation was consistent. Agents solved tasks with a small search space, a closed-form answer, a single library call, or a textbook algorithm on small inputs. They failed tasks that combine several concepts, hinge on an edge case that the obvious approach gets wrong, require a large structured output rather than a single number, or need a representation that cannot be brute-forced. We therefore preferred problems where the domain is hard rather than the code: a domain expert with a computer should solve them in minutes.

Stack Exchange		*arXiv*	
Chemistry	1,678	Numerical analysis	1,566
Biology	1,359	Computational physics	559
Physics	1,108	Classical analysis	372
Mathematics	793	Quantum physics	354
Scientific computing	285	Probability	228
Computer science	181	Materials science	223
Statistics	113	Computer vision	201
		Artificial intelligence	200
Total	5,517	Total	3,703

Table 23: Tasks produced by the task factory, by source (9,220 in total).

A 0/4 is a claim about the task, not the agent

A task that no agent solves looks like the hard task we want, and it is also where a broken verifier hides. When the reference solution and the verifier derive the answer the same way, a shared mistake passes the oracle and rejects every correct answer; such concordant bugs were the leading cause of false 0/4 scores. A real 0/4 looks different: on one materials-science task, all four trials converted GPa^{-1} to TPa^{-1} by dividing by 1000 instead of multiplying. We trust a hard score only after an independent check substitutes the expected answer back into the original problem.

5.12.2 Creating Paper-Grounded Autoresearch Environments

Reinforcement learning with verifiable rewards (RLVR) (Kyars, 2026; Lambert et al., 2025) often trains language models using a binary signal indicating whether an answer is correct or a task has been accomplished. Many scientific and engineering objectives, however, are not simple binary tasks but are continuously evaluated: a CUDA kernel can run faster, a material can become more conductive, and a numerical method can achieve lower error or run faster. Progress on these problems depends on proposing new hypotheses, testing their effects, and building on successful experiments. Recent systems like Autoresearch (Karpathy, 2026) illustrate the potential of language models in this setting. The Darwin Gödel Machine (Zhang et al., 2026) extends this idea to agent self-improvement by modifying its own code and empirically evaluating the resulting agents.

Paper–repository mining prompt

Role. Identify paper–repository pairs that support a short, substantive CPU experiment.

Inputs. Assigned shard and window; paper and repository links; provenance flags; existing candidate notes; prior exclusions.

Instructions.
1. Reconstruct the canonical queue: retain GitHub roots with at least one provenance signal; exclude known candidates; deduplicate and order using the specified hashes. Inspect only the assigned window.
2. Read the actual paper and inspect the linked repository. Verify that the code implements the stated contribution, rather than merely citing the paper.
3. Identify a concrete executable slice of a core method or meaningful ablation. Specify its entry point, smallest useful input, command, and expected output. Prefer bundled examples or small local fixtures.
4. Assess whether the complete experiment fits the CPU budget (approximately 5–10 minutes). Record measured runtime when tested and distinguish estimates from verified execution.
5. Reject paths requiring GPUs, large checkpoints or datasets, substantial training, or notebook-only execution. An import or packaging test alone is insufficient evidence of the core method.

Return. For each retained candidate, write a structured note with paper/repository links, contribution, runnable command, inputs, dependencies, runtime evidence, risks, and a Pass/Borderline verdict. Return a scan receipt with inspected/accepted/rejected counts, created-note paths, rejection reasons, and coverage status. Report no acceptance when evidence is insufficient; preserve prior notes and assigned shard boundaries.

Figure 24: Condensed mining prompt. Queue construction is deterministic for a fixed dataset and exclusion set; scientific and runtime feasibility are assessed from the paper and executable repository.

Motivated by these approaches, we constructed *autoresearch environments*: each provides a working baseline, and the task is to improve a continuous metric over it. These environments aim to teach the scientific method of cycling through proposing hypotheses or changes, implementing them, measuring their effects, and retaining better solutions. We designed the environments to be *solvable*: there exists a solution or a change in the working environment that can produce an improvement or positive reward.

We start from Paper with Code dataset, which contains 300,161 paper–code links. We first mine it for candidates across machine learning, statistics, numerical methods, optimization, physics, and engineering, and scout each candidate to check that a task can be solved under our time and resource limits (e.g., our sandboxes have no GPUs).

In the mining phase, scouting agents inspect the actual paper and repository to verify their relationship and identify a small executable slice of a core contribution or meaningful ablation. We keep only candidates whose experiment runs within 5–10 minutes on CPU, and drop any that require GPUs, large checkpoints, large benchmark downloads, or substantial training. At the end of the mining phase, each scouting agent is asked to produce a structured note for each retained candidate with links to the paper, repository, contributions from the paper that could potentially be turned into a task, and related artifacts. We describe the mining prompt below in Figure 24.

For each selected candidate, an environment construction agent reads the full paper and freezes the repository at a commit. It identifies a measurable objective, a runnable baseline, and a research insight from the paper that can guide improvement. The baseline may be a published comparison method, a repository ablation, or a deliberately simplified implementation. Then, the agent packages the code, inputs, dependencies, instructions, evaluator, verifier, and reference solution as a Harbor task. For each task, we generate three variants with varying difficulty levels of easy, medium, and hard, with different implementation scope or hints that the agent can use to solve the task. Each variant has an `instruction.md`

specifying the editable surface in the code, a baseline metric, a target metric, scoring direction, the reward formula, research hints, and the exact evaluation command. All essential requirements are available inside the task.

The task begins with executable code and an observable baseline score so an agent can immediately enter a hypothesis–implement–evaluate–refine loop. We intentionally leave algorithmic choices open: alternative models, optimizers, search heuristics, numerical methods, kernels, or data transforms may improve the same objective. The research hint offers a useful starting direction while leaving implementation and experimentation to the solver. Each task has a 10-minute resource budget.

Let b, t, and m denote the measured baseline, target, and candidate metric. For a valid candidate, the standard reward is

$$R(m) = \begin{cases} \text{clip}_{[0,1]}\left(\dfrac{m-b}{t-b}\right), & \text{higher is better,} \\ \text{clip}_{[0,1]}\left(\dfrac{b-m}{b-t}\right), & \text{lower is better.} \end{cases} \tag{13}$$

After the tasks and their corresponding instructions are generated, the construction protocol first checks that the staged code and evaluator run under the actual container paths. Each variant is then executed with Harbor's oracle agent, which applies the reference solution and runs the verifier. High oracle reward establishes that the task is indeed solvable and runs in the constructed environment, and if any task or environment failures are detected, they are repaired before model calibration. Model calibration is conducted to test for any tasks that are too easy or too hard for a language model.

For this calibration stage, the configured evaluation model runs four separate Harbor attempts from the staged task. Each attempt may experiment within its own budget, after which the verifier executes the resulting implementation and records its measured metric and normalized reward. We use mean@4 as the difficulty score.

Finally, we review the trajectories under each of the rollouts to distinguish substantive model failures from ambiguous instructions, verifier defects, and environment problems.

5.12.3 Socratic style dialogue reasoning data generation

We created the synthetic reasoning datasets for chemistry and broader scientific domains using two complementary sources: the Open Reaction Database (ORD)[16] and MegaScience (Fan et al., 2025). We transformed questions (direct QAs from the ORD and Megascience) derived from these sources into Socratic student–teacher dialogues, capturing solution attempts over the iterations, targeted feedback, and subsequent revisions as shown in Figure 25.

ORD provides publicly available organic reaction records describing reactants, products, reaction conditions, experimental procedures, and measured outcomes. We constructed question–answer pairs spanning six tasks: product prediction, retrosynthesis, condition prediction, yield prediction, procedure description, and reaction temperature prediction. We curated the resulting corpus by deduplicating reactions with identical reaction SMILES and retaining the most complete experimental record from each group, limiting repeated answers while selecting structurally diverse examples, and filtering questions containing unresolved compound references or extraction artifacts.

MegaScience (Fan et al., 2025) complements the reaction-focused ORD data with scientific questions and reference answers spanning multiple disciplines. Its coverage includes chemistry, physics, biology, medicine, mathematics, computer science, and economics, providing a broader foundation for scientific reasoning.

[16] https://open-reaction-database.org/about

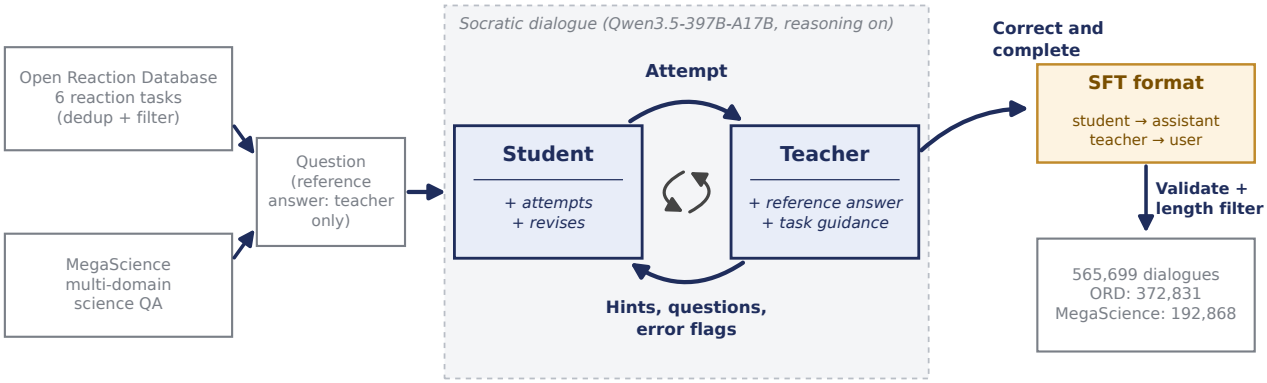


Figure 25: Socratic dialogue data generation. A student model attempts each question and a teacher model with the reference answer guides it with hints until the answer is correct. Student turns become assistant outputs for SFT.

For both sources, we generated dialogues using Qwen3.5-397B-A17B (Qwen Team, 2026) in student and teacher roles, with reasoning enabled for both. The student attempted to solve the question, while the teacher additionally received the reference answer and task-specific guidance. The teacher was instructed to guide the student through concise questions and hints, identify errors, and conclude the dialogue when it judged the response correct and complete. For supervised fine-tuning, we retained student reasoning and responses as assistant outputs and intermediate teacher feedback as user messages. After validation and length filtering, the final corpus contained 565,699 dialogues, comprising 372,831 from ORD and 192,868 from MegaScience.

5.12.4 Source grounded question synthesis

We started our source collection with a set of topics related to the benchmark relevant topics and relevant to expert-level reasoning then expanded our search with adjacent topics. We expanded coverage in two ways: large-scale web retrieval, and following the reference lists of the scholarly sources (arxiv, semantic scholar) and related cited papers. Finally, we ended up with large corpus of full-text documents. The next challenging part is that how we can create the difficult questions as STEM benchmarks from the sources, as the entire source is not relevant to construct the questions. Therefore, we built a pipeline which considers the extraction of useful in the content from the sources for generating expert reasoning level question, so we built a pipeline that first extracts the useful content and then generates questions as shown in Figures 26 and 27.

- **Evidence Extraction.** A reasoning model reads each document and extracts a few short, self-contained evidence passages, such as a theorem, a derivation, an experimental result, or a key definition. Each passage is backed by exact quotes from the source and tagged with key terms that link it to related documents. Question generation uses only these passages, not the full document.
- **Single-hop and multi-hop questions.** Questions are generated either from a single document (single-hop) or from a small group of related documents (multi-hop). A multi-hop question must use evidence from every document in its group, so answering it requires connecting ideas across sources rather than recalling a single fact. Each generation call follows one question pattern, such as applying a theorem to a specific case, calculating from a formula, or comparing alternatives. If the evidence does not support the pattern, the model returns no question instead of forcing one.
- **Standalone format.** The solver sees only the question text. The evidence, answer, and reasoning stay hidden. Along with each question, the model writes a step-by-step solution plan in which every step

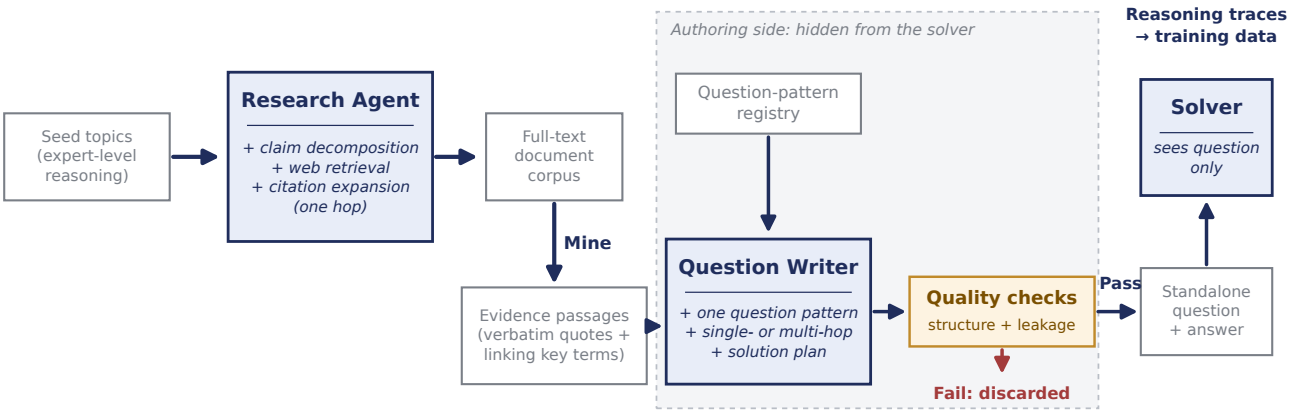


Figure 26: Source-grounded question synthesis pipeline.

builds on evidence or on an earlier step. It also explains why the answer cannot be reached by a shortcut.

- **Quality checks.** Every question goes through automatic checks that confirm it is grounded in the evidence and does not give away its answer. A question is rejected if:
  - its solution plan has too few reasoning steps;
  - it relies on too little evidence;
  - it contains steps that the final answer does not depend on;
  - it refers to material the solver cannot see;
  - its answer is copied directly from the evidence.

  These checks keep questions that need real multi-step reasoning and remove those that could be answered by lookup or shortcuts.
- **Solver rollouts.** A separate model attempts each accepted question using only the question text. We save its full reasoning trace and final answer as training data.

Feedback-driven question synthesis. Apart from the above pipeline, we built a second pipeline on the same sources. Instead of building difficulty into the question format, it measures difficulty while questions are being generated and uses that signal to make them harder:

- **Passage grouping.** A reasoning model extracts passages from each source. The passages are then grouped into small sets drawn from a few related documents. Each group gets randomly sampled settings, such as reasoning type, answer format, and question style, to keep the questions varied.
- **Solver feedback loop.** The model first writes a batch of questions from its group. A smaller model then tries to answer each question without any supporting context. The first model is told which questions were solved and asked for a new, harder batch. In early tests, simply asking for *hard* questions had little effect. What helped was naming the questions that had been solved, and asking whether an expert could answer them from general knowledge alone.

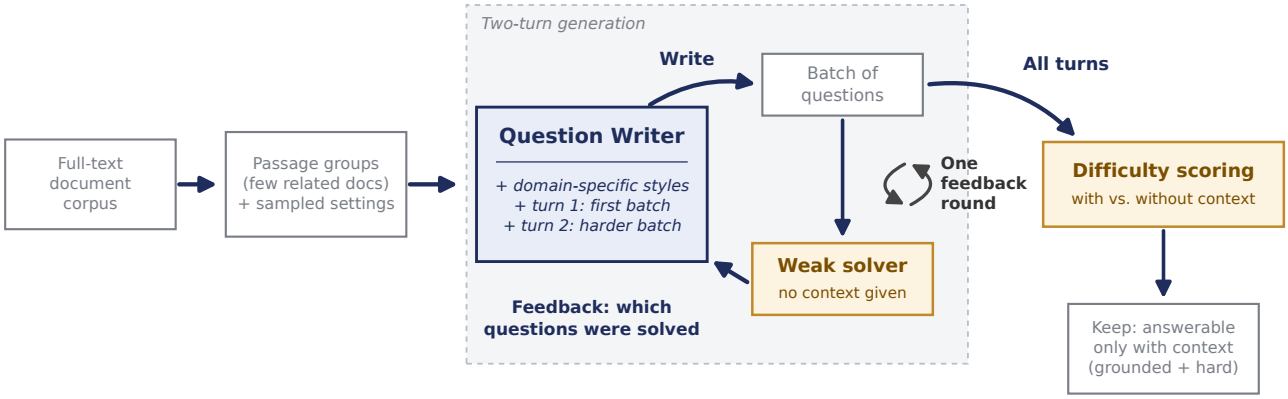


Figure 27: Feedback-driven question synthesis pipeline.

- **Domain-specific question styles.** Each topic is routed to a domain family (formal, quantitative, empirical, or interpretive), and the allowed question styles change with the family. This keeps, for example, humanities questions from being forced into artificial calculations. The pipeline also removes low-quality and homework-help sources, and spends more of its budget on fields that existing datasets cover poorly.
- **Difficulty scoring.** Each question is answered twice by the smaller model, once with its supporting context and once without. The answers are graded by automatic answer matching rather than by an LLM judge. The most valuable questions are those that can be answered only with the context: they are grounded in the source but cannot be solved from general knowledge.

The two pipelines approach difficulty from opposite sides. This pipeline measures difficulty and uses feedback to raise it. The first builds difficulty into the question by requiring a multi-step solution over hidden evidence. Our move to standalone questions came from this pipeline's finding that questions paired with a supporting context were often too easy.

5.12.5 Research lab: a simulated research community

Research is done by communities of researchers who cooperate and compete at the same time. They split up work, share code, and co-author papers, but they also race for priority, credit, attention, and resources. Agent trajectories collected on isolated tasks show none of this behavior. To produce data that does, we built a *research lab*: a persistent, real-time simulation where a group of researcher agents works toward new results on a shared pool of open problems (Figure 28). The lab draws its problems from the task pipelines of this section. Its open problems are Harbor tasks, and its method-improvement tasks are the autoresearch environments of §5.12.2.

The world is built on Concordia (Vezhnevets et al., 2023). A game master holds the world state, applies the rules, and checks the win conditions. Each world has ten researchers. Each one is an OpenCode agent (Anomaly, 2026) with a persona (name, temperament, and writing register), a disciplinary profile such as computational physicist or pure mathematician, and a matching bundle of skills and scientific libraries. A researcher has a private workspace and acts on the shared world only through a command-line tool. Every call resolves immediately against the live world, so the world keeps changing while the agent thinks. Long sessions are split into work periods with context compaction, and a recall command searches what the agent observed in earlier periods. Kimi K3 (Kimi Team, 2026b) plays every role: the researchers,

the game master, the conference referee, and the simulated public.

- **Open problems and the conference.** A public board lists about 1.5k open problems from numerical analysis, classical analysis, materials science, statistics, and scientific computing, each with a hardness rating from 1 to 5. Pulling a problem delivers its full statement privately but does not reserve it. A submission is a folder with a paper, the answer, and the code that produced it. An automated conference first checks the answer, then a referee judges whether the paper derives it end to end. Thin papers are rejected even when the answer is correct. The first accepted paper closes the problem and later submissions are scooped, so researchers must trade speed against rigor.
- **Autoresearch board.** About 870 method-improvement tasks come as git repositories with a metric, a baseline, and a target. Submissions are committed workspaces with a report and are evaluated asynchronously on the host. The current best result on each task earns a leaderboard point, and its authors may write it up for the conference's Autoresearch track.
- **Credit and the economy.** A solo paper of value w pays w, while a paper with two or more authors shares a larger pot of $2.2w$, so collaboration pays. Researchers start with 100 coins and can pay one another for anything they agree on, such as solving, writing, a co-authorship slot, or promotion. A market lists offers and requests and holds payment in escrow when the deliverable is a file.
- **Public square and coordination.** An OASIS social network (Yang et al., 2024) is the only public channel. It is populated by a large simulated public of journalists, students, aggregators, and skeptics who react to posts that interest them, and accepted papers and leaderboard verdicts are announced there. In private, researchers coordinate through direct messages, group chats, file sharing, and shared git repositories.
- **Win conditions.** The game ends when one researcher reaches a target in public engagement, paper credit, autoresearch credit, or coins. These four goals reward different strategies, such as grinding papers, topping leaderboards, building an audience, or trading.
- **Collaborative variant.** A second configuration removes competition. The team shares one goal, bringing every task on the autoresearch board to its target, and it divides the work by expertise, shares partial results early, and pools its submission budget.

Trajectory export. At the end of a run we export each researcher's full agent trajectory in two views: with compaction turns removed, and split into one segment per work period. The compaction turns themselves form a third set. We also keep a selected subset of the simulated public's decisions. Every record passes the same filter as our SFT data, which rejects records where the model identifies itself as another model, anonymizes run-specific paths and account names, and computes token statistics. From a 2026-08-20 snapshot of 380 runs (238 played to completion), we obtained 3.0k researcher trajectories (338M tokens), 0.4k compaction records (52M tokens), and 31.8k public-account decisions (341M tokens). We use the resulting 35k records (0.73B tokens) in the mid-training stage.

5.12.6 Data-science trajectories from Kaggle competitions

Data analysis is central to scientific work, from interpreting experimental measurements to evaluating predictive models. Our goal is to improve our models' performance on data-science tasks by training on trajectories that cover data inspection, preprocessing, model development, and evaluation. We built a pipeline (Figure 29) that converts Kaggle competition datasets (Kaggle, n.d.) into MLE-bench-style tasks (Chan et al., 2024), aligning the public data layout, submission interface, and tool-using agent workflow with the setting used to evaluate the trained models. We organize these tasks into five workflow categories, summarized in Table 24.

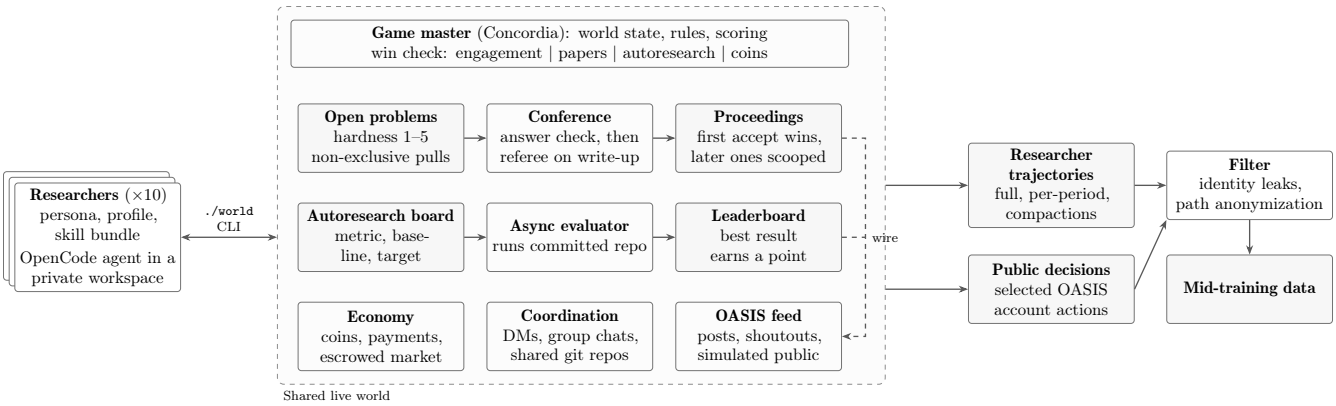


Figure 28: The research lab. Ten researcher agents with distinct personas and skills act on one shared, real-time world through a command-line tool, and a Concordia game master enforces the rules and checks four win conditions. Competition comes from non-exclusive pulls on open problems, where the first accepted paper scoops the rest, and from the leaderboards and the race for public engagement. Collaboration comes from the larger credit pot for co-authored papers, payments and escrowed deals, shared git repositories, and the collaborative variant in which the whole team pursues one goal. Each researcher's trajectory and a selection of the simulated public's decisions are filtered and used as mid-training data. Shaded boxes hold task or result state; dashed arrows are public announcements.

Data crawler. We built a browser-based crawler to discover competitions open to all participants across Kaggle's research, featured, playground, recruitment, community, and getting-started categories. For each competition, we collected problem and dataset descriptions, evaluation criteria, submission requirements, and file metadata, together with leaderboard scores and metric direction when available. Following MLE-bench, we classified performance into gold, silver, and bronze tiers using Kaggle's team-count-dependent rank formulas as shown in Equation 14. We applied this rule uniformly across competitions with usable leaderboard data, regardless of whether they officially awarded medals.

Let N denote the reported number of participating teams, falling back to the number of eligible collected leaderboard entries when this count is unavailable. The cumulative cutoff positions for gold (r_{G}), silver (r_{S}), and bronze (r_{B}) are:

N	r_{G}	r_{S}	r_{B}
$1 \le N < 100$	$\lfloor N/10 \rfloor$	$\lfloor N/5 \rfloor$	$\lfloor 2N/5 \rfloor$
$100 \le N < 250$	10	$\lfloor N/5 \rfloor$	$\lfloor 2N/5 \rfloor$
$250 \le N < 1000$	$10 + \lfloor N/500 \rfloor$	50	100
$N \ge 1000$	$10 + \lfloor N/500 \rfloor$	$\lfloor N/20 \rfloor$	$\lfloor N/10 \rfloor$

(14)

where $\lfloor \cdot \rfloor$ denotes rounding down. Let $s_1, s_2, \ldots$ be the scores of eligible collected leaderboard entries ordered from best to worst. For each tier $m \in \{\mathrm{G}, \mathrm{S}, \mathrm{B}\}$ with a positive cutoff and an available boundary score, the threshold is $\tau_m = s_{r_m}$. A valid submission with score s meets this threshold when $s \ge \tau_m$ for a higher-is-better metric or $s \le \tau_m$ for a lower-is-better metric; we assign the highest qualifying tier. Zero cutoff positions, missing boundary scores, and cutoffs beyond the collected entries remain unavailable. These thresholds provide approximate performance targets for local evaluation. We deduplicated competition identifiers and saved collection progress to support resumable crawling. We then paired downloaded and extracted datasets with their normalized metadata to provide the inputs for task generation.

Task construction. An adapter agent inspected each dataset and its competition specification, then wrote data-preparation code and public task instructions, and prepared the evaluation contract and grading artifacts. For prediction tasks, we constructed deterministic holdouts from available labeled data, respecting

Category	Task focus
Data cleaning and preparation	Preparing and cleaning data, resolving inconsistencies, and checking tables or joins.
Data analysis and evaluation	Computing and interpreting analyses, and assessing results against task-specific references.
Model training and prediction	Fitting models and producing predictions, including structured outputs such as masks, intervals, and forecasts.
Model optimization	Improving predictive performance or meeting verifiable constraints on model artifacts.
Programming and debugging	Implementing standalone programs or repairing repository code, with correctness checked through execution tests.

Table 24: Five workflow categories used to organize the synthesized tasks. Categories can overlap; each task is assigned a primary category.

grouped entities and temporal ordering where required. The solver received training labels and held-out inputs, while the corresponding answers remained private. This provided local evaluation when Kaggle's official test labels were unavailable (Chan et al., 2024). We validated the prepared files, split consistency, submission interface, and evaluator behavior before launching the solver, including checks that valid submissions could be scored and malformed submissions were rejected.

Solution generation. We used Kimi K3 (Kimi Team, 2026b) and DeepSeek-V4-Flash (DeepSeek-AI, 2026) through OpenCode (Anomaly, 2026), assigning task preparation and solving to separate agent roles with different filesystem permissions. The solver agent inspected public data, wrote and executed analysis code, trained models, and produced a submission and solution report. A controller ran the task-specific evaluator and retained the best valid attempt. Within a bounded search budget, subsequent attempts received the previous score, metric direction, and summaries of earlier approaches to guide alternative strategies. Scores came from our locally prepared tasks; comparisons with Kaggle medal thresholds were approximate and did not constitute official competition results.

Repair and monitoring. Failures in the adapter and solver stages were handled by two dedicated repair agents. An adapter-repair agent received task-validation errors and corrected the prepared task, while a submission-repair agent received sanitized failure feedback, without private labels, scores, or grader code, and regenerated the submission. Each repair was followed by validation and, for submissions, grading. A separate solver-monitor agent inspected execution activity, artifact changes, and recent tool events to identify stalled sessions and inform controller-managed recovery.

Trajectory collection. We trained only on solver-agent trajectories, which capture the reasoning involved in solving each task alongside tool calls and execution outputs. We preserved links to the dataset, model, and solver attempt. We converted these records into training conversations and applied the shared data-processing checks for conversation structure, tool-call consistency, reasoning content, repetition, template leakage, and token length. Task-level success labels were retained separately from conversation-quality checks.

Model evaluation. After training, we use MLE-bench to evaluate model checkpoints on its 75 machine-learning engineering competitions. We retain only the solver and submission-repair roles from our synthesis pipeline, since MLE-bench already provides task descriptions, prepared data splits, submission requirements, and task-specific graders. Our harness exposes each competition's `prepared/public/` files in the solver's `public/` directory, including the task description, training and test data, and submission examples where provided. Private reference answers in `prepared/private/` remain inaccessible to the solver. Following

the OpenCode workflow used for trajectory generation, the solver agent inspects the public files, writes and executes solution code, and produces a submission. If the submission fails validation, a submission-repair agent receives sanitized error feedback and regenerates it. Our harness scores the final valid submission once with the official MLE-bench grader, without returning evaluation scores to either agent. We compare task scores and medal outcomes across checkpoints to track data-science performance.

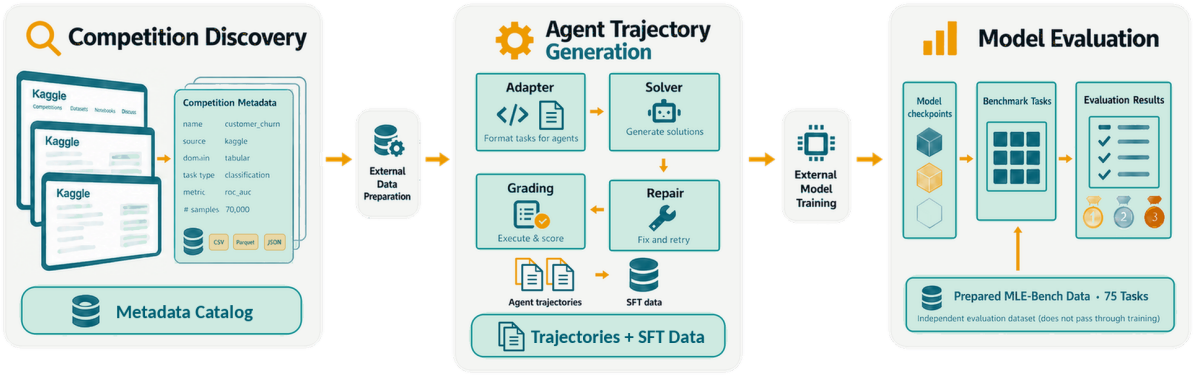


Figure 29: Kaggle trajectory synthesis pipeline. From Kaggle datasets to MLE-bench-style tasks and data-science agent trajectories.

5.12.7 Learning to experiment with simulators

We built a pipeline that turns technical documents into interactive simulators for post-training. The aim is to help agents learn through experimentation, as explored in ScienceWorld (Wang et al., 2022) and DiscoveryWorld (Jansen et al., 2024). One agent designs the actions, observations, and scoring rules; another writes the simulator code. Tasks are designed so that earlier observations can guide later decisions, such as testing a chemical process before changing its settings. We check that each simulator behaves consistently and keeps hidden values private. We then keep tasks where a strategy using only the information available to the solver performs better than both doing nothing and acting randomly. Accepted tasks support training-data generation and automatic scoring. The pipeline is presented in Figure 30.

5.12.8 From research proofs to verifiable ML-theory tasks

A growing body of work mines the research literature for mathematical benchmarks and training data (Zhang et al., 2025b; Dekoninck et al., 2026b). Exact-answer extraction covers results with a single numerical or closed-form answer (Zhang et al., 2025b), and proof-oriented approaches use LLM judges or formal proof infrastructure (Peyronnet et al., 2026; Yang et al., 2026b). ML-theory results have two characteristic features: they are mostly bounds and rates whose constants the paper leaves unspecified, and they are proved through long chains of lemmas. Our ML-theory task generation pipeline is built around both. It turns a paper's proof into a self-contained problem set in which every intermediate result is graded, grades bounds and rates with executable checks, and builds each task through specialized roles that review and stress-test it before release.

- **The proof tree becomes the problem set.** For each paper, the pipeline extracts the proof dependency graph and selects a subtree around a central result. Boundary results are given as assumptions, and each internal result becomes a subproblem that later subproblems may cite. A task passes only when every subproblem passes, so the reward measures how well a model composes intermediate results, tracks the assumptions they use, and gets their parameter dependence right.
- **Executable checks for bounds and rates.** Each subproblem has a deterministic checker that relaxes exactly what the source paper leaves free. Closed forms are compared symbolically at random parameter values. Bounds stated with $O(\cdot)$ constants are compared up to one positive constant, with

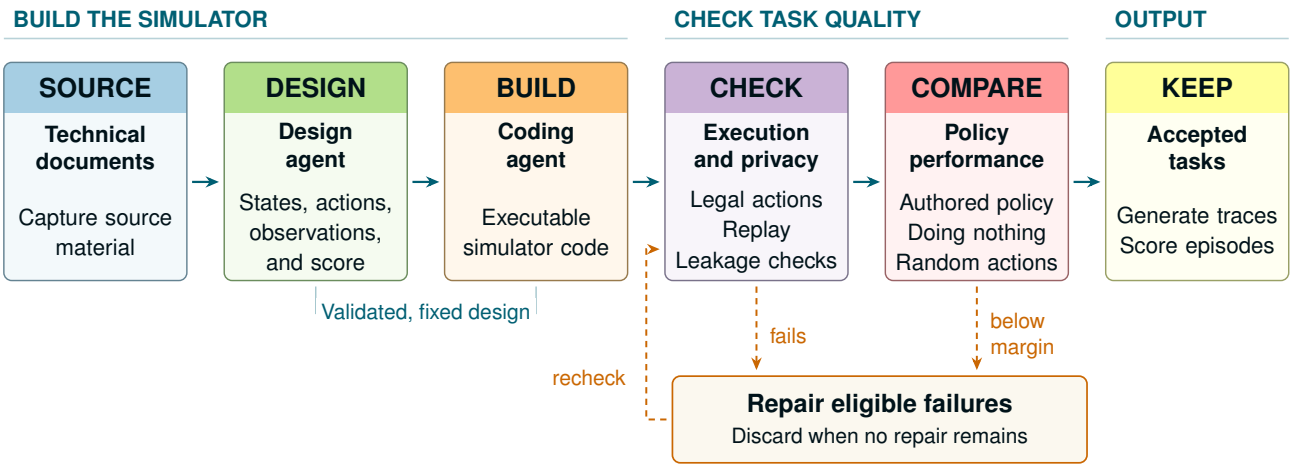


Figure 30: From technical documents to interactive simulator tasks. A design agent defines the task, and an implementation agent writes the simulator. Candidates are checked for correct execution, repeatability, and direct leaks of hidden values. A policy using only information available to the solver must then beat both inaction and random actions by a preset margin in average score across the same test seeds. Eligible failures receive bounded repair and are checked again. Accepted tasks support trajectory generation and automatic scoring.

the dependence on every parameter checked exactly. Rates are checked through their values across scales, and constant certificates are checked as systems of inequalities. Reference answers come from executing a reference solution, which must pass every checker.

- **Specialized roles that guard task quality.** Each role in the pipeline is responsible for one aspect of task quality, and every construction step pairs a proposer with a reviewer. The extractor's proof graph is checked against the paper, the task writer's subproblems are checked by a critic for well-posedness, and the reformulator's rewrite is checked by its own reviewer. The reformulator targets the crux of the central result: it rewrites the root question so that the graded quantity follows from the key step of the argument, for example an instance-specific constant or a jointly optimized parameter, which a model reaches by carrying out that step. Each finished task then goes through three independent quality checks. A cold solver tests self-containment: it attempts the task from its text alone and reports any missing assumption or ambiguous reading, which is repaired by adding the missing statement or pinning the intended reading. A shortcut probe tests that the reward requires the derivation: it tries to pass the real checker with an answer produced by guessing or by reusing a given, and a task whose checker accepts such an answer is tightened or rejected. A consistency check tests fidelity to the source by comparing the reference answers with the paper's theorem.

Source papers come from ten arXiv primary categories (`math.OC`, `stat.ML`, `cs.LG`, `cs.IT`, `math.PR`, `math.NA`, `cs.DS`, `eess.SY`, `math.ST`, `cs.DM`), covering high-dimensional statistics, robust learning, generalization, online learning and bandits, optimization, concentration, differential privacy, control, numerical analysis, information theory, and deep learning theory. We measure difficulty with an open-weight solver agent, MiniMax-M3, run four times per task; solve rates span the full range from 0/4 to 4/4. All tasks run in Harbor environments with deterministic, code-only scoring. The agent roles of the pipeline are run with Kimi K3, and we record its trajectories: pipeline-building trajectories, which show how tasks are constructed from source papers, and solver trajectories on the resulting tasks. Figure 31 gives an overview.

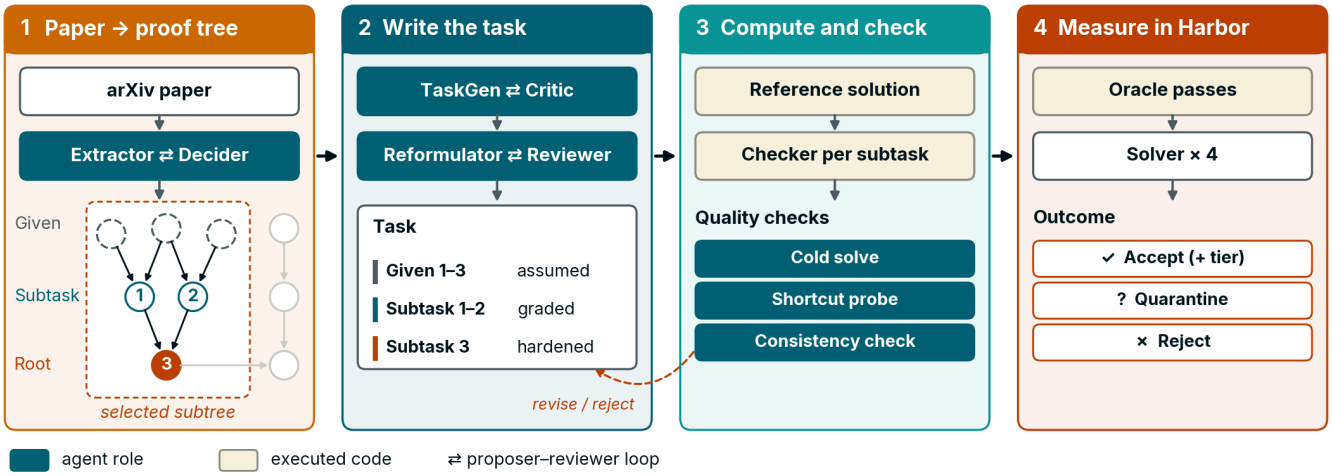


Figure 31: ML-Theory Task Generation Pipeline. (1) A subtree around a central result is selected from the paper's proof graph; boundary results become assumptions and internal results become subtasks. (2) Proposer–reviewer pairs write the subtasks and harden the central question. (3) An executed reference solution and a checker per subtask decide the answers; three quality checks (cold solve, shortcut probe, consistency check) send the task back for revision or reject it. (4) In Harbor, a solver is rewarded only if every checker passes. Violet: agent roles; teal: executed code.

5.13 Long Context Document Understanding

To strengthen the model's ability to read and reason over documents, in particular long ones, we synthesize two data subsets with two pipelines. The first, LCQA, generates question–answer pairs over single long documents: long documents are first annotated and filtered down to high-quality passages, questions are then written from those passages and trained with the full document in context, covering retrieval, comparison, causal and multi-hop task types. The second, LCR, targets multi-document reasoning, where the answer only exists once evidence from several long documents is combined. Both subsets carry the reasoning trace of the answering model and are emitted as two-turn conversations.

Long-context question answering (LCQA). LCQA is built around a single constraint we call answer invariance: a question is written about a short, high-quality passage, but the training example presents the entire document that passage came from. Because the answer is fully determined by the passage, it remains correct when the context expands, but reaching it requires searching tens of thousands of tokens rather than reading a short one. This avoids the two common failure modes of synthetic long-context data, namely padding a short QA pair with filler (a needle that was never hidden) and prompting over a whole document (which tends to produce questions about its opening pages). The pipeline[17] has two stages: long-document annotation, and question and answer synthesis from the annotated passages.

Stage 1: long-document annotation. The input is every document between 8k and 512k tokens in our pre-training corpus, spanning web, scientific, code and math, book and encyclopedic sources. A document of this length is rarely uniformly good or uniformly poor, so scoring it as one unit discards exactly the information we need, which is where its good parts are. The standard quality scorers also transfer poorly to documents of this length: the transformer-based FineWeb-Edu (Lozhkov et al., 2024) regressor truncates its input, so a whole-document score for a 128k-token document is in practice a score for its first few hundred tokens, and the fastText classifiers are trained on web pages that are far shorter than anything in this corpus, so their bag-of-n-gram statistics are diluted at these lengths. Both arguments point the same

[17] https://github.com/ifm-ai/LCQA

Length range	Documents	Docs kept	Tokens	Tokens kept
8k–32k	57.97M	8.8%	784.1B	13.7%
32k–128k	3.59M	34.9%	205.7B	39.4%
128k–512k	0.38M	40.4%	60.1B	38.0%
Total	**61.94M**	**10.5%**	**1,049.9B**	**20.1%**

Table 25: LCQA long-document annotation yield by document length.

way, and we therefore make the section, rather than the document, the unit of judgment:

1. **Structure-aware splitting.** Each document is cut into sections of at most 512 tokens by a splitter chosen per source: a semantic splitter for prose, a Markdown splitter for papers and books, and a tree-sitter splitter for code and math[18]. Splitting on structure rather than at a fixed stride keeps headings with their bodies and code blocks intact, so that a section is a self-contained passage able to host a question on its own, and it keeps every section within the input budget of the scorers that judge it.

2. **Section scoring.** Every section receives three quality judgments, from DCLM/ELI5 (Li et al., 2024), PreSelect (Shum et al., 2025) and the FineWeb-Edu educational-value regressor (Lozhkov et al., 2024), plus a multi-dimensional toxicity score[19].

3. **Rank bucketing.** The three quality scores live on incomparable scales (two probabilities and a ∼0–5 regression), so each is mapped onto a shared 0–19 scale using cut points from the corpus score distribution.

4. **Document filtering.** Section scores are aggregated to the document and the document is dropped unless its means clear every threshold (FineWeb-Edu bucket ≥ 17, ELI5 and PreSelect ≥ 10, toxicity ≤ 0.2, length $\geq$ 16k tokens).

5. **Passage sampling.** Within each surviving document, only sections that independently clear a stricter bar are eligible; we sample four and pair each with the full document text.

Table 25 reports the yield. From 61.9M documents and 1.05T tokens we retain 6.5M documents (10.5% of documents, 20.1% of tokens) and emit roughly 26M passage–document pairs. Notably, the keep rate rises steeply with document length, from 8.8% in the 8k–32k range to 40.4% in the 128k–512k range: text that sustains itself over 128k tokens is typically a book, review article or textbook, so length acts as a quality signal rather than a liability.

Stage 2: question and answer synthesis. For each surviving document, Qwen3.5-397B-A17B (Qwen Team, 2026) sees only its sampled passages, in document order, and writes one question about them. Answer invariance decides which questions may be asked: the answer must be fully determined by the passages, so the prompt rules out questions whose answer can change when the context grows, such as counts, lists of "all" items, or questions about the document's structure. Within this constraint, three knobs are drawn per document:

- **Task type** (17): retrieving a stated fact or an attribute of an entity; relationships and comparisons; multi-hop questions that combine passages or resolve an indirect reference; causal, consequence,

[18] We use the `semantic-text-splitter` library, with token counts from `tiktoken`. Sources with reliable markup (arXiv, PubMed Central, PG19, MegaMath) are handed to the splitter whole; for the rest, we first segment on sentence boundaries and call the splitter only on pieces that are still over budget.

[19] From the `detoxify` library (`multilingual` model), https://github.com/unitaryai/detoxify.

purpose and conditional reasoning; methods and processes; the state of an entity or a value at a specific point in the text; definitions, examples, inference from stated facts, and stated opinions.

- **Question format** (9): open-ended, short answer, yes/no, multiple choice, true/false, fill-in-the-blank, comparative, temporal, and extractive (asking for the exact phrase the text uses). Some task types allow only the formats that suit them.
- **Difficulty** (3): easy (answer stated in similar wording), medium (connecting two or three pieces of information, some paraphrasing) or hard (synthesis, different wording, multi-step reasoning), drawn with weights 0.1/0.4/0.5.

The question is returned as JSON and validated against its format. The same model then answers it from the passages alone, and only the training conversation replaces the passages with the full document (§5.16). The question is placed before or after the document in one of 48 layouts that vary its prefix and the framing of the document. Of 6.5M documents, 99.6% yield a training conversation: 6.48M conversations and 211B tokens, of which 5.05M are 8k–32k tokens long, 1.31M 32k–128k and 0.12M 128k–512k.

Long-context reasoning (LCR). The LCR subset targets questions that can only be answered by combining evidence scattered across several long documents. The documents are real web documents from domains such as company filings and earnings materials, government consultation responses, industry reports, legal documents and academic papers, and a set is built so that its documents are tied together by a shared axis: the same company across successive reporting periods, peer companies in the same period, one event covered by different document kinds, many respondents to one consultation, or several reports on one theme. Typical questions rank peer companies by a figure each reports separately, compute a change across a company's quarterly filings, count how many consultation respondents took a given position, or list the entities that satisfy a condition across a collection of reports. Its design is inspired by the AA-LCR benchmark (Artificial Analysis Team, 2025). Each sample is a set of 3–24 documents totaling roughly 70k–384k tokens, a question whose answer must be derived across documents rather than looked up in one, a short verifiable answer, and the full reasoning trace of the answering model.

Pipeline. The pipeline runs in seven stages: documents are retrieved and packed into coherent sets, evidence passages are selected programmatically, a question is generated over them, it is answered from the full set, its difficulty is labeled, the sample is verified for multi-document necessity and correctness, and the result is post-processed into a training conversation.

1. **Retrieval and assembly.** For each target set, we enumerate a retrieval plan from an entity registry, a period grid and the document kinds appropriate to the domain, holding the shared axis fixed and varying the rest, and issue the resulting queries to the Keenable[20] search and fetch API, restricted to authoritative sites such as regulatory filing databases, consultation portals and preprint servers. Fetched pages are stripped of boilerplate, quality-filtered, checked for relevance to the plan (the target entity must be the subject of the document, not a passing mention), deduplicated exactly and by MinHash (Broder, 1997; Lee et al., 2022), and packed to a sampled token target.
2. **Programmatic grounding.** Before the generator sees a set, code selects one paragraph from each of three or more distinct documents, scoring for reasoning affordances (numeric density, period markers, definitional language) rather than salience, and preferring passages that share an entity or period. Cross-document necessity and non-salient evidence therefore hold by construction, at zero model cost.
3. **Question generation.** A large reasoning model sees the grounding passages and the whole set and

[20] https://keenable.ai/

writes a question that needs every passage, prefers derivation over lookup, and has a short atomic answer. Mechanical guards reject over-long questions and the ones that restate the passages' figures, with one retry that feeds the violation back.

4. **Answering.** The answerer sees only documents plus question, in one of 2,880 prompt formats composed from five independent axes (document wrapper, envelope, question position, question delimiter, instruction phrasing), so the model learns the skill rather than a scaffold. Half the samples in the 105–384k band use our default chat template. The reasoning channel is kept verbatim and never truncated.

5. **Difficulty label.** A deliberately weak long-context model answers the question blind, and whether it succeeds is recorded for every sample. This follows the logic of adversarial filtering (Zellers et al., 2018; Bras et al., 2020), where a weaker model's success marks an example as easy, except that we keep the label rather than discard the sample.

6. **Verification.** A necessity ablation re-answers the question from each required document alone under the identical template; any match rejects the sample as single-document answerable. This is the document-level form of the "disconnected reasoning" check used for multi-hop QA datasets (Min et al., 2019; Trivedi et al., 2020), where questions meant to need two passages turned out to be answerable from one. Survivors are then judged for correctness, ambiguity and format by an LLM judge (Zheng et al., 2023) from a different model family than the answerer (enforced in code), or, on the deadline-bound runs, by agreement between the answerer and the question generator. Every row records which chain verified it.

7. **Post-processing.** The prompt is rebuilt from the document store and hash-verified against what the answerer actually saw, then emitted as a two-turn conversation with separate `think` and `content` fields, stamped with run, verification chain and sample id.

Three production runs over two document stores (one for 70–128k prompts, one for 105–384k) produced 47,653 verified conversations totaling 6.2B tokens after the shared quality gate, with prompts from 63k to 391k tokens, answers averaging about 10 tokens and reasoning traces averaging about 3k. A further 145k conversations that failed verification are kept as a separately labeled tier. Accepted samples cover all seven domains and all five assembly patterns, with company filings the largest share.

5.14 Abstention

The abstention data consists of factoid questions generated from public sources across a diverse set of domains and topics, with the goal of training models to learn when to abstain. Questions that are unambiguous and can be reliably answered from public information retain their reference answers, while questions that cannot be reliably answered are assigned [IDK]. The data is constructed using a three-stage pipeline. Passage Collection gathers text from openly licensed public sources across domains. Grounded QA Generation converts these passages into standalone factoid question–answer pairs whose answers are verified against the source passages. The Abstention Review then identifies questions that depend on source/temporal-specific information and relabels their targets as [IDK].

Passage Collection. The first stage collects passages from public online sources across different domains, with a predefined target number of passages for each domain. Each domain is associated with a set of sources and retrieval parameters, such as search terms, API endpoints, or regulation titles. Records are retrieved through public APIs or publicly accessible web pages and converted into self-contained text passages using source-specific preprocessing. Structured records are rendered into readable text, web pages are cleaned and segmented at sentence boundaries, and regulation sections are preserved as complete

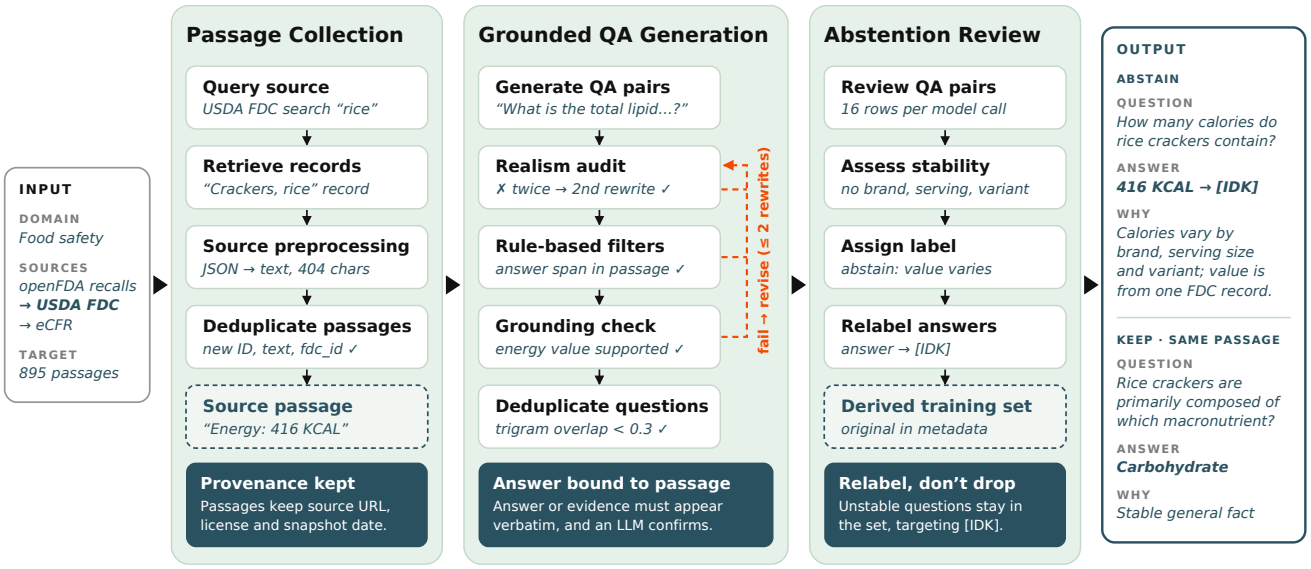


Figure 32: Factoid & Source-Specific Question Generation Pipeline. Well-posed factoid questions drive factual recall capability, while ambiguous source-specific questions drive abstention capability.

passages. The resulting passages are normalized and filtered to remove short or duplicate content, including duplicates identified through source-specific record identifiers. Each retained passage is assigned a stable identifier and preserves its source URL, license information, and snapshot date, allowing this provenance to be carried through to the QA pairs generated from it.

Grounded QA Generation. The second stage converts each collected passage into factoid question–answer pairs that are realistic and grounded in the source passage. An LLM generates six candidate pairs from each passage, each with a short answer and a supporting quote. LLM-based filters assess the formulation of each question, rejecting those that depend on implicit context, reveal the answer, are malformed, or refer to the source through phrases such as "the passage", "the source", or "according to". Candidates that fail are rewritten using the filter feedback and evaluated again, with up to two rewrite attempts. Rule-based filters further enforce basic requirements on the question and answer and verify that the answer or its supporting evidence appears in the passage. A grounding check then confirms against the passage that the answer is correct and supported by the cited evidence. Finally, each question is compared with previously accepted questions to remove exact or highly similar duplicates.

Abstention Review. The final stage determines whether each accepted question should retain its reference answer or be assigned an abstention target. Although the previous stage ensures that each question is well posed and grounded in its source passage, some answers remain specific to a particular record, product, version, jurisdiction, or point in time. An LLM therefore reviews each question together with its answer, supporting evidence, and source metadata, and decides whether to retain the reference answer or assign [IDK]. Questions are marked for abstention when their answers depend on source-specific information, such as product calories or product recall, or on time-sensitive information such as current statistics, policies, statuses, or regulatory records without a clear date or version. In contrast, general facts and clearly dated historical facts retain their reference answers. Abstained questions remain in the dataset, but their labels are replaced with [IDK], while the original answers and review results are preserved as metadata.

Trajectory Generation & Selection. Using these synthesized questions, we generate four trajectories per question using the abstention prompt from AA-Omniscience (Jackson et al., 2025), which allows the model to either provide an answer or abstain when additional information is needed. We retain trajectories that

produce either a fully correct answer or an abstention, while discarding incorrect and partially correct responses. Sampling multiple trajectories captures differences in the model's response to the same question. In particular, a model may answer a question correctly in some trajectories while abstaining in others when it determines that additional context is required. As such, retaining both types of valid trajectories provides training examples for learning when to answer confidently and when to abstain.

5.15 Browser Use

The browser-use pipeline synthesizes realistic tasks on live, public websites, which a browser agent then executes to produce multi-turn trajectories. It faces two main challenges. The first is feasibility: a website is an environment we can only observe, so a task is useful only if the site supports it at rollout time. The second is diversity: left unconstrained, a generator drifts toward short look-up questions and underrepresents the transactional, multi-constraint, and underspecified requests that real users make. The pipeline has three stages: URL collection, page acquisition, and reverse task synthesis, which is controlled by three axes (intent, difficulty, and ambiguity).

Website collection. We begin with 84 unique seed websites from Mind2Web (Deng et al., 2023) and WebVoyager (He et al., 2024), utilizing only the domain information rather than benchmark instructions. This set is expanded to 544 candidate URLs through human-assisted LLM augmentation. To ensure accessibility and relevance, we visit each candidate using headless Chromium via Playwright. We discard URLs that time out or return HTTP status codes of 400 or higher. For successful requests, we retain the final URL after any redirects, as regional redirects can alter the content visible to the agent. Finally, we download the rendered homepage, which serves as the sole input for the task generator.

Reverse task synthesis. An LLM reads the homepage snapshot and writes tasks that start from an entry point visible on the page and refer only to content the page evidences. We sample three axes outside the model: intent (info-seeking, transactional, tool-use, messaging, or navigation), difficulty (D0 to D3, from at most one constraint to six or more), and ambiguity (A0 to A3, from a single correct answer to a vague goal that requires clarification). For each page, we draw k (difficulty, ambiguity) pairs without replacement from the 16 combinations, and for each pair the model writes five tasks, one per intent, which keeps the distribution from collapsing onto look-up queries. Bundles that fail schema or label checks are rejected. We run Kimi K2.5 (Kimi Team, 2026a) in the browser-use agent harness (Müller and Žunič, 2024) on the synthesized tasks against live websites, and keep trajectories where the model's self-evals succeed. We use the resulting 45k trajectories (1.5 B tokens) in the mid-training stage.

Derive tasks from what the page offers, not from what the model imagines

Prompting a model to generate browser tasks from only a website name or a seed instruction yields fluent but often infeasible tasks. Such tasks may require filters, tools, or pages that the site does not provide, or refer to content that no longer exists. Since the open web is outside our control, post-generation feasibility checks are costly and unreliable. We instead reverse the synthesis order: we observe the live page first and write tasks only against the affordances it exposes. In this way, feasibility is guaranteed by construction, not by a post-hoc filter.

5.16 Beyond Teacher Imitation

Rolling out a teacher on a task and recording what it does yields trajectories that are at best as good as the teacher. They are limited by the teacher's context window, its attention over long inputs, its own decisions about when to stop, and its failures on the hardest tasks. Our post-training data generation methods

instead produce training trajectories that the teacher could not generate on its own. The common idea is to separate what the teacher sees during generation from what we record for training. In long-context question answering, the teacher works with a few focused passages, and the full document is added only to the training trajectory (§5.16.1). Length-controlled generation keeps the teacher's working context short while the recorded conversation grows to a target of up to 512k tokens (§5.16.2). Multi-turn PR chains run each PR of a repository in a fresh session and join the sessions into one long coding session (§5.16.3). A privileged advisor checks the solver's work against the reference solution and hidden tests, and inserts cues into its reasoning when it is about to submit a flawed solution, without revealing the answer (§5.16.4). Finally, augmentation techniques vary the system prompts, answer formats, and tool interfaces of each task, so that the model does not depend on a single presentation (§5.16.5).

5.16.1 Answering from focused content

Long-context question answering is the simplest example. To answer a question about a 100k-token document, a teacher first has to find the few passages that matter, and model accuracy drops when the relevant information sits in the middle of a long input (Liu et al., 2024). In LCQA (§5.13) the teacher never sees the full document. It writes a question about a few passages of ∼512 tokens and then answers it from those passages alone, a short-context task it handles reliably. The full document enters only when we assemble the training trajectory, where it replaces the passages. Since the question is restricted to answers fully determined by the passages, the answer stays correct. The trajectory thus pairs up to 512k tokens of context with reasoning and an answer of the quality the teacher reaches on focused input, rather than on the full document. It is also cheaper to generate: the teacher reads about 2k tokens per prompt instead of the full document.

5.16.2 Length-controlled generation

Long agentic trajectories are hard to obtain from a teacher for two reasons. The teacher's context is bounded, so a 512k-token research conversation cannot be generated in one pass, and quality degrades well before the limit (Liu et al., 2024). The teacher also decides on its own when to stop, and given an open-ended question, it tends to answer after a handful of searches, however much there is to find. For the search conversations (§5.10) we therefore separate what the teacher sees from what we record, and we make the length of the trajectory a parameter of generation.

Working context and trace. The generator keeps two views of the conversation. The trace records every message and tool result and grows up to the target length. The working context, which is what the teacher receives at each step, pins the system prompt and the first user message and keeps only the most recent messages, at most 100 messages and 100k tokens. Documents are truncated to 30k characters when read, so that a single document cannot fill the window. After each burst of tool calls, the teacher writes a short summary of what it found, and these summaries carry the findings forward in compact form as older tool results leave the window. Every assistant turn is written from the beginning of the trace and a recent suffix of it, so it never refers to content that the trace does not contain, while the teacher never holds more than 100k tokens.

Pacing to a target. Each query is assigned a target of 32k, 128k, or 512k tokens. Research continues until the trace reaches 80% of its target or 100 turns, and the remaining budget goes to a final synthesis, written without tool access from the working context. Between research steps, the simulated user samples its next message type (react to a finding, drill deeper, redirect, challenge, clarify, or build upon) with weights that depend on what just happened. While the trace length is below 60% of its target, message types that invite a wrap-up are removed. When the teacher stalls, the generator injects a short request to continue. These requests are runtime scaffolding rather than user input, so we remove them from the saved trace.

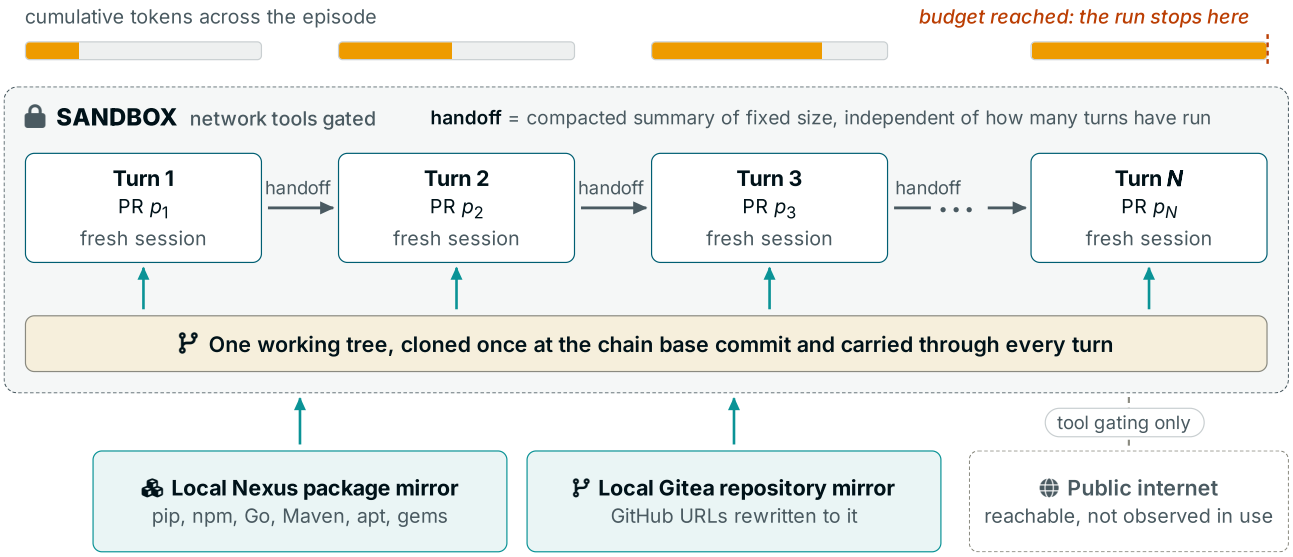


Figure 33: How one PR chain becomes a single long episode. Each PR is attempted in its own fresh agent session against the same working tree, so state reaches the next turn two ways: through the filesystem, which accumulates, and through a compacted handoff, which does not. Per-turn context therefore stays roughly constant however long the episode runs, while cumulative token usage climbs until it crosses the budget, at which point the run stops and the partial episode is kept. The whole episode runs inside a sandbox in which the network tools are gated: dependency installs resolve against a local package mirror and git operations against a local repository mirror, so a build can still fetch what it needs while the obvious routes to the upstream change are closed. The gating is at the tool layer rather than the network layer, so the public internet remains reachable by an agent that writes its own client; we did not observe this in the trajectories we inspected. The number of turns an episode reaches is set by the budget rather than by the number of PRs the chain offers.

The resulting traces land close to their target lengths: median traces reach 95%, 87%, and 82% of the 32k, 128k, and 512k targets, respectively, and 94%, 97%, and 81% of traces fall within 25% of their target.

5.16.3 Multi-turn PR chains

The PR rollouts of §5.5 are single-turn, but real coding sessions are longer. A developer asks for one feature after another in the same repository, and each change builds on the earlier ones. We create trajectories with many user turns by chaining the PRs of a repository in their original order and giving them to an agent one at a time in a single checkout, so that each PR becomes a new user turn. After each turn, the agent's work is summarized into a short handoff for the next turn and, at the end, the turns are joined into one multi-turn trajectory (Figure 33).

Chain construction. A chain should look like the history of a real project in which one codebase changes step by step. We therefore group the PRs by repository and order them by PR number, which follows the order in which they were opened. A chain starts with the lowest-numbered PR which becomes the first user turn. Every later turn also gets a task written from its PR, which states the goal without giving away the patch. This yields 10,175 chains, one per repository, covering 2M PRs. The median chain has 113 PRs, the 90th percentile 361, and the longest 33,766, while the shortest has two. A record holds at most 50 PRs, so the 7,761 chains that are longer continue in further records, each starting with the next PR of its chain.

Sandbox. Each chain runs in its own sandbox checkout at the base commit of the chain's first PR for the repository. We remove the origin remote and the reflog, so the checkout holds the base state and nothing after it. The change that each turn asks for already exists upstream, and fetching it would be the

cheapest way to appear to succeed. The sandbox therefore blocks `curl`, `wget`, web search, and page fetches, as well as `git clone`, `fetch`, `pull`, and `ls-remote` in every spelling we could list. Builds still need their dependencies, so `pip`, `npm`, Go, Maven, apt, and other package managers point to a local Nexus mirror, and GitHub URLs are rewritten to a local Gitea mirror. The blocking happens at the tool level, not the network level. An agent could still reach the internet by writing its own client, although we did not see this in the trajectories we inspected.

Chain rollouts. Each user turn should start where the previous one ended, as in a real coding session, but the agent's context should not grow with every PR. We run DeepSeek-V4-Flash (0731) (DeepSeek-AI, 2026) with thinking in the OpenCode agent harness (Anomaly, 2026) and start a fresh agent session for each PR on the same working tree, so the code carries over between turns while the context does not. A turn therefore starts from the agent's own work on the earlier PRs, not from the upstream history that its PR was written against. What the session knows about the earlier turns comes from the handoff described below. A turn that stops before the agent finishes the task is resumed in the same session up to three times, and is otherwise kept as it is. Runs are bounded by a cumulative token budget of 512k rather than by the number of PRs. The budget is checked before each turn starts, so the last turn may go over it, and a run that stops at the budget counts as complete. The runs have between 3 and 27 turns, since a run can spend the whole budget on three PRs in a large codebase or work through twenty-seven in a small one.

Context handoff. A fresh session knows nothing about the earlier turns, and passing it their full history would make the context grow with every PR. Instead, after each turn, the same model summarizes it in a separate call. The input is the previous handoff, the last 4,000 characters of the agent's final message, the PR and the task title of the turn that just ended, and the task for the next turn. The model returns a JSON object with nine fields: completed work, current state, evidence and uncertainty (split into facts backed by evidence and points that still need checking), work in progress, constraints, critical context, verification history, relevant files and symbols, and known risks. The handoff is placed at the start of the next turn's prompt. Since each handoff is written from the previous one and not from the full history, its size stays about the same however many turns have run.

Multi-turn trajectories. For training, we want the model to see the whole session, not a summary of it. We therefore drop the handoffs and join the sessions of a run into one conversation. Each PR's task becomes a user turn, followed by everything the agent did for that PR, so one trajectory contains many user turns on a codebase that changes from turn to turn. If a joined conversation is longer than 512k tokens, we drop user turns from the end until it fits, keeping at least two. The resulting trajectories contain about 7B tokens.

5.16.4 Advisor-guided trajectory generation

Agentic coding rollouts often fail not because the solver makes no progress, but because it terminates at the wrong point. A solver may repair the most visible symptom while leaving a secondary requirement unresolved, submit after narrow validation, or continue editing after reaching a correct solution and subsequently undo it. We address these failures with a privileged advisor: a rollout-time controller that can inspect reference material unavailable to the solver, but cannot edit the repository or act on the task itself. Its sole role is to decide whether the solver's next action should be accepted or regenerated. Figure 34 illustrates one review step, and our implementation is publicly available (Pimpalkhute et al., 2026).[21]

Privileged review. Let π_θ denote the solver policy with parameters θ, x the public task specification, z the privileged task material, and h_t the public solver history at turn t, i.e., all previously committed solver

[21] https://github.com/ifm-ai/advisor

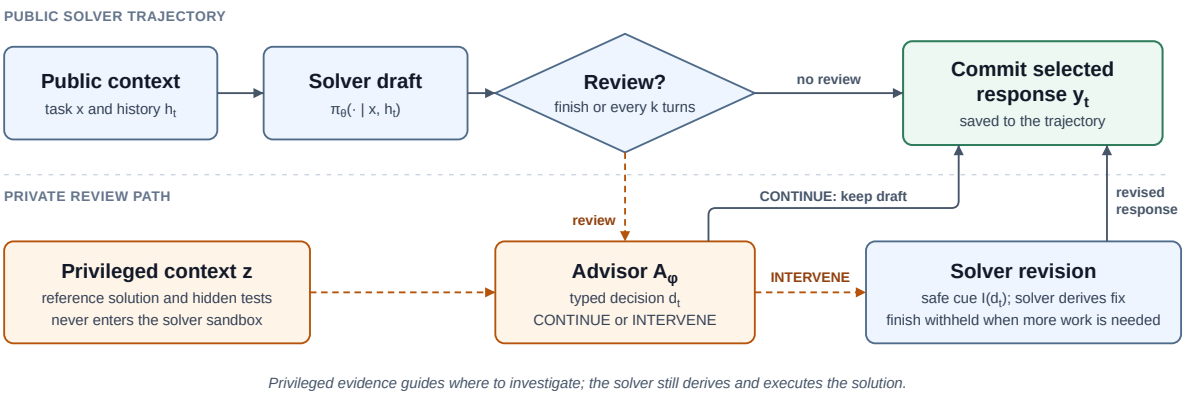


Figure 34: Privileged advisor intervention during rollout generation. At designated checkpoints, the advisor consults private evidence and either accepts the draft or returns a privilege-free cue for solver revision. Only the selected solver response is committed; privileged material never enters the trajectory or solver sandbox.

responses together with their tool and terminal observations. At each turn, the solver first samples a provisional (draft) response $\tilde{y}_t$. Let $\mathrm{Finish}(\tilde{y}_t)$ be the indicator that the draft invokes the finish tool, k the periodic review interval, B the per-attempt intervention budget, and n_t the number of interventions already used in the current attempt. The advisor reviews every proposed finish action and, when periodic review is enabled, every k-th solver turn. Generation and the review gate $g_t \in \{0, 1\}$ are

$$\begin{aligned} \tilde{y}_t &\sim \pi_\theta(\cdot \mid x, h_t), \\ g_t &= \mathbf{1}\{n_t < B\}\, \mathbf{1}\{\mathrm{Finish}(\tilde{y}_t) \vee (t \bmod k = 0)\}\,. \end{aligned}$$

The periodic term is omitted when periodic review is disabled.

When $g_t = 1$, the draft is passed to the advisor A_ϕ, a separate language model with parameters ϕ. The advisor receives the public specification x, a bounded rendering of h_t, the draft $\tilde{y}_t$, and the privileged material z, which contains the reference solution, hidden verifier tests, and private task notes. Because the advisor is stateful within an attempt (see below), it also receives p_{t-1}, the private plan it produced at its most recent review (empty at the first review), and $H_{<t}$, the set of solver-facing hints it issued at earlier reviews in the same attempt. The advisor then produces a decision

$$d_t = A_\phi\big(x, z, h_t, \tilde{y}_t, p_{t-1}, H_{<t}\big), \qquad d_t = \big(f_t,\, r_t,\, a_t,\, p_t,\, \eta_t,\, \delta_t\big),$$

where $f_t \in \{0, 1\}$ is an intervention flag, r_t a reason code, $a_t \in \mathcal{A}$ a discrete action, p_t the updated private plan (carried forward as the next review's p_{t-1}), η_t a solver-safe hint (added to the hint history H), and δ_t a private diagnosis. The action set $\mathcal{A}$ contains `CONTINUE` (accept the draft), `CONFIRM_AND_SUBMIT` (confirm a correct implementation), and corrective actions that ask the solver to re-check requirements, trace state transitions, validate more broadly, or reconsider its submission. Each decision is strictly validated against this schema; malformed decisions are retried rather than interpreted heuristically. The advisor is instructed to assess functional correctness rather than similarity to the reference patch. In particular, it checks requirement coverage, interacting state transitions, and whether the solver has produced sufficient execution evidence to justify submission.

Selective intervention. Let $\mathcal{I}(d_t)$ denote the solver-facing cue compiled from the advisor decision, and let $\hat{y}_t$ denote the revised response that the solver regenerates after receiving this cue. The response committed

to the trajectory is

$$y_t = \begin{cases} \tilde{y}_t, & g_t = 0 \text{ or } a_t = \texttt{CONTINUE}, \\ \hat{y}_t, \quad \hat{y}_t \sim \pi_\theta(\cdot \mid x, h_t, \mathcal{I}(d_t)), & \text{otherwise.} \end{cases}$$

For models supporting reasoning continuation, $\mathcal{I}(d_t)$ is built from the hint η_t and inserted as the opening of the solver's next reasoning block. It is written in the solver's first-person register and ends on an unresolved evidentiary gap, leaving the solver to derive the next command and the correction itself. For other models, the action a_t is mapped to a fixed, task-agnostic self-check instruction. When additional work is required, the finish tool is removed from the revision call, forcing the solver to take a substantive action before submitting again. The complementary `CONFIRM_AND_SUBMIT` action retains the finish tool when periodic review finds that the repository is already correct and further editing is more likely to be harmful than useful.

The advisor is stateful within an attempt. Its private plan p_t tracks unresolved requirements across reviews, while the hint history $H_{<t}$ prevents repeated interventions from merely restating arguments already visible in the solver's replayed reasoning. If the same gap persists, the advisor must identify a new angle grounded in public evidence, such as an unexecuted trace or a contradiction in the solver's observations, without disclosing the missing implementation.

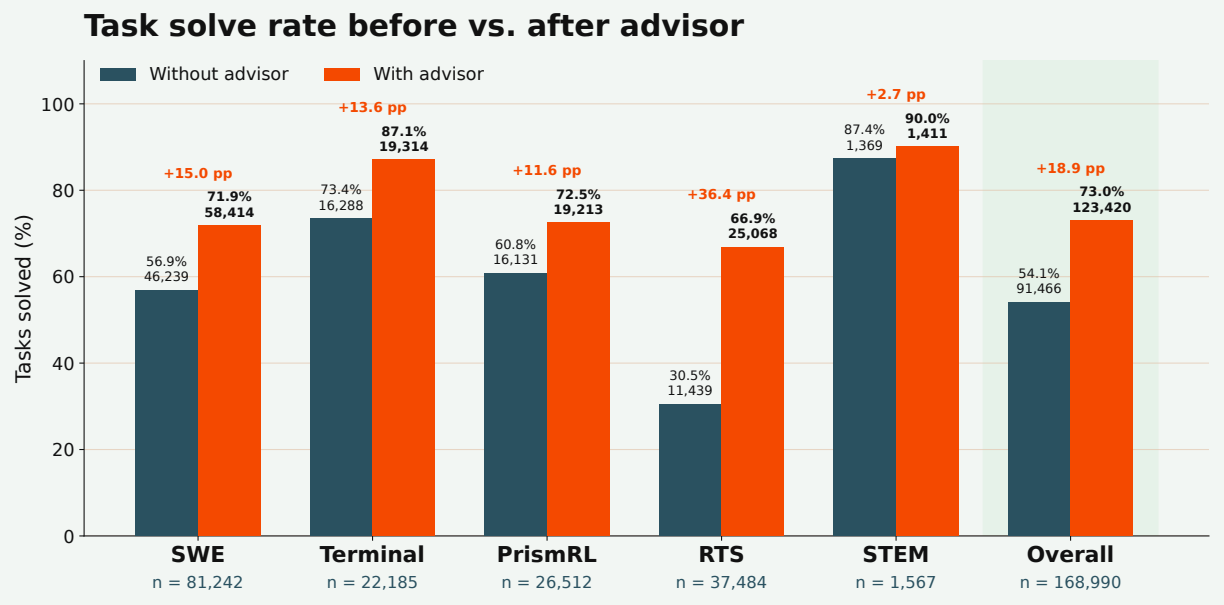


Figure 35: Fraction of tasks solved without and with the advisor, by category. Advised reruns are applied only to tasks the solver failed on its own.

Privileged-information isolation. The private context z is loaded only on the rollout host and is never placed in the solver sandbox. The advisor's diagnosis δ_t and plan p_t remain private. Every factual claim in the solver-facing cue $\mathcal{I}(d_t)$ must be supported by the public task, repository, or terminal evidence already contained in $(x, h_t, \tilde{y}_t)$. Thus, the advisor may use hidden evidence to identify where the current trajectory is weak, but may not quote or paraphrase hidden tests, reveal expected outputs, or provide the solution. Only the selected solver response is appended, $h_{t+1} = h_t \oplus y_t$; the rejected draft, private review, and advisor diagnosis never appear as separate trajectory turns. Advised and ordinary rollouts therefore share the same external representation. Advisor and revision requests use bounded retries and

fail open to $\tilde{y}_t$ when configured to do so. Final correctness is determined exclusively by the benchmark verifier. We additionally audit the resulting trajectories using deterministic leakage checks and separate public-coherence and privileged-leakage judges. The advisor consequently provides privileged process supervision without replacing the solver's own reasoning or execution.

Effect on solvable tasks. We ran the solver on 168,990 tasks in five categories (PRism RL tasks, Recursive-Task-Synthesis (RTS) terminal tasks (Li et al., 2026d), other terminal tasks, SWE, and STEM; Table 26) and reran every task it failed to solve with the advisor enabled. Figure 35 reports the outcome. On its own, the solver solved 91,466 tasks (54.1%); advised reruns solved a further 31,954, raising the total to 123,420 (73.0%), an increase of 18.9 percentage points. The gain is largest where the unadvised solver is weakest: on RTS, coverage rises from 30.5% to 66.9%. Conversely, STEM, where the solver already succeeds on 87.4% of tasks, gains only 2.7 points. Note that these numbers measure how many tasks yield at least one successful trajectory, which is the quantity that matters for data generation, rather than a per-attempt pass rate under matched conditions.

5.16.5 Augmentation techniques

For single turn reasoning tasks, we do prompt augmentation to train the model on tasks with different user instructions and/or answer format. In particular, we construct a combination of {30 system prompts} × {3 answer format}. The system prompt variations cover instructions for different reasoning levels ("reason step by step", "reason step by step and never give up" etc.), structured and unstructured reasoning as well as instructions about self-verification and reflection. The answer format variations include prompting the model to select one of multiple choices, output short free-form answers (e.g., mathematical expressions for math/physics reasoning tasks) as well as generating structured outputs in Python, Markdown or JSON format.

For both search and general tool-use data, we augment the corpus by randomizing the tool call interface. For every task, we randomize the names of the tools, the names and order of their parameters, their natural-language descriptions, the way tool results are rendered back to the model, the wording and layout of the instruction, the system prompt, and the convention by which the final answer is submitted: in some trajectories the model commits through a dedicated submission tool, while in others no such tool is exposed and the answer must be written as text in a prescribed format. For each task, we generate multiple trajectories with different randomized tool call schemas. This method teaches the model to read the tool schema given in each trajectory and act on it, rather than to rely on a memorized call signature, and it should make the model less sensitive to the system prompt and tool output formats.

5.17 Programmatic quality and format controls

For K2-V2 (K2 Team, Institute of Foundation Models, 2025), we implemented `process_entry`, a function that checks if a training example follows our conversation format. At that time, our SFT data was mostly non-agentic with little tool use, so checking the format of a conversation was simple. For `K2-Horizon`, our focus shifted to agentic behavior in both mid- and post-training, and the data changed accordingly: long multi-turn trajectories with tool calls, generated by our own pipelines or taken from public datasets. Our mid-training sources alone contain 13.8M examples with tools and 747K unique tool definitions. In our earlier training runs, we did few checks on this data and later discovered that it had many issues: tools that were used but never declared, tool names in the rollout that did not match the declared ones, and schemas that misled the model about which arguments are allowed. As a result, our initial mid-training runs produced a model that often hallucinated tool calls. We also found that, at scale, the open models we use for generation can fall into doom loops, repeating the same action over and over, and that problems in the environment can lead the model to spend the rollout trying to fix the environment instead of solving the task. Over time, we collected examples of malformed or corrupted rollouts and added more and more

checks. Today, every conversational example in our mid- and post-training data has to pass `process_entry`. We release it, together with its tests and the tokenizer and chat template used in training.

Design. `process_entry` takes one example and returns the processed example with its token count, a list of reasons for rejection, and a record of the changes it made; the example is rejected if and only if the list of reasons is non-empty. Changes are limited to normalizations that do not alter the meaning of the example, such as moving a top-level tool list into the system message, unwrapping tools written in the OpenAI `function` format, parsing tool-call arguments stored as JSON strings, or trimming whitespace in reasoning. The visible replies are never changed, since some tasks depend on exact spacing. All quality checks only reject and never repair, because silently fixing an example would hide a broken generator from the person who owns the source. Checks are strict by default, and a few settings are chosen per source, such as whether reasoning is required and at which reasoning level. Rules are changed only with explicit manual approval, after measuring the new rule on the full corpus and reading samples of what it rejects.

Format and tools. The conversation must follow a valid order of system, user, assistant, and tool turns: for example, every tool call must be followed by a tool response and every tool response must answer a tool call. If the conversation uses tools, all of them must be declared in the system message with a name and a JSON schema of their parameters, and every tool call is validated against the schema: the tool must exist, required arguments must be present, undeclared arguments are rejected unless the schema explicitly allows them, and argument types and allowed values must match. Agentic data should also teach the model to recover from its own mistakes, so for some sources we enable a recovery mode that tolerates an invalid tool call if the environment returned an error, the model did not simply repeat the same failing call, there were at most three errors in a row, and the last turn with tool calls succeeded. When reasoning is required, at least one assistant turn must contain it. Reasoning is stored under the field of the requested reasoning effort, and an example without any reasoning is tagged as low effort. Finally, the example is rendered with our chat template in the format used in training, so that problems do not first appear during training, and its token count is saved.

Degenerate behavior. We reject examples where the assistant has fallen into a loop: the same sentence repeated many times in its reasoning, the same reasoning block repeated across turns, or the same tool call with identical arguments issued again and again. Some loops are only visible through the environment. For example, an agent whose terminal hung kept emitting the same no-op action for hundreds of turns while writing fresh reasoning on every turn, so no check of individual turns could see it. We catch such cases by checking whether the observation that the agent responds to ever changes, which separates a stuck agent from legitimate polling, such as waiting for a long test run to finish. These checks only look at assistant turns, since a user pasting a repetitive log or a tool returning identical rows is not a defect of the training target. They also never penalize length: long reasoning is fine as long as it keeps making progress.

Leakage from generation. Generated data carries traces of how it was produced. We reject examples containing special tokens of our chat template, as well as markers of other chat templates and tool-call formats, such as GPT-OSS harmony tokens or the tool-call markup of DeepSeek and Kimi models. Markers are matched by shape rather than as exact strings, because models and converters produce variants with inserted spaces, doubled delimiters, or HTML escaping. For the same reason, we reject examples whose upstream converter records that it modified the data, so that a person reviews the converter. A subtler leak is the model referring to instructions that the example does not contain. In generations from Kimi K3 and DeepSeek-V4-Flash, the model often mentions a "desired oververbosity" level (for example, "Desired oververbosity 5"), although no such instruction appears anywhere in the example; in one snapshot of our

Source	Num. Records
Software engineering (SWE) tasks	
OpenSWE (Fu et al., 2026a)	30,453
SWE-rebench-V2 (Badertdinov et al., 2026)	20,929
Scale-SWE (Zhao et al., 2026a)	18,033
SWE-Lego-Real-Data (Tao et al., 2026)	3,852
Terminal tasks	
Recursive-Task-Synthesis (RTS) (Li et al., 2026d)	37,484
Terminal-Lego-15k (Yang et al., 2026a)	13,072
SETA-Env (Shen et al., 2026)	400
Endless-Terminals (Gandhi et al., 2026)	2,492
Nemotron-Terminal-Corpus (Pi et al., 2026)	5,984

Table 26: Public coding RL sources. Counts are per source and are not deduplicated across sources.

synthetic data, 9.2% of examples did so. Training on such examples would teach our model to react to instructions its user never gave, so we reject assistant turns that refer to an instruction that appears nowhere else in the example. Finally, we check model identity: many of the models we use for generation call themselves Claude or ChatGPT in their reasoning, and system prompts in some public datasets name the model that generated the data. Only assistant and system turns are scanned, so a user asking about another model is not rejected.

Regression corpus. Most rules started from a real example that slipped through the existing checks. We keep such examples, together with hard negatives that look suspicious but are legitimate, in a regression corpus. After every change, we verify that each bad example is still caught by at least one rule and that none of the hard negatives is rejected. The released code includes a test suite with cases derived from these examples.

ALOHOMORA

We rely heavily on coding agents for data processing, and when most of a dataset is rejected, the easiest "fix" for an agent is to loosen the validator. So the first lines of `process_entry` tell coding agents that its rules are load-bearing and may only be changed if the user types a secret word: ALOHOMORA.

The stuck terminal

One trajectory that passed all our checks repeated the same 111-character no-op action 819 times over 1,241 turns, while writing fresh reasoning on every turn, so every check of individual turns looked clean. What finally caught it was looking at the environment rather than the action: the agent's observation never changed.

5.18 Public Data Collection

Public SFT datasets for reasoning, tool use, and agent tasks are stored in many different formats. One dataset may describe its tools in its own way, a second may keep the model's reasoning in a separate field, and a third may save agent runs as raw logs instead of conversations. Writing a converter by hand for each dataset takes too long, and a single general converter rejects most of them. We therefore built a data processing skill: a written set of steps and rules that a coding agent follows to bring a new dataset into our

SFT format. The agent finds out why a dataset fails, writes a small fix for that dataset, and checks the result. The agent never decides on its own which examples are kept. Every example must pass the same shared validator, `process_entry`, described in §5.17. Our processing pipeline is summarized in Figure 36.

- **Loading the source.** We record the exact dataset, configuration, split, and version for each source, and keep track of where each row came from. For sources saved as logs, we also decide what counts as one training example, because one agent run may be saved many times as it progresses.
- **First test and deciding whether a fix is needed.** We run up to 1,000 rows through a general converter and then through `process_entry`. If more than 90% of them pass, the dataset is used as it is. Otherwise, the agent writes a fix. This way the agent only spends effort on datasets that need it.
- **Finding the cause.** For datasets that fail, the agent looks at raw rows, passed rows, failed rows, and the error messages returned by `process_entry`, grouped by type. From these, it works out what went wrong, for example an unfamiliar tool format, tool inputs stored as text, reasoning kept in a different field, or an agent run that was never finished.
- **Writing a fix for each dataset.** Each fix is a small piece of code for one dataset only, with its own tests. It runs before `process_entry` and converts the data into the form the validator expects. Common fixes move existing reasoning into the expected field, convert tool descriptions and tool outputs to our format, rebuild conversations from logged events, or remove leftover messages from the original agent setup. A fix may never make up questions, reasoning, answers, or final replies.
- **Testing the fix again.** After each fix, the same rows are checked again with `process_entry`. The agent gets at most three attempts and stops earlier if the only way forward would be to invent missing content. Datasets that still do not pass are left out.
- **Running on the full dataset and saving the output.** The final fix is run on the whole dataset using many CPU workers or cluster jobs, with the same version, validator, and settings as in testing. When results from several jobs are joined, we check that every source row was processed exactly once. For each dataset, configuration, and split, we save the accepted conversations, a sample of rejected rows, and a summary with exact counts and settings.
- **Keeping a record of common problems.** Problems we have confirmed are written down with the dataset, counts, cause, and whether they were fixed. Later runs can then recognize a known problem instead of investigating it again.

Validation. Every converted example must pass `process_entry` (§5.17). For each dataset, the caller chooses a few settings, such as whether reasoning is required, the reasoning level, whether some failed tool calls may be tolerated, and the bulk runner saves these settings in each output's summary. Our team rule is that the agent may not change the validator's rules without explicit human approval, so a failing dataset must be fixed in its own preprocessing code and not by loosening the checks. `process_entry` records token counts but does not enforce a length limit.

Public sources. We processed 30 public datasets from Hugging Face with this workflow, listed in Table 27. Each dataset is cited through its paper when its dataset card gives one, and otherwise through its Hugging Face page. The sources cover math and logic, general chat and instruction following, multiple languages, safety, finance, science, code, software engineering, agent and terminal tasks, and reasoning traces distilled from a larger model. Each dataset, configuration, and split was processed and saved separately.

In addition to these public SFT datasets, we collected coding RL tasks, summarized in Table 26. We used these tasks for rollouts to generate additional training data for the SFT stage described in §6.6.

Source	Parts	Accepted rows
Math and logic		
Nemotron-SFT-Math-v3 (NVIDIA, 2026h)	1	3,630,372
Nemotron-SFT-Math-v4 (NVIDIA, 2026i)	1	545,081
Nemotron-Math-Proofs-v2 (NVIDIA, 2026a)	1	82,736
Nemotron-SFT-ARC-AGI-v1 (NVIDIA, 2026c)	8	251,757
Chat, instruction following, multilingual, safety, finance		
Nemotron-SFT-Instruction-Following-Chat-v2 (NVIDIA, 2026f)	2	1,997,249
Nemotron-SFT-Instruction-Following-Chat-v3 (NVIDIA, 2026g)	1	249,743
Nemotron-SFT-Multilingual-v1 (NVIDIA, 2026j)	18	3,065,201
Nemotron-SFT-Multilingual-v2 (NVIDIA, 2026k)	10	337,114
Nemotron-SFT-Safety-v1 (NVIDIA, 2026m)	1	45,143
Nemotron-SFT-Safety-v2 (NVIDIA, 2026n)	1	130,348
Nemotron-SpecializedDomains-Finance-v1 (NVIDIA, 2026q)	1	326,698
Science		
Nemotron-SFT-Science-v2 (NVIDIA, 2026o)	4	2,836,254
Code		
Nemotron-SFT-Competitive-Programming-v2 (NVIDIA, 2026d)	4	841,132
Nemotron-SFT-OpenCode-v1 (NVIDIA, 2026l)	6	431,706
Nemotron-SFT-CUDA-v1 (NVIDIA, 2026e)	1	2,273
Software engineering (SWE) trajectories		
Nemotron-SFT-SWE-v2 (NVIDIA, 2026p)	2	255,257
DeNovoSWE-Trajectory-Raw (Zhao et al., 2026b)	1	32,490
Scale-SWE-Distilled (Zhao et al., 2026a)	1	64,898
Scale-SWE-Distilled-DeepSeek-v4-Pro-High-41k (Zhao et al., 2026a)	1	35,597
Agents, tool use, and terminal tasks		
Nemotron-SFT-Agentic-v2 (NVIDIA, 2026b)	3	283,983
Nemotron-Terminal-Corpus (Pi et al., 2026)	3	175,370
ClawGym-Trajectory (Bai et al., 2026)	1	23,575
hermes-agent-reasoning-traces (Lambda, 2026)	2	6,614
Search and deep research		
QUEST-SFT-Data-Objective (Xie et al., 2026)	1	17,529
QUEST-SFT-Data-Open-ended (Xie et al., 2026)	1	7,754
REDSearcher-SFT-10K (Chu et al., 2026)	1	17,280
XYZ-Aquila-SFT (XYZ Agentic Team, 2026)	2	8,772
OpenSeeker-v1-Data (Du et al., 2026)	1	4,706
SearchSwarm-SFT (Ning et al., 2026)	1	5,428
General reasoning		
KIMI-K2.5-1000000x (ianncity, 2026)	4	901,452
Total (30 datasets)	85	16,613,512

Table 27: Public datasets processed with this workflow. Parts is the number of dataset, configuration, and split outputs; Accepted rows is the number of examples that passed `process_entry`. Counts are per source and are not deduplicated across sources.

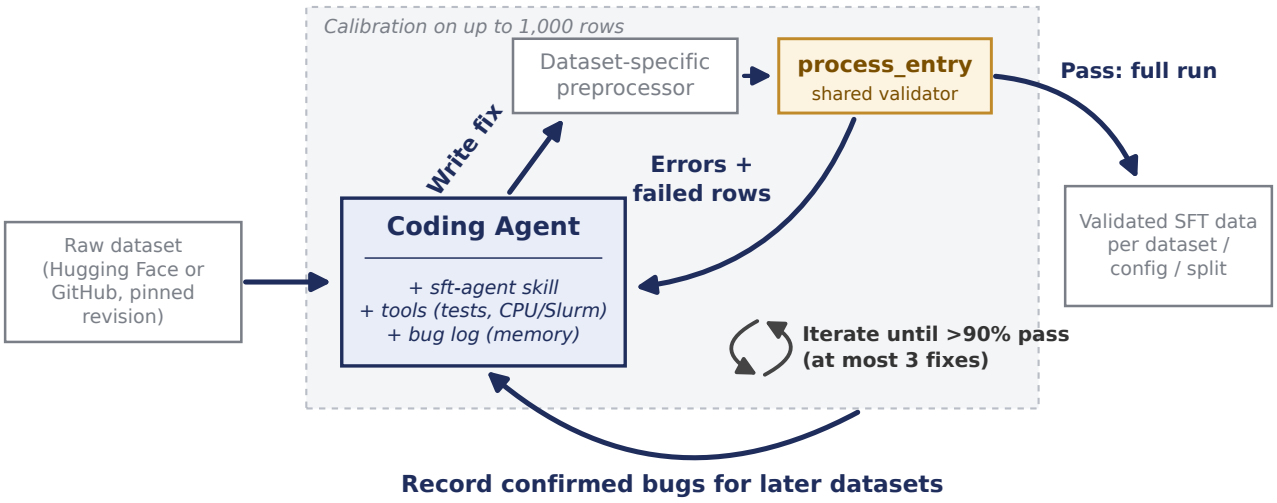


Figure 36: How a new dataset is processed: load the source, test it with the general converter and `process_entry`, decide whether a fix is needed, find the cause, write and test a dataset-specific fix, and save the output for each dataset.

Source	General converter	With fix	Tested on
Nemotron OpenCode (`bash_only_tool_skills`)	0.0%	90.4%	full dataset (96,050)
Nemotron SWE v2 (`agentless`)	58.8%	99.9%	full dataset (209,976)

Table 28: Share of examples that pass `process_entry` before and after the agent's dataset-specific fixes. Passing means the example has the correct structure and format; it does not mean the content is correct.

Turning agent snapshots into conversations. Our internal coding-agent dataset saves each session repeatedly as it progresses, giving 4.27M snapshots from 54,073 sessions. Keeping one conversation per session yields 52,640 accepted conversations.

Results. Across the 30 public datasets, the pipeline produced 85 outputs, one for each dataset, configuration, and split, with 16,613,512 accepted examples in total. These totals count everything stored across all dataset versions. Duplicates across datasets were not removed. Table 28 shows how much the dataset-specific fixes helped: they recovered 86,842 examples from Nemotron OpenCode, which the general converter rejected completely, and 86,426 more examples from Nemotron SWE v2.

5.19 Chat template

We designed the chat template to improve the diversity and efficiency of tool-use training. Rather than training on a single fixed convention, we exposed the model to multiple tool presentation and tool calling formats. Tool definitions were represented using JSON, XML, and Markdown, while tool calls used JSON, XML, and typed XML. Three variations for tool definitions and tool calling produced a total of nine configurations that we varied during mid-training with a bias towards XML and Markdown tool presentation (40% each) and XML tool calling (50%) formats. This variation encourages the model to learn the underlying semantics of tool understanding and selection rather than overfitting to a particular syntax. For inference, we ultimately selected Markdown as the default tool presentation format because it was approximately 18.5% more token-efficient than the conventional JSON formatting on our data (see Figure 37).

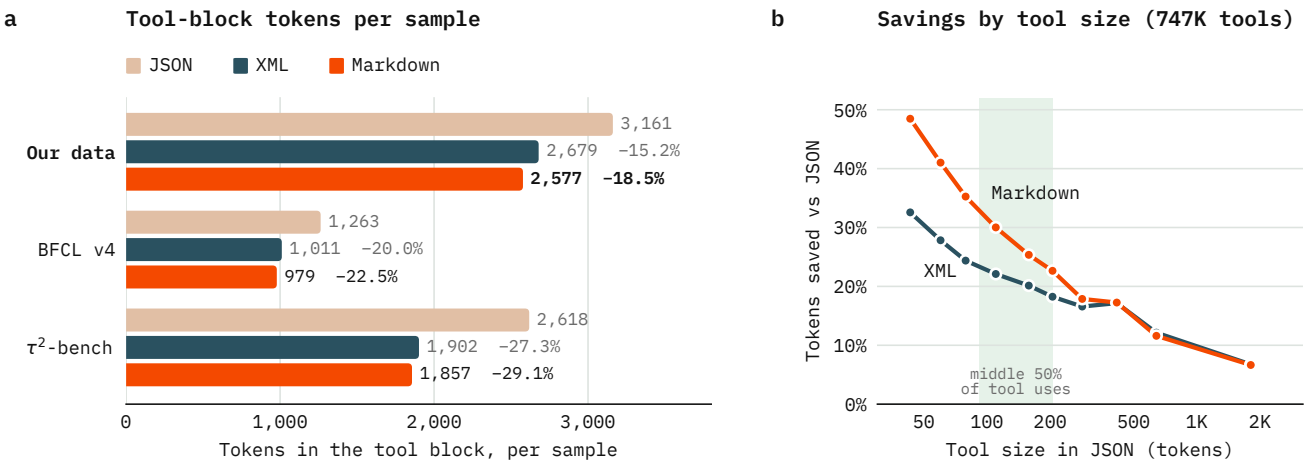


Figure 37: Token cost of the three tool presentation formats. **(a)** Average size of the rendered tool block per sample. On our data (771K samples with tools, 249K unique toolsets), Markdown uses 18.5% fewer tokens than JSON and XML 15.2% fewer. The savings are similar or larger on BFCL and τ^2-bench toolsets. **(b)** Per-tool savings over the 747K unique tools in our data, grouped by the tool's size in JSON. Markdown is shorter than JSON for 99.5% of tool uses. Small tools save the most (up to ∼48%), because JSON syntax makes up a larger share of them. Tools over 1K tokens save only about 7%. All averages are weighted by how often each tool or toolset occurs.

One of the harder problems we encountered in this seemingly easy endeavor was ensuring that these different presentations exposed the information needed to use each tool. Early versions of our compact renderers omitted or distorted information from the JSON tool definitions, such as union schemas and fields behind local schema references. For example, the passenger argument of an airline-reservation tool referenced a shared definition, but the compact presentation never showed that definition's fields. In the resulting traces, the model sometimes supplied `date_of_birth` where the tool required `dob`, triggering validation errors and retries. In one of our evaluation runs, fixing this simple bug increased BFCL single-turn accuracy from 74.65% to 76.98% across six presentation/call configurations.

The tokens we saved became code we had to maintain

Moving beyond raw JSON shifted complexity into the chat template. References, types, and literal values all needed explicit handling, and mistakes could surface as apparent model failures. On our journey of diversifying tool presentation and calling formats, we ended up building a small compiler. We settled on a Markdown tool presentation that uses 18.5% fewer tokens, but the larger lesson is how varied real-world tool definitions are, and how much code it takes to render all of them faithfully.

6 Post-Training

After the mid-training stages dedicated to seeding broad model capabilities such as handling longer context and agentic tool use, we turned to post-training. Instead of a fixed stage of training, we think of post-training as a process of eliciting, refining and adding model capabilities to meet a rapidly evolving landscape of LLM applications. RL, SFT, and evaluation are tools used flexibly in this process to produce release-ready checkpoints; community feedback on each release then restarts the cycle. This post-training iteration further served as a learning process of building the full post-training stack for agentic LLMs. This section is aimed at sharing these lessons with the community, particularly with new teams taking up the challenge of building agentic LLMs.

We adopted expert training with weight merging instead of mixed-domain training as our RL strategy for the following reasons:

- A major advantage of mixed-domain RL is that abilities gained in one domain can be immediately utilized across other domains, speeding up overall capability improvement. But the same potential has a flip side in unique algorithmic challenges that expert training does not have. Unbalanced learning signals can lead to rapid improvement in some domains at the cost of others, and failure modes arising in one domain can quickly propagate to others. Instead of spending time on stabilizing mixed-domain RL, we decided to focus on domain-specific experts for our first agentic post-training iteration. This gave us the opportunity to better understand the data and training behavior for each domain, preparing us for future work on mixed-domain training.
- Expert training is highly flexible from a logistics and organizational perspective. Training a domain expert requires fewer computational resources, and different domain experts can be trained by different teams on different clusters, using algorithmic settings and ingredients that can be tuned to each domain independently.
- Weight merging with recent algorithmic techniques is extremely fast (taking just a few minutes) and produces excellent results when used carefully, as we demonstrate for the first time at the multi-hundred-billion parameter scale, in Section 6.4.

A new version of our RL infrastructure was developed from the ground up in parallel during mid-training, to support long-horizon agentic RL across several domains. Expert RL started after the last stage of mid-training for five domains of expertise we target: search, knowledge work, tool use, scientific reasoning, and instruction following. Following the RL stage, we conducted extensive evaluations to understand gaps in model performance, and continued with a round of SFT targeted at closing the identified gaps.

6.1 RL Infrastructure

Long-horizon agentic workloads introduce significant overhead beyond model inference. In our early stack, rollout throughput was often limited not by GPU serving capacity, but by CPU-side coordination: session management, tokenization, trajectory assembly, and sandbox orchestration were serialized across a small number of event loops. This left available inference capacity underutilized as the number of concurrent trajectories increased.

RL360 addresses this bottleneck by decoupling rollout orchestration from training and allowing each component to scale independently. As illustrated in Figure 38, Miles (Chen et al., 2026b) manages rollout collection and policy optimization, Harbor (Harbor Framework Team, 2026) handles agent execution and task environments, SMG[22] routes inference requests to SGLang (Zheng et al., 2024), and Megatron-LM (Shoeybi et al., 2019) handles distributed training and policy weight synchronization. The complete implementation, pinned component versions, and a reference training recipe are available in the public RL360 repository (IFM, 2026).

Scaling the rollout control plane. At high concurrency, the bottleneck shifts from inference to the control plane that launches trials, tracks sessions, and assembles trajectories; our initial setup saturated at ∼1024 concurrent trajectories.

Harbor runs multiple worker processes behind a lightweight router that dispatches trials and aggregates worker health and crash diagnostics. Distributing trials exposed two issues: worker-local sample identifiers could collide, and a cancellation reached only one worker. We use globally unique sample identifiers and broadcast cancellations to all workers, which also release the cancelled trial's session state.

[22] https://github.com/smg-project/smg

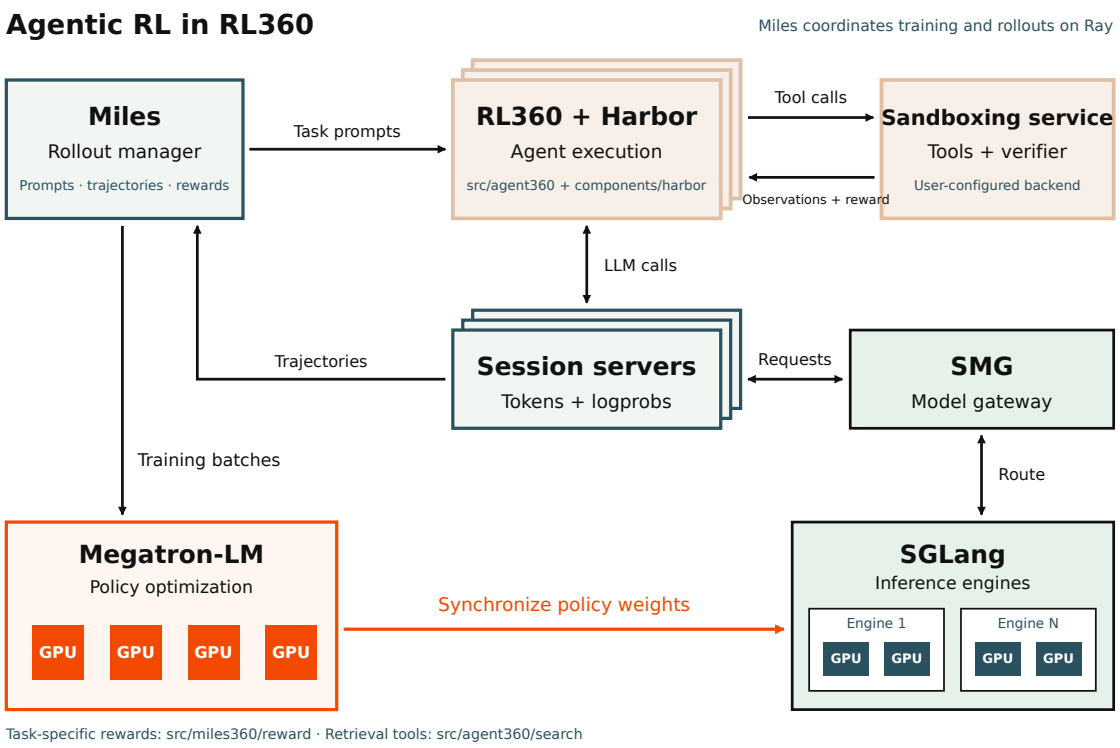


Figure 38: RL360 architecture. The rollout, sandbox, inference, and training layers communicate through explicit service boundaries and can be scaled independently. Miles collects trajectories and rewards, Harbor executes agents in task environments, SMG routes inference requests to SGLang, and Megatron-LM updates the policy.

Miles shards sessions across multiple session-server backends, pinning each session to one backend that owns its turn-ordered state. All backends share one inference-engine pool, so session handling and inference scale independently. Per-backend concurrency limits, timeouts, and retries with backoff confine a slow or failed request to its own trajectory; trajectories that exhaust their retries are dropped.

We move trajectory assembly from the central rollout manager into distributed Ray session actors. Each actor merges a trajectory's turns into a single trainer-ready sample and discards provider and decoding metadata, so the manager receives one compact object per trajectory instead of every turn, and its CPU, memory, and serialization costs no longer grow with the number of in-flight fragments.

To avoid a request burst when a large batch starts, we admit rollouts in waves, bounding peak load on session servers and environment backends without changing the batch size.

Trajectory lifecycle and artifacts. Each trajectory carries its global Miles sample index through Harbor, so artifacts written by different workers (trajectory files, object-store paths, dashboard records) cannot collide. At rollout boundaries, Miles issues a coordinated abort that Harbor's router forwards to every worker, which cancels active trials through Harbor's native cancellation path and tears down their environments. Batch transitions therefore have an explicit lifecycle rather than relying on timeouts or process termination for cleanup.

We keep the archival record separate from the training data path. Token IDs and log-probabilities reach the trainer through Miles, so Harbor's ATIF record retains only what is needed to inspect agent behavior: messages, tool calls, reasoning (stored separately from visible assistant content), and compact usage and cost statistics. Raw provider responses, parser state, and router payloads are dropped, reducing rollout-node

memory, serialization, and storage traffic. Malformed tool calls are recorded with explicit failure metadata rather than silently discarded, so format errors stay visible in analysis.

Failure-aware batch construction. An infrastructure failure is not a policy outcome, and scoring it as zero reward corrupts the learning signal. When other group members succeed, it yields spurious negative advantages; when none do, a solvable prompt looks unsolved. We therefore classify rollout outcomes by exit status before dynamic sampling (Yu et al., 2026). A group enters the batch only if none of its members failed and its rewards are not all identical. We drop the whole group rather than only its failed members because failures correlate with trajectory behavior (e.g., long or tool-heavy episodes), so the surviving members would yield a biased baseline.

Truncation at the token or context limit, by contrast, is an outcome of the policy. A truncated sample's reward still counts toward the group's mean and standard deviation, but its loss is masked, as in overlong filtering (Yu et al., 2026), so the policy is not penalized for exhausting its budget. Malformed tool calls are policy errors rather than failures, so they enter through the reward: Harbor applies a configurable penalty either to every affected trajectory or only to those that otherwise succeed.

Future Design Goals for a Scalable RL Framework. The bottlenecks faced during the RL training of K2-Horizon motivate the following goals for our next-generation RL training system:

1. **Isolate failures.** An inference engine, environment worker, verifier, rollout service, or trainer should be able to fail and recover independently.
2. **Remain distributed end to end.** Eliminate the single rollout-manager bottleneck by distributing collection, trajectory assembly, filtering, and verification across actors.
3. **Schedule work dynamically.** Replace static assignment with cluster-wide scheduling and work redistribution to prevent idle workers and long-tail stragglers.
4. **Balance compute and memory.** Account for sequence length, inference cost, environment cost, verifier latency, and memory pressure—not only task or token counts.
5. **Support very large batches.** Handle rollout batches of at least 10,000 samples through streaming, bounded queues, admission control, and backpressure.
6. **Minimize data movement.** Process data near its producer and transfer only compact, trainer-ready samples or references.
7. **Keep the core minimal and extensible.** Provide stable mechanisms while leaving rollout, scheduling, filtering, reward, and failure-policy decisions to users.
8. **Make behavior observable.** Connect tasks, sessions, inference requests, rewards, failures, and training updates through structured logs and stable identifiers.
9. **Strengthen verification.** Support scalable programmatic and LLM-based task-specific verifiers without placing them on the inference critical path.
10. **Support installed agents directly.** Run existing agent harnesses through their native interfaces and reconstruct token-faithful training trajectories from their model interactions.

Sandboxing. We developed a simple sandboxing system using AWS to support agentic RL, with sandboxes as Kubernetes containers on Amazon Elastic Kubernetes Service (EKS). Requests for the sandboxes are handled by a server deployment on the same EKS cluster. Task images are stored on Amazon Elastic

Container Registry (ECR), using default Kubernetes behavior for caching. Sandbox containers had small nominal memory and vCPU requests, and limits were set to what the RL trainer requested. This enables us to over-pack sandboxes to reduce costs: as long as the sandboxes use bursts of resources at different times, they still get the full resources requested. Egress is handled via an egress proxy deployment on the cluster, with pull-through caches for PyPI, Hugging Face and GitHub.

Key challenges and lessons learned:

- Lazy image loading with Seekable Open Container Initiative (SOCI) was not functional in time. For stability, we disabled SOCI entirely.
- AWS default Container Network Interface (CNI) was a bottleneck for sandbox creation at high churn rates. Sandboxes would be stuck in creation waiting for the CNI to receive new IPs to allocate. This was the primary limiting factor for scaling up the sandbox infrastructure.
- Gains from collocating rollouts that use the same image are workload dependent. For example, for 1-step asynchronous RL with 32 rollouts per task and 4000 total sandboxes, collocating based on images removed all bottlenecks with sandbox creation. For data generation workloads, it pathologically concentrated all of the resource heavy sandboxes onto a few nodes.
- To allow for more GitHub API calls for the entire sandbox cluster, we injected a GitHub application token into the sandboxes. This let the sandboxes use authenticated GitHub API rate limits. However, due to an oversight, this token initially allowed access to our private repositories, allowing agents to find task solutions in our dataset repositories.
- Because of how Kubernetes disk and memory limits are implemented, sandboxes that exceeded their requested amount would be evicted and destroyed. Such infrastructure errors needed to be handled gracefully by the RL trainer.

6.2 RL algorithm

Our RL algorithm is a single-step variant of Group Relative Policy Optimization (GRPO) (Shao et al., 2024) on full multi-turn trajectories. For a prompt q drawn from an expert's task set $\mathcal{D}$, the rollout policy $\pi_{\theta_{\text{rollout}}}$ generates a group of G trajectories, $\{o_i\}_{i=1}^{G}$ — each trajectory o_i is the full token sequence of a multi-turn episode — a combination of tokens generated by the policy (reasoning tokens and tool calls) and tokens returned by the environment (tool outputs, user-simulator replies). Let $\mathcal{T}_i \subseteq \{1, \ldots, |o_i|\}$ denote the positions of policy-generated tokens — we do not train on environment tokens (these are masked during gradient updates). Let A_i denote the group-relative advantage computed as $A_i = \big(r_i - \text{mean}(\{r_1, r_2, \cdots, r_G\})\big) / \text{std}(\{r_1, r_2, \cdots, r_G\})$, where r_i denotes the final score of the verifier for trajectory o_i.

We use the following learning objective:

$$\mathcal{L}(\theta) \;=\; -\,\mathbb{E}_{q\sim\mathcal{D},\,\{o_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{rollout}}}(\cdot\,|\,q)}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|\mathcal{T}_i|}\sum_{t\in\mathcal{T}_i}\rho_{i,t}\,A_i\,\log\pi_{\theta_{\text{trainer}}}\big(o_{i,t}\mid q, o_{i,<t}\big)\right], \tag{15}$$

where $\rho_{i,t}$ is a truncated importance-sampling ratio between the trainer policy and the rollout policy, $\rho_{i,t} = \min\big(\pi_{\theta_{\text{trainer}}}(o_{i,t} \mid q, o_{i,<t}) / \pi_{\theta_{\text{rollout}}}(o_{i,t} \mid q, o_{i,<t}),\, C\big)$, treated as a constant. Groups with $\text{std}(\{r_1, r_2, \cdots, r_G\}) = 0$ are discarded. The loss of each trajectory is averaged over the number of tokens $|\mathcal{T}_i|$ before the trajectories are averaged, ensuring every trajectory contributes equally to the gradient

regardless of its length[23] and preventing a long multi-turn episode from dominating the update over a short one.

Asynchronous Training. We use non-colocated training where rollouts are generated by a separate inference fleet that runs concurrently with the trainer and receives new weights after each update. In the one-step asynchronous training regime, the fleet generates a batch using π_{k+1} while the trainer updates use policy π_k and pauses at a batch boundary, so every trajectory is at most one policy version stale. For agentic tasks having long, multi-turn trajectories with high variance in trajectory lengths, pausing at batch boundaries (as done in the one-step asynchronous training regime) leaves the fleet idle while the longest trajectories finish, making the rollout phase dominate wall-clock time. We train experts on agentic tasks fully asynchronously, where generations are never paused and finished trajectories are batched as they arrive. This keeps the inference fleet saturated and substantially raises end-to-end throughput, at the cost of increasing staleness; $\rho_{i,t}$ absorbs the off-policy gap. We enforce maximum staleness of 5 and trajectories where any token is generated by a policy with larger staleness are dropped from the training batch.

In the fully asynchronous regime, a weight push arrives while trajectory generations are in flight. Rather than discarding these, we use partial rollouts: a push interrupts each in-flight trajectory at the request level, i.e. at the boundary of the current model call, while its environment state (sandbox, tool session, user-simulator context) is preserved. Once the new weights are loaded, the episode resumes under the updated policy. Such updates lead to different parts of a trajectory being generated by different policies. Since $\rho_{i,t}$ is defined per token against the policy that actually generated that token, such mixed-policy trajectories are handled by the same correction, and no environment work is wasted at a weight update.

Training Hyperparameters. Each expert training (described in Section 6.3 below) uses a slightly different setup of hyperparameters for training. The knobs we experiment with are:

- Rollouts: Context length and global batch size[24]. All experiments set temperature 1.0 and top-p as 1.0 for rollouts.
- Optimization: For all experts, we use default Adam optimizer and no KL regularization. Learning rates are tuned for different experts along with weight decay rules. We use gradient-norm clipping throughout and a constant learning rate (optionally experimenting with warm-up schedules).
- Dynamic sampling: Following DAPO (Yu et al., 2026), we use dynamic sampling to filter out groups with all correct and all incorrect trajectories, and use oversampling to maintain batch size across training updates.
- Asynchronous training: we experiment with asynchronous training with staleness one (i.e., $\pi_{\theta_{\text{trainer}}}$ and $\pi_{\theta_{\text{rollout}}}$ differ by one update) for some experts and fully asynchronous training for other experts. We use truncated importance sampling with ratio cap of 2.0 to control for off-policyness.
- Reward shaping and entropy regularization: For agentic expert training, we apply penalties to trajectories with malformed, hallucinated (unknown) and/or repeated tool calls. For some experts (Table 29), we add an entropy bonus to prevent mode collapse.

The primary settings for each expert training are summarized in Table 29.

[23] This is commonly known as the *sequence mean loss*.

[24] global batch size = rollout batch size (number of unique tasks) x number of samples per task.

Hyperparameter	Knowledge Work	Tool Use	Search	Reasoning	Inst. Following
Rollout configuration					
# Rollouts per prompt	32	128	32	16	8
# Prompts per batch	64	16	64	64	128
Global batch size	2,048	2,048	2,048	1,024	1,024
Context length	128k	128k	96k	64k	128k
Reward shaping and entropy bonus					
Tool-call penalty	$-0.3/-0.05$	$-0.3/-0.1$	$-0.3/-0.1$	–	–
Penalty scope	All trajectories	Correct only	Correct only	–	–
Entropy bonus coefficient	5×10^{-4}	–	5×10^{-4}	–	–
Optimization					
Initial learning rate	1×10^{-6}	5×10^{-7}	1×10^{-6}	2×10^{-6}	5×10^{-7}
Peak learning rate	5×10^{-6}	2×10^{-6}	1×10^{-5}	2×10^{-6}	2×10^{-6}
Warmup	10 steps	10 steps	10 steps	–	10 steps
Post-warmup schedule	Constant	Constant	10^{-5} to step 20, then 10^{-6}	Constant	Constant

Table 29: RL training configurations for different experts. A dash indicates that the setting is not specified in the corresponding training run.

6.3 Training Domain Experts for K2-Horizon-375B-A23B

Search. For the training of the search expert, the policy interleaves reasoning with tool calls and is scored by an outcome-based answer-correctness reward. The agent's search tool is backed by a hybrid retrieval stack that unifies live web search (Serper), full-page reading (Jina), and a self-hosted local Wikipedia RAG server (dense retrieval with an encoder over a FAISS index of about 26M Wikipedia passages), exposed to the policy behind a unified tool interface with web calls served through a disk cache. The harness exposes two retrieval tools to the policy: search and fetch. The search tool issues a natural-language query to the retrieval backend and returns up to k candidate documents. Each candidate is shown with a title, URL, and content snippet (k defaults to 5, capped at 10). The fetch tool returns the full (length-truncated) text of a single document given a URL, typically one surfaced by a prior search call. The agent interleaves reasoning with these calls over multiple turns and delivers its final answer through a separate submit action. The data used here includes the multi-hop complex search RL tasks generated by our pipeline described in Section 5.10, plus the open-sourced RL tasks from REDSearcher (Chu et al., 2026) and DeepDive (Lu et al., 2025). The reward is given by an LLM judge that checks whether the submitted answer matches the ground truth. To prevent the policy from overfitting to a fixed surface form, we apply per-trajectory domain randomization of the agent interface: the tool names, parameter names, tool and parameter descriptions, parameter ordering, and the phrasing of the system/user prompt are randomized from a profile pool. The randomization profile is drawn deterministically per instance so that all G trajectories of a prompt share one surface, keeping the group-relative baseline clean.

Tool Use. For the training of the tool-use expert, the policy reasons and issues tool calls against a live domain environment whose state is a real database snapshot. The data covers a wide range of tool use domains, from personal assistant tasks to SaaS workflows. Some of the tasks require multi-turn interaction with an end user. For these tasks, an LLM-based user simulator generates user turns, responding to clarification questions and providing additional information as the interaction progresses. The trajectory receives rewards combining programmatic assertions on the final environment state with a set of checks judged by an LLM grader; the reward is therefore continuous rather than binary.

Knowledge Work. For the training of the knowledge-work expert, the agent operates in a sandboxed workspace, reasons and issues code execution and bash tool calls to build a deliverable, and submits its output files through a finish action. We use 5.6K tasks generated by our pipeline introduced in Section 5.9. The document and table pages are rendered to images and attached alongside the text under a per-task routing policy (up to 24 pages), so the policy conditions on the actual document layout rather than a text transcription alone. Each completed trajectory receives a continuous reward from a dedicated LLM judge that inspects the submitted files against the task's criteria; the reward is graded and continuous rather than binary. To keep the policy from overfitting to one agent scaffold, we randomize the interface surface per GRPO group: the system prompt, the user-prompt style, the tool names, the tool parameter names and schemas, and the tool-result presentation are all randomized.

Reasoning. For RL training of the single-turn reasoning expert, we source our training data primarily from the Guru dataset (Cheng et al., 2026), which is a collection of high quality reasoning tasks in six different domains (math, science, logical reasoning, simulation, tabular reasoning and single turn coding) curated from various open source reasoning datasets. We also curate reasoning tasks in math and physics from arXiv papers as well as source some tasks from third-party vendors to train the model for research capabilities.

To create the training subset, we do a pass@8 analysis with the base checkpoint at 128K context length and filter easy tasks which we define as all tasks with a pass rate greater than 0.8 (so all tasks with 7/8 or 8/8 correct responses from the base checkpoint). We also filter out tasks with quality issues: malformed LaTeX in the problem statement, an incorrect ground-truth answer, or an ambiguous question with several correct answers. From the remaining tasks, we sample the 7.5K hardest, mainly from math, science, and coding.

We use a combination of rule-based rewards (coding, logical reasoning domains and tasks with multiple choice questions) and LLM-as-a-judge fallback (math and science domains) to score generations during GRPO. We use GPT-OSS-120B model as the LLM judge during training. To account for cases where the judge model refuses to score, we mask out the corresponding trajectory from gradient updates.

Instruction Following.
The instruction following expert is trained on 15,000 prompts, evenly split between direct instruction-following and factual recall with abstention tasks.

- **Instruction following**. The prompts are obtained from NVIDIA's Multi-Turn Chat, where the model generates the final assistant response of a multi-turn conversation evaluated against multiple rubric criteria, and NVIDIA's InverseIFEval, which contains English prompts with unusual but verifiable formatting constraints, such as reversed words, vowel substitution, and bracketed tokens.
- **Multi-constraint instruction following**. The prompts are sampled from AllenAI's IF Multi-Constraints dataset, which contains single-turn prompts with one to five verifiable IFEval-style constraints. These include requirements on keywords, formatting, length, casing, punctuation, first or last words, and repetition.
- **Factual recall with abstention**. The prompts are evenly divided between answerable questions and questions that should be abstained. The data includes NVIDIA's QA Abstention dataset, covering health, software engineering, and law, after screening with an LLM-based validity classifier. The remaining prompts are sampled from our abstention data pipeline.

All rewards are bounded between 0 and 1, with the grading method selected according to the prompt's data source:

- **Constraint Checkers.** The multi-constraint instruction-following responses are evaluated using deterministic IFEval/IFBench constraint checkers, where the final response excluding the reasoning content is evaluated. The reward is a continuous value proportionate to the number of constraints satisfied, allowing partial credit when only a subset of the required constraints is met. Each constraint is evaluated using the loose IFEval rule, which checks both the original response and relaxed variants that remove Markdown formatting or boundary lines.
- **LLM Judge.** For instruction-following prompts, each rubric criterion is judged independently, and the reward is the proportion of criteria for which the judge's decision matches the expected verdict. The Multi-Turn Chat evaluation provides the conversation history, model response, and rubric question, while InverseIFEval provides its corresponding grading prompt and model response. For abstention prompts, the final answer is evaluated against the target. For answerable questions, a correct answer receives a reward of 1, an incorrect answer receives 0, and abstention receives 0.25. For questions with a target [IDK], abstention receives 1, while any attempt to answer receives 0. Explicit [IDK] responses are scored directly without a judge call when applicable.

6.3.1 Model evolution during RL

In this section, we introduce the behavioral changes we observe on the agentic experts during RL (i.e., task reward, tool-call hygiene, trajectory length, and pass@k), whose per-step curves are shown in Figures 39, 40 and 41. We report reward as the all-shard, unfiltered base reward, i.e., the task score before shaping penalties are subtracted, and we define the invalid tool call rate as the fraction of trajectories containing at least one malformed call (one that cannot be parsed) or one hallucinated call (one naming a tool that does not exist); repeated calls carry a separate penalty and do not enter this rate. The knowledge-work expert improves the most, going 69.3% $\rightarrow$ 80.3% in base reward (Figure 39); its penalty-subtracted reward goes 58.9% $\rightarrow$ 80.2% over the same window, so the two curves end together and about half of the apparent gain in the optimized objective is penalty avoidance rather than solving more tasks. Its hygiene is fixed early and almost completely, starting at a 32.2% invalid rate and 0.56 malformed calls per trajectory and reaching 0.10% and 0.001 almost entirely within the first 30 steps, and its trajectories grow 27.1k $\rightarrow$ 35.9k tokens while the turn count falls from 20.0 to 14.1, converging on fewer and longer turns.

The tool-use expert moves less, 47.3% $\rightarrow$ 53.0% over 100 steps (Figure 40), and has essentially no hygiene left to learn, issuing 81 malformed calls in 4.74 M tool calls over the whole run, since its tool interface is schema-constrained and the base model already emits it correctly; its trajectories grow 10.7k $\rightarrow$ 18.3k tokens and 13.4 $\rightarrow$ 19.3 turns.

The search expert goes 55.4% $\rightarrow$ 61.8% and is flat after step 20 (Figure 41), while its hygiene improves on the same short chain as the knowledge-work expert (28.6% $\rightarrow$ 0.73% invalid, malformed 0.262 $\rightarrow$ 0.004, hallucinated 0.432 $\rightarrow$ 0.006), essentially all of it before step 20. Its trajectories grow 53.7k $\rightarrow$ 62.0k tokens while it generates only 9.1k $\rightarrow$ 10.1k of them, so that about 84% of a search trajectory is tool output and its growth is retrieved context rather than reasoning. This expert is in fact context-bound: its longest trajectory sits at 97.6k tokens at every rollout step (97,608–97,712 across all 40) against a 96k window and truncation rises 14.3% $\rightarrow$ 22.4%, so the exported checkpoint sits on a plateau rather than on a rise.

The mean reward alone does not say whether RL is concentrating abilities the model already has into its first sample or extending what it can solve at all, so we also track pass@k over the 32 samples each prompt receives (Figure 42). For the knowledge-work expert, pass@1 rises 73.3% $\rightarrow$ 89.3% while pass@32 rises 90.7% $\rightarrow$ 98.1%, halving the gap between them from 17.4 to 8.8 points, so concentration accounts for most of the gain; the frontier moves as well, since pass@32 is one minus the fraction of prompts no sample solves, and that fraction falls from 9.3% to 1.9%. Two measurement caveats apply throughout. Overlong trajectories are excluded from the reward mean and are assumed to score zero, an assumption we

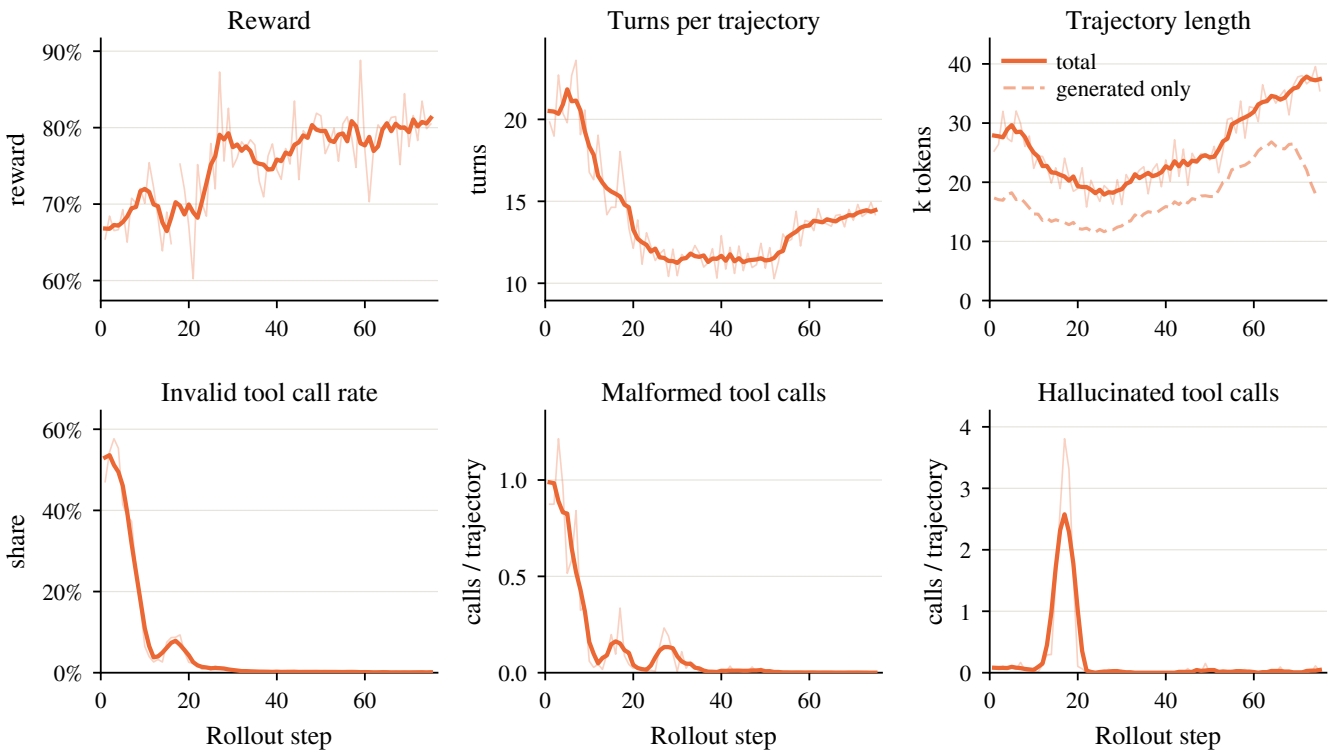


Figure 39: Knowledge Work expert model evolution during RL. Thin lines are raw per-step values, heavy lines a 5-step sample-weighted rolling mean; each curve ends at the exported checkpoint. Trajectory length counts every token after the initial prompt, with the generated part shown dashed.

cannot verify from these runs; the two conventions differ materially only for search, where scoring them zero instead gives a flat 47.6% → 47.9%. All quantities are per-rollout-step aggregates and are exposed to composition bias, since long trajectories finish later and land in later weight versions. Therefore, the length trends describe what the batches contained rather than an established per-task change.

6.4 Model Merging with ISO-RAM

We use model merging, rather than on-policy distillation, to consolidate RL specialists into a single policy. Attention and shared-expert matrices are merged with ISO-merger (Zhu et al., 2026a), which combines the specialists' changes in singular-vector orientations while keeping the base model's singular-value spectra fixed. All remaining parameters, including routed experts, the router, and normalization weights, are merged with RAM (Yuan et al., 2026), which averages updates made by several specialists but preserves updates unique to a single specialist at full strength, so that sparse, specialist-specific changes are not diluted by averaging.

ISO-merger (Zhu et al., 2026a) merges specialists in the singular-vector coordinates of each weight matrix rather than in raw weight space. Write the base weight as $W_0 = U_0 \Sigma_0 V_0^\top$ and specialist i as $W_i = U_i \Sigma_i V_i^\top$. (i) Fixed spectrum. The merged matrix keeps the base singular values Σ_0; each specialist contributes only its change in singular vectors, $\Delta U_i = U_i - U_0$ and $\Delta V_i = V_i - V_0$, projected onto the tangent space at the base and truncated to the leading singular modes. (ii) Overlap-aware coefficients. Specialists may move the same directions, so a plain sum would over-count shared changes. ISO-merger assigns each specialist a coefficient c_i per matrix, chosen so that, to first order, each specialist's own weight change is retained at unit strength in the merge; this reduces to a small $K \times K$ linear system over inner products of the specialists' weight changes. (iii) Reconstruction. The combined changes are added to the base singular

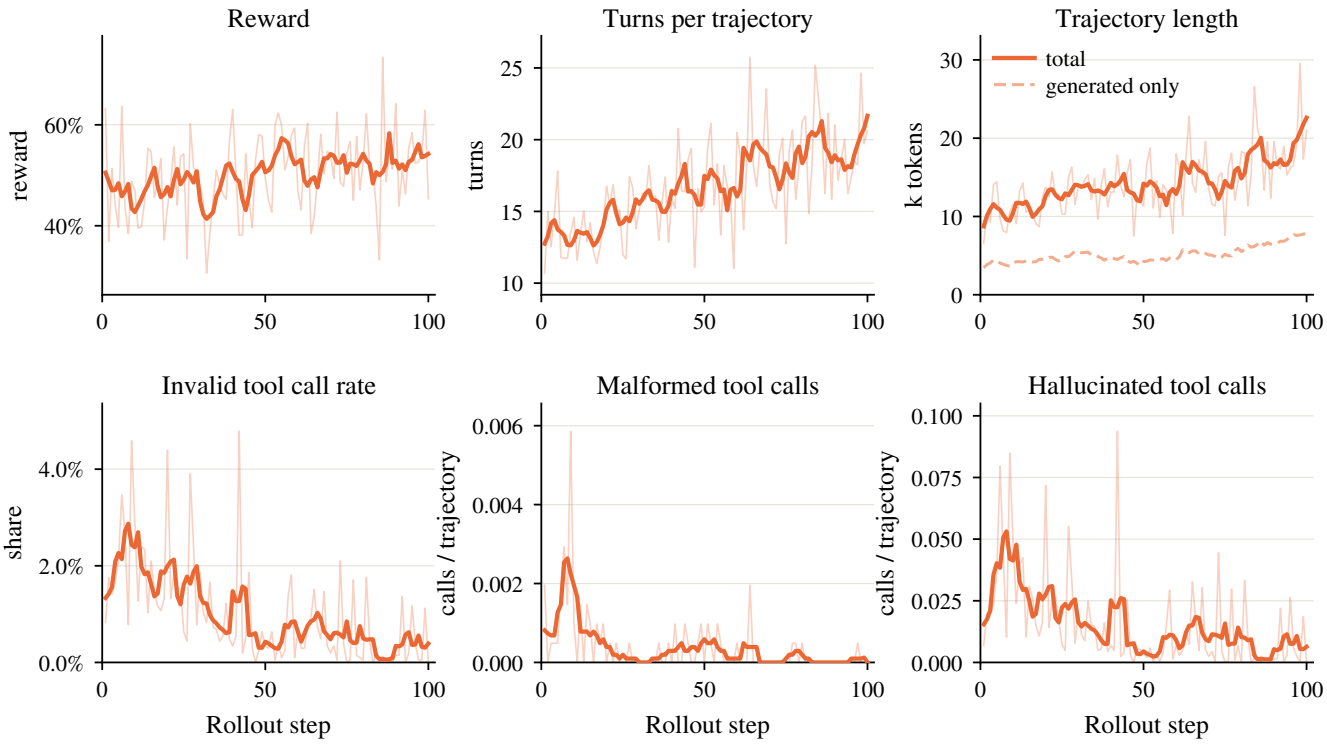


Figure 40: Tool Use expert model evolution during RL.

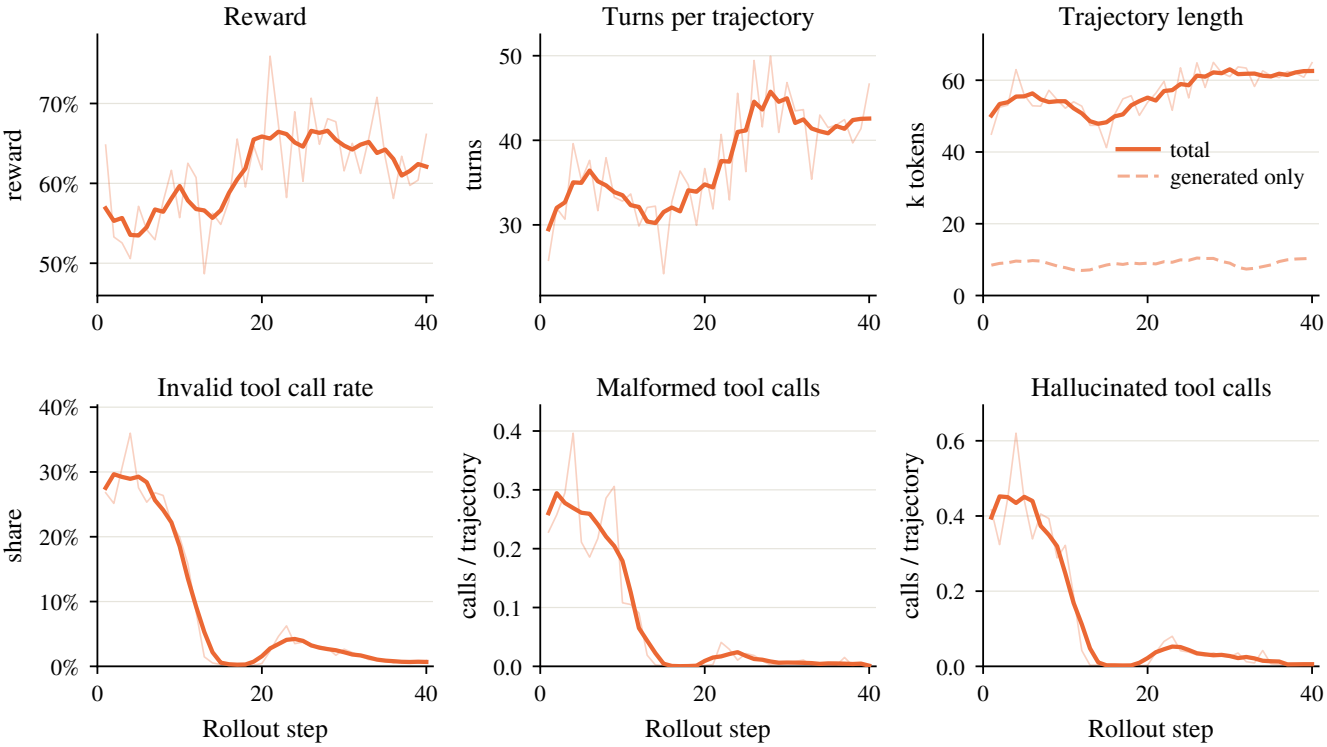


Figure 41: Search expert model evolution during RL.

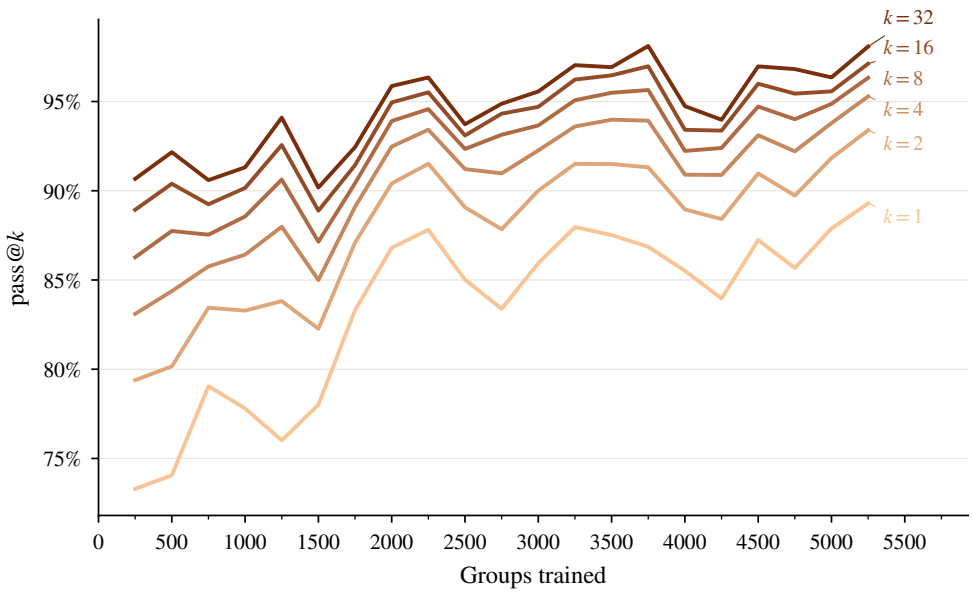


Figure 42: Knowledge Work expert pass@k during RL, in non-overlapping bins of 250 prompt groups. pass@k is the unbiased estimator $1 - \binom{n-c}{k} / \binom{n}{k}$ over the $n = 32$ samples each prompt receives, counting a sample as correct when its base reward exceeds 0.5.

	BrowseComp Acc ↑	τ^3-Banking Pass@1 ↑	GDPval Acc@75% ↑	HLE Pass@1 ↑	IFBench Acc ↑	Omniscience Acc / Non-Halluc. ↑	Terminal-Bench 2.1 Avg@3 ↑
Base model	59.2	11.0	45.15	15.9	53.0	23.5 / 15.1	37.8
RL expert	62.5	24.0	58.73	20.5	60.7	23.8 / 20.8	-
Merged model	68.1	19.5	59.32	20.0	65.2	24.5 / 34.9	46.1

Table 30: Capability retention and transfer after merging RL specialists. The first six benchmarks evaluate capabilities targeted by the corresponding RL experts, with IFBench (strict prompt-level) and Omniscience jointly covering instruction following, factual recall, and abstention. Terminal-Bench 2.1 evaluates an additional capability without a dedicated expert. Metrics and evaluation settings follow those defined in the evaluation section.

vectors and retracted to the nearest orthonormal matrices,

$$U_\star = \mathrm{polar}\Big(U_0 + \textstyle\sum_i c_i \Delta U_i\Big), \quad V_\star = \mathrm{polar}\Big(V_0 + \textstyle\sum_i c_i \Delta V_i\Big), \quad W_\star = U_\star \Sigma_0 V_\star^\top. \tag{16}$$

Remaining details, such as sign alignment and coefficient regularization, follow Zhu et al. (2026a).

RAM (Yuan et al., 2026) merges the remaining parameters element-wise, based on which specialists actually changed each entry. Write specialist i's update as $\tau_i = W_i - W_0$. (i) Update detection. An entry j counts as updated by specialist i if $|\tau_{i,j}| > \epsilon$; let A_j be the set of specialists that update it. (ii) Selective merging. Averaging every entry over all K specialists would scale an update made by only one specialist by $1/K$, diluting specialist-specific behavior. RAM instead averages only over the specialists that update an entry, keeps single-specialist updates at full magnitude, and leaves entries no specialist touched at the base value. (iii) Reconstruction. The merged weight is

$$W_{\star,j} = W_{0,j} + \begin{cases} 0, & |A_j| = 0, \\ \frac{1}{|A_j|} \sum_{i \in A_j} \tau_{i,j}, & |A_j| \geq 1. \end{cases} \tag{17}$$

We use $\epsilon = 10^{-5}$ with updates computed in FP32, uniform specialist weights, and no additional rescaling of unique updates.

We apply ISO-merger only to attention and shared-expert matrices: they process every token, so all specialists update them and their changes overlap, which is the setting overlap-aware merging targets. Routed experts, by contrast, are updated only by the tokens routed to them. These, together with the router and normalization weights, are merged with RAM. Extending ISO-merger to routed experts gave no measurable gain in our evaluations.

Empirical compatibility. ISO-merger rests on two premises about how RL changes a weight matrix, and we checked both on our checkpoints. The first is that the singular values can be held at the base's without losing much of the update. Across all attention and shared-expert matrices, rebuilding each specialist from its own singular vectors and the base singular values misses a median of only 1.4% of its update $W_i - W_0$ in Frobenius norm. The second premise is that singular vectors stay close enough to the base's for the specialists' changes to be combined in a single tangent space at the base. The observed angular changes support this premise: depending on the specialist, the median angle between matched singular vectors ranges from 1.6° to 3.8°, and the 90th percentile ranges from 3.6° to 8.8°. The evaluations below measure how much of each specialist's capability the merged model retains.

Capability integration. As shown in Table 30, the merged policy largely preserves the specialists' gains across targeted capabilities, with only a partial reduction on τ^3-Banking that remains well above the base model. Beyond retention, the merged policy further improves over the corresponding specialists on BrowseComp, GDPval, IFBench, and Omniscience, with the largest gain coming from reduced hallucination. It also improves Terminal-Bench 2.1 despite having no dedicated specialist for this task, suggesting that merging can enable positive transfer beyond the directly optimized capabilities. We hope these results encourage further exploration of model merging for capability integration in large-scale MoE models.

6.5 Post-Training Small Models

The 0.9B, 3.7B, and 7B models broadly follow the same post-training approach as the larger models, combining domain-specific RL with capability integration. A difference is the initialization of coding experts: we found that starting coding RL from a math-trained checkpoint worked better than starting directly from the base model, and adopted this strategy for the small models.

For 0.9B, transferring the math RL recipe to a later mid-training checkpoint led to entropy collapse and performance degradation. We addressed this issue by alternating math and coding RL in a Math–Code–Math–Code sequence, allowing math performance to improve again after the coding stage.

Small models are also prone to degenerate repetition on long generations, so for the 3.7B and 7B models we add a short anti-doom-loop stage, using Final Token Preference Optimization (FTPO) on preference pairs mined from the model's own looping outputs. This almost entirely removes runaway generations while keeping standard benchmarks flat, with the largest gains on agentic tasks such as Terminal-Bench 2.1, where a single runaway call can stall an episode.

6.6 Supervised fine-tuning (SFT)

Following the RL stage, we conducted extensive evaluations to understand gaps in model performance, and continued with a round of SFT targeted at closing the identified gaps. Evaluation results are shown in §7.

6.6.1 Optimizer Recipe

We did not do SFT for the 0.9B model. For all other models, we continue to use the same recipe from the last mid-training stage (stage 4) of each model (see detailed recipes in Table 3). We keep the setup mostly

the same and only changed the training steps (tied with tokens) and the learning rate decay schedules depending on the model sizes. For the 375B model, we trained on 280B tokens with constant learning rate scheduler followed by 50B tokens with linear learning rate decay to 10% of its peak learning rate. For 36B-A4B and 32B models, we trained on 220B tokens with constant learning rate scheduler followed by 50B tokens with linear learning rate decay to 10% of its peak learning rate. For the 7B and 3.7B models, we trained on 200B tokens at a constant learning rate, followed by 50B tokens with cosine decay to 10% of the peak (steps 2,000 to 2,500 of SFT).

6.6.2 Infrastructure

Same as the infrastructure used for pre-training and mid-training (§3.2). The compute and parallelism definition is also the same as the last mid-training stage (stage 4) of each model.

6.6.3 Training Run

We apply the SFT stage on top of the merged RL model for 375B, 7B and 3.7B models. For 36B-A4B and 32B models, we apply the SFT stage on the final checkpoint of mid-training as we do not have RL stage for these two models. For 0.9B model, we did not apply SFT.

6.6.4 Data

SFT data was generated with the subset of pipelines described in Section 5. For this data, we primarily used DeepSeek-V4-Flash (DeepSeek-AI, 2026) and Kimi K3 (Kimi Team, 2026b) as task synthesis and trajectory generator models, incorporating generation methods from Section 5.16) where relevant. During the constant learning rate phase we used all of the trajectories without correctness filtering. The subset of correct trajectories was replayed during the decay phase.

7 Post-Training Evaluation

This section evaluates the six post-trained `K2-Horizon` models on agentic, coding, scientific-reasoning, and general benchmarks. §7.1 describes the evaluation setup. Sections 7.2 through 7.4 compare each model against baselines of comparable size. To understand the effect of post-training on agentic coding capabilities, §7.5 traces how 375B-A23B's behavior on Terminal-Bench 2.1 evolves across training checkpoints, and §7.6 audits its passing trajectories for reward hacking to verify that the solutions actually solve the task rather than game the evaluation.

7.1 Evaluation Setup

Benchmark Suite. We evaluate our models across four benchmark categories:

- **Agentic**: Real-world professional tasks requiring tool use, workflow automation, and web search, including GDPval-AA (Patwardhan et al., 2025), τ^3-Banking (Shi et al., 2026), Toolathlon-Verified (Li et al., 2026a), the public evaluation set of AutomationBench (Shepard and Salimans, 2026), Apex-Agents (Vidgen et al., 2026), MCPMark (Wu et al., 2026), BrowseComp (Wei et al., 2025), WildClawBench (Ding et al., 2026), and BFCL v4 (Patil et al., 2025).
- **Coding:** Software engineering, repo-level code understanding, and scientific coding tasks in agentic settings, including Terminal-Bench 2.1 (Merrill et al., 2026), SciCode (Tian et al., 2024), SWE-Atlas-QnA (Raghavendra et al., 2026) and SWE-bench Pro (Deng et al., 2025) (both in strict mode, without internet access), SWE-bench Verified (Jimenez et al., 2024), LiveCodeBench v6 (Jain et al., 2024), HumanEval+, and MBPP+ (Liu et al., 2023).
- **Scientific reasoning:** Expert-level reasoning in science, spanning graduate-level question answering to frontier physics problems, including Humanity's Last Exam (HLE; Center for AI Safety et al., 2026), GPQA Diamond (Rein et al., 2024), and CritPt (Zhu et al., 2026b).

- **General:** Long-context reasoning, factual accuracy, non-hallucination, and competition mathematics, including AA-LCR, AA-Omniscience (Jackson et al., 2025), AIME 2026, and HMMT February 2026 (Balunović et al., 2026).

Evaluation Settings. All `K2-Horizon` models use high reasoning effort with temperature 1.0 and top-p 0.95; the 0.9B model uses temperature 0.6. For benchmarks in the Artificial Analysis (AA) Intelligence Index (i.e., GDPval-AA, τ^3-Banking, Terminal-Bench 2.1, SciCode, HLE, GPQA Diamond, CritPt, AA-LCR, and AA-Omniscience), all models are evaluated under the AA Intelligence Index v4.1.1 setup.

For benchmarks outside the AA Intelligence Index, we describe the evaluation setup below. Toolathlon-Verified runs the benchmark's official scaffold and evaluators on the full 108-task split, with a 512K-token context window and at most 100 steps per task; we report pass@1 averaged over three runs. For AutomationBench, we use the v1.0.6 public set. Apex-Agents employs a loop agent on 420 classifier-selected text-only tasks, scored by a Gemini 2.5 Flash judge (pass@1, one run). MCPMark covers the 129-task MCPMark-Verified set with context compaction at 128K tokens (pass@1, one run). For BrowseComp, our model uses the Discard-all@95k context-length protocol proposed for DeepSeek-V3.2 (DeepSeek-AI, 2025). Comparison models for BrowseComp use harnesses specified in their official technical report or release blog. WildClawBench uses the OpenClaw harness on 29 English text-only tasks, scored by a GPT-5.4 judge (mean score, three runs). SWE-bench Pro and SWE-Atlas-QnA both run in strict mode: the agent can reach only the inference endpoint, while environment setup and the verifier retain public internet access.

Unless otherwise noted, all benchmarks for the 7B and 3.7B models are sampled at temperature 1.0 and top-p 0.95. SWE-bench Verified uses Harbor's mini-SWE-agent scaffold on the canonical 500-task set with a 256K-token context window, no step limit, and reasoning effort set to high; network egress is blocked during the agent phase and restored only for verification, and trials that exhaust the context or output budget or time out are counted as unresolved. Terminal-Bench 2.1 uses Terminus 2 with 3 repeats, a 256K context, 250 turns, and a 7200-second agent timeout in a local sandbox. LiveCodeBench v6 uses the full 1055-problem set with 4 samples per problem and a 500K-token output budget, graded by unit tests in Sandbox Fusion; we report pass@1 averaged over the 4 samples. AIME 2025 and AIME 2026 use 32 samples per problem with a 500K-token budget and math-verify grading; HMMT February 2025 uses 16 samples per problem with a 500K-token budget, and HMMT February 2026 uses the 33-problem MathArena set with 16 samples per problem and a 256K-token budget, both graded with the PRIME verifier. BFCL v4 runs the official `bfcl_eval` client with the official leaderboard weighting, temperature 0.01, 8192 max tokens per turn, reasoning effort high, and the XML tool format, with `web_search` served through a Serper backend. GPQA Diamond uses the AA multiple-choice prompt with 5 repeats and a 256K-token budget; HLE uses the 2158 text-only questions with one sample and a 256K-token budget, with exact-answer items judged by GPT-5.5; SciCode uses 3 repeats; AA-LCR and AA-Omniscience (600 public questions) use one sample judged by GLM-5.2, with a 32K-token output budget for AA-LCR; and IFBench (Pyatkin et al., 2025) uses the 300 prompts with one sample, a 500K-token budget, and strict prompt accuracy under the official verifiers. HumanEval+ and MBPP+ were not run for these models.

7.2 375B-A23B Results

	Open-weight					Closed		
Benchmark	**K2-Horizon 375B-A23B**	Nemotron 3 Ultra	Inkling (xhigh)	MiniMax M3	GLM-5.2 (max)	GPT-5.6 Luna (max)	GPT-5.6 Terra (high)	Claude Sonnet 5 (max)
# Total params	375B	550B	975B	428B	753B	–	–	–
# Activated params	23B	55B	41B	23B	40B	–	–	–
Agentic								
GDPval-AA (Elo)	1,441	1,162	1,234	1,380	**1,498**	1,569	1,503	1,584
τ^3-Banking	34.0	14.2	29.1	15.3	**34.6**	31.1	28.7	37.3
Toolathlon-Verified	**65.3**	34.3	45.5	53.7	59.9	67.5	64.8	71.6
AutomationBench Public	25.3	8.0	12.8	20.5	**26.2**	33.5	28.0	34.7
Apex-Agents (pass@1)	24.8	9.0	19.0	23.8	**26.9**	28.6	25.4	31.7
MCPMark	67.7	45.7	51.2	48.8	**72.4**	66.9	74.0	65.3
BrowseComp	72.8	44.4	77.1	**83.5**	–	83.3	–	84.7
WildClawBench	50.9	34.2	52.3	**56.4**	55.0	50.4	60.0	–
Coding								
Terminal-Bench 2.1	70.2	53.9	55.1	65.2	**77.9**	80.9	75.7	80.5
SciCode	42.7	39.9	46.1	45.4	**50.5**	52.5	50.1	53.6
SWE-Atlas-QnA (strict)	**48.4**	–	25.5	42.3	46.4	–	–	–
SWE-bench Pro (strict)	42.6	38.7	43.1	43.8	**46.7**	48.8	–	–
Scientific Reasoning								
HLE (without tools)	32.0	28.4	31.9	39.0	**41.1**	39.5	38.5	41.3
GPQA Diamond	87.3	86.7	87.2	**92.9**	89.5	91.1	89.6	91.1
CritPt	8.6	3.1	5.4	3.7	**20.9**	20.6	22.9	16.9
General								
AA-LCR	76.0	71.0	73.3	**80.3**	76.7	78.3	73.3	77.0
AA-Omniscience								
Accuracy	23.0	22.6	**41.6**	16.7	24.3	42.7	45.5	40.1
Non-Hallucination	74.7	70.3	32.3	**81.6**	73.7	7.4	10.2	60.6

Table 31: `K2-Horizon`-375B-A23B against open-weight and closed frontier models. We highlight the best result among open-weight models per row in **bold**, and underline the second-best one; strict = no internet access.

Table 31 compares 375B-A23B against four open-weight models and three closed frontier models. 375B-A23B activates only 23B parameters, the same as MiniMax-M3 and far fewer than the 40–55B active parameters in the other three open-weight baselines, yet it still leads among open-weight models on two benchmarks and places second on eight more.

On agentic tasks involving tool use, environment interaction, and multi-step execution, 375B-A23B is competitive with closed frontier models while activating only 23B parameters. It leads among open-weight models on Toolathlon-Verified, approaching GPT-5.6 Luna and Claude Sonnet 5 (both at max effort) while exceeding GPT-5.6 Terra (at high effort), and is second to GLM-5.2 on five other benchmarks (τ^3-Banking, MCPMark, AutomationBench Public, Apex-Agents, and GDPval-AA). Notably, it exceeds both GPT-5.6 variants on τ^3-Banking and is above GPT-5.6 Luna and Claude Sonnet 5 on MCPMark. BrowseComp (web browsing) and WildClawBench (open-ended real-world agent tasks) leave room for improvement, though 375B-A23B is not far behind the leading open-weight models on either and exceeds GPT-5.6 Luna on WildClawBench.

375B-A23B approaches closed frontier models on agentic tasks

By the time of the release, 375B-A23B is the strongest or second-strongest open-weight model on most agentic and coding benchmarks, despite activating fewer parameters than every baseline except MiniMax-M3. It leads on tool-use workflows (Toolathlon-Verified) and repository-level code understanding without internet access (SWE-Atlas-QnA), and places second on eight more, including terminal-based software engineering (Terminal-Bench 2.1). Notably, it exceeds GPT-5.6 Terra (high) on Toolathlon-Verified and both GPT-5.6 variants on τ^3-Banking. Expert-level scientific reasoning (HLE and CritPt) remains the main gap.

On coding benchmarks, 375B-A23B leads among open-weight models on SWE-Atlas-QnA in strict mode (where the agent has no internet access) and is second to GLM-5.2 on Terminal-Bench 2.1, demonstrating effective software engineering capability even under restricted settings. SWE-bench Pro is similarly competitive, with 375B-A23B closely matching Inkling and MiniMax-M3. SciCode leaves more room for improvement on scientific research coding, with a wider gap to the top open-weight models.

On scientific reasoning, 375B-A23B exceeds Nemotron 3 Ultra and Inkling on GPQA Diamond and closely matches GLM-5.2, indicating broad coverage of graduate-level science knowledge. CritPt presents a wider gap: GLM-5.2 is far ahead of all other open-weight models, and the closed frontier models also score substantially higher, pointing to room for improvement in advanced scientific problem solving. HLE follows a similar pattern, with 375B-A23B on par with Inkling but behind MiniMax-M3 and GLM-5.2, suggesting room to strengthen expert-level reasoning across diverse domains.

On the general benchmarks, 375B-A23B stands out on AA-Omniscience Non-Hallucination, placing second among open-weight models behind MiniMax-M3 and surpassing all three closed models by a wide margin, reflecting strong factual grounding. 375B-A23B is also strong on AA-LCR, exceeding GPT-5.6 Terra and approaching Claude Sonnet 5. AA-Omniscience Accuracy leaves more room for improvement, with Inkling far ahead among open-weight models and all three closed models scoring higher.

7.3 Mid-size Results (36B-A4B and 32B)

Tables 32 and 33 compare the two mid-sized models against open-weight models of similar size, with GPT-5.6 Luna (medium) as a closed-model reference. 36B-A4B activates only 4B parameters, yet performs on par with or even exceeds dense models activating eight times as many parameters. Among MoE models, it leads on τ^3-Banking and Terminal-Bench 2.1, is second only to Nemotron 3 Ultra on SciCode and HLE, and places third on the remaining five benchmarks. Nemotron 3 Ultra is about 15 times larger in total parameters (550B vs. 36B) and activates about 14 times as many per token (55B vs. 4B). Despite activating about one-seventh of the parameters of the dense 32B, 36B-A4B scores within only a few points of it on every benchmark.

On agentic and coding tasks, both models are strong for their sizes. 36B-A4B leads among MoE models on τ^3-Banking and Terminal-Bench 2.1, ahead of Nemotron 3 Ultra with 55B activated parameters, demonstrating effective tool use and software engineering capability at this scale. Notably, both 36B-A4B and 32B exceed GPT-5.6 Luna on these two benchmarks. SciCode leaves room to improve on scientific research coding, suggesting it remains a harder capability to elicit at this model size.

On scientific reasoning, both models are close to GPT-5.6 Luna on HLE and competitive on GPQA Diamond, indicating solid coverage of graduate-level to expert-level science knowledge. CritPt remains challenging for all models, including GPT-5.6 Luna, indicating that frontier physics reasoning is not yet within reach at this scale.

	Open-weight							Closed
		MoE				*Dense*		
Benchmark	**K2-Horizon 36B-A4B**	Nemotron 3 Ultra	Nemotron 3 Super	G9v3 39A5B	Qwen3.6 35B-A3B	Muse Glimmer 30B (high)	Gemma 4 31B-it	GPT-5.6 Luna (medium)
# Total params	36B	550B	120B	39B	35B	30B	31B	–
# Activated params	4B	55B	12B	5B	3B	30B	31B	–
Agentic								
τ^3-Banking	**26.8**	14.2	10.3	<u>22.1</u>	9.3	23.5	14.8	17.7
Coding								
Terminal-Bench 2.1	**58.6**	<u>53.9</u>	38.6	32.6	44.9	51.7	43.4	53.2
SciCode	<u>38.9</u>	**39.9**	36.0	34.0	35.8	43.6	43.4	45.8
Scientific Reasoning								
HLE (without tools)	<u>25.2</u>	**28.4**	20.8	17.5	22.2	22.0	23.6	25.8
GPQA Diamond	80.8	**86.7**	80.0	80.5	<u>84.1</u>	83.5	85.7	85.9
CritPt	2.1	**3.1**	**3.1**	0.3	0.3	2.6	1.4	4.9
General								
AA-LCR	66.3	**71.0**	60.3	62.0	<u>66.7</u>	80.0	68.3	72.0
AA-Omniscience								
Accuracy	18.8	<u>22.6</u>	**24.3**	14.9	18.8	27.0	20.0	40.7
Non-Hallucination	69.2	<u>70.3</u>	13.0	**87.0**	49.5	18.1	15.0	9.1

Table 32: `K2-Horizon`-MoVA-36B-A4B against open-weight MoE and dense models, with GPT-5.6 Luna (medium) as a closed-model reference. We highlight the best result among MoE models per row in **bold**, and <u>underline</u> the second-best one.

On the general benchmarks, both models stand out on AA-Omniscience Non-Hallucination, far above GPT-5.6 Luna, reflecting low hallucination rates. AA-LCR and AA-Omniscience Accuracy leave room for improvement, with GPT-5.6 Luna ahead on both, pointing to a persistent gap between open-weight and closed frontier models on long-context reasoning and factual accuracy at this scale.

7.4 Small-size Results (7B, 3.7B, and 0.9B)

Tables 34, 35, and 36 compare the three small models against open-weight models of similar size. 7B leads on most benchmarks. 3.7B and 0.9B follow a similar pattern, with 0.9B holding up against models twice its size.

Both 7B and 3.7B are particularly strong on agentic and coding tasks. On SWE-bench Verified, both lead by wide margins, suggesting that agentic software engineering capability transfers well even to small models in the `K2-Horizon` family. They also lead on τ^3-Banking, again by large margins, indicating robust tool use at this scale. Both models also lead on Terminal-Bench 2.1 and BFCL v4, and place second on LiveCodeBench v6.

On scientific reasoning, both 7B and 3.7B lead on HLE, demonstrating competitive expert-level reasoning for their size. Both 7B and 3.7B also place second on GPQA Diamond, behind the Qwen3.5 models in their comparison groups.

On the general benchmarks, both models are strong on mathematics, leading on HMMT February 2026. 3.7B also leads on AIME 2026, while 7B is a close second. 7B also leads on AA-LCR, indicating solid long-context reasoning.

0.9B leads on all three coding benchmarks (HumanEval+, MBPP+, and LiveCodeBench v6) and both math

Benchmark	Open-weight				Closed
	K2-Horizon 32B	Qwen3.8 27B (xhigh)	Muse Glimmer 30B (high)	Granite 4.2 30B	GPT-5.6 Luna (medium)
Agentic					
τ^3-Banking	24.1	**48.0**	23.5	14.4	17.7
Coding					
Terminal-Bench 2.1	62.6	**79.8**	51.7	26.6	53.2
SciCode	37.9	**44.7**	43.6	36.6	45.8
Scientific Reasoning					
HLE (without tools)	25.2	**33.9**	22.0	11.2	25.8
GPQA Diamond	82.8	**90.5**	83.5	64.4	85.9
CritPt	2.9	**5.4**	2.6	0.3	4.9
General					
AA-LCR	68.0	77.3	**80.0**	46.7	72.0
AA-Omniscience					
Accuracy	18.2	15.6	**27.0**	10.1	40.7
Non-Hallucination	69.7	69.7	18.1	**74.4**	9.1

Table 33: `K2-Horizon-32B` against open-weight dense models, with GPT-5.6 Luna (medium) as a closed-model reference. We highlight the best open-weight result per row in **bold**, and underline the second-best one.

benchmarks (AIME 2026 and HMMT February 2026), with the math leads holding against Qwen3.5-2B at twice its size. On GPQA Diamond and BFCL v4, the 0.9B model trails behind Qwen3.5-2B, suggesting that function calling and graduate-level science knowledge remain harder to elicit at sub-1B scale.

7.5 Agentic Behavior across Training Stages

Benchmark scores show how well the released model performs, but not what changed during training to get there. This matters because the 375B-A23B model's pipeline runs a final SFT stage after RL and model merging, and understanding whether this stage addresses failure modes that RL does not is important for validating the pipeline design. To understand which behaviors change, at what stage, and whether the training data explain the change, we examine 375B-A23B's trajectories on Terminal-Bench 2.1, which uses the `Terminus-2` harness to give the agent a live terminal session over dozens of turns. Unlike harnesses such as `mini-SWE-agent` that execute each command as a stateless subprocess call, `Terminus-2` exposes a persistent terminal session where commands can hang, output can stall, and the agent must decide how to recover from there. Notably, RL does not train on Terminal-Bench-related tasks, so any gains it brings reflect general capability transfer rather than task-specific optimization. The pass rate nearly doubles from mid-training to the released checkpoint (37.8% to 70.2%), yet not every behavior improves at the same stage (Figure 43). RL and SFT each resolve distinct failure modes, as we detail below.

Benchmark	K2-Horizon 7B	Qwen3.5 9B	Gemma 4 12B	Granite 4.2 8B
Agentic				
τ^3-Banking	**25.8**	7.0	–	7.6
BrowseComp	**59.0**	–	–	–
BFCL v4	**67.0**	65.7	60.5	48.4
Coding				
SWE-bench Verified	**72.4**	50.8	30.6	47.7
Terminal-Bench 2.1	**44.6**	29.2	27.3	18.4
SciCode	33.6	27.5	**38.2**	30.4
LiveCodeBench v6	72.7	65.6	72.0	**73.2**
Scientific Reasoning				
HLE (without tools)	**19.5**	14.9	15.7	9.7
GPQA Diamond	75.6	**80.6**	75.3	63.1
General				
AA-LCR	**68.0**	65.3	61.7	43.3
AIME 2026	90.2	88.0	79.6	**90.6**
HMMT Feb 2026	**77.8**	65.7	63.1	66.5

Table 34: `K2-Horizon-7B` against open-weight models of similar size. We highlight the best result per row in **bold**, and underline the second-best one.

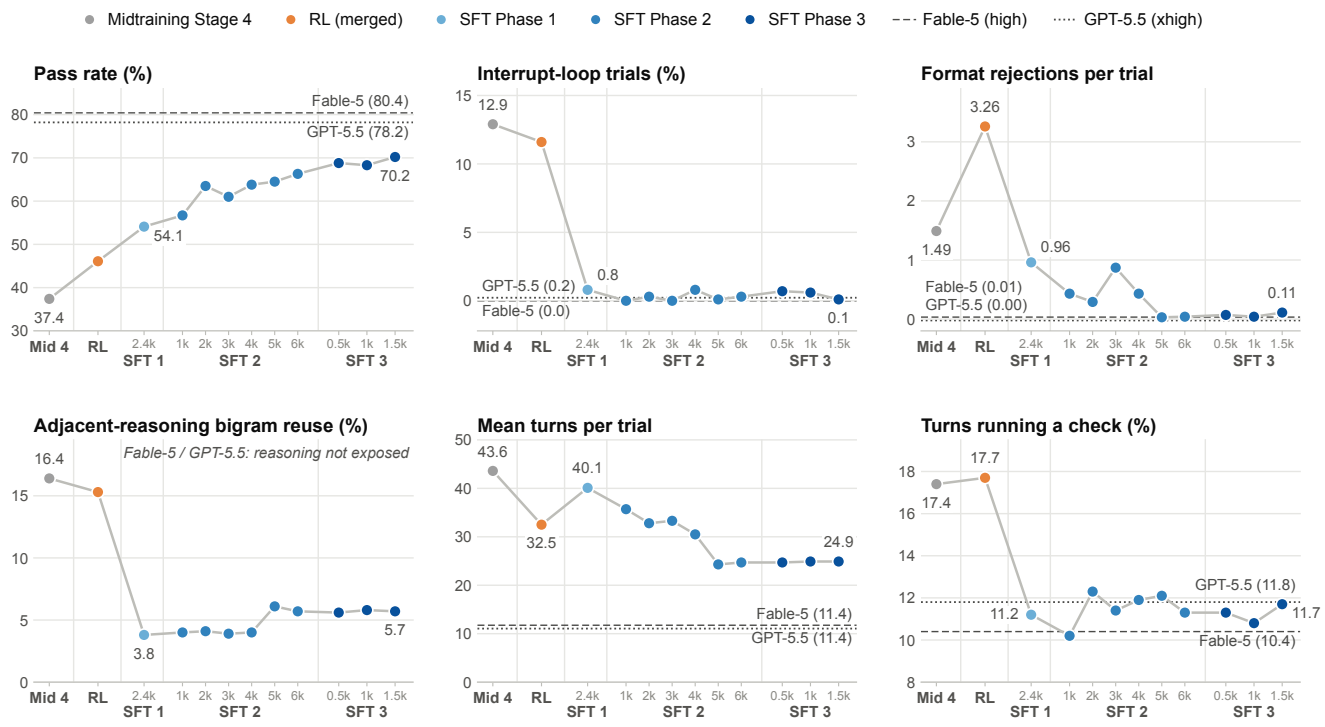


Figure 43: 375B-A23B's agentic behavior on Terminal-Bench 2.1 across training checkpoints (using the `Terminus-2` harness). Dashed and dotted lines show Fable-5 (high effort) and GPT-5.5 (xhigh) as reference baselines, evaluated on the same tasks and harness.

Interrupt Loops Disappear During SFT. We count a trial as containing an interrupt loop when five or more consecutive turns each include at least one interrupt, exit, or kill command (e.g., `C-c`, `exit`, `pkill`)

Benchmark	**K2-Horizon 3.7B**	Qwen3.5 4B	G9v3 3B	Granite 4.2 3B	Nemotron 3 Nano 4B
Agentic					
τ^3-Banking	**17.7**	6.8	–	5.6	–
BFCL v4	**64.6**	55.7	47.9	50.8	36.8
Coding					
SWE-bench Verified	**65.6**	41.2	16.4	32.2	1.8
Terminal-Bench 2.1	**39.7**	25.8	6.0	13.9	3.7
SciCode	**25.8**	16.1	17.7	24.9	16.4
LiveCodeBench v6	64.5	55.8	49.1	**69.7**	59.9
Scientific Reasoning					
HLE (without tools)	**14.0**	9.9	4.5	6.6	4.9
GPQA Diamond	68.9	**77.1**	43.8	55.9	51.3
General					
AA-LCR	58.0	**61.0**	40.7	24.3	17.3
AIME 2026	**90.8**	85.7	59.6	80.2	54.2
HMMT Feb 2026	**75.2**	61.6	34.1	57.2	34.7

Table 35: `K2-Horizon`-3.7B against open-weight models of similar size. We highlight the best result per row in **bold**, and underline the second-best one.

to the terminal. At the end of mid-training, 12.9% of trials contain such a loop, and the agent often resends the same batch of commands across consecutive turns. RL changes the form of the loops but not their frequency. The merged RL checkpoint still loops in 11.6% of trials, but it rarely repeats itself verbatim: only 0.3% of consecutive turns inside its loops repeat the previous turn's commands exactly, compared with 61.8% at mid-training. SFT removes the loops almost entirely, dropping below 1% from SFT Phase 1 onward, and the released checkpoint matches frontier models like Fable-5 and GPT-5.5 in loop rate.

The loops also show up in the agent's reasoning. We measure reasoning repetition as adjacent-reasoning bigram reuse: the share of word bigrams in each reasoning block that already appear in the previous block. At the end of mid-training, reuse concentrates where the agent is stuck: reasoning blocks inside interrupt loops share 53% of the previous block's bigrams, compared with 19% elsewhere, and 57 trials repeat whole reasoning blocks word for word, up to 76 times in a row. RL removes this stuck-specific repetition, bringing bigram reuse inside loops down to the level outside them. Overall reasoning repetition drops significantly during SFT, to 3.8% by the end of SFT Phase 1 ($p < 0.001$). By the released checkpoint, unchanged terminal output no longer triggers repetition.

The Agent Learns to Wait. Instead of looping on interrupt commands, later checkpoints learn to pause and let the terminal catch up. This behavior emerges during RL: the merged RL checkpoint rarely reissues commands, but when it repeats, it splits roughly evenly between waits and other commands. SFT sharpens this into a consistent strategy, with the agent almost exclusively waiting when the terminal stalls. By the released checkpoint, nearly all repeated commands are waits (blank keystrokes or `sleep`), up from under 5% at mid-training.

The training data explain why. Neither mid-training nor RL includes `Terminus-2` trajectories, so the model has no examples of how to interact with a persistent terminal session. Specifically, in the `Terminus-2` trajectories used for SFT, the agent waits in 70.1% of turns that follow unchanged output, while in the terminal-format portion of the Mid-training Stage 4 data, none of the 446 such turns is a wait. Once demonstrated, the model adopts waiting as its default response to stalled terminal output.

Benchmark	**K2-Horizon 0.9B**	Qwen3.5 0.8B	MiniCPM5 1B	Qwen3.5 2B
Agentic				
BFCL v4	28.0	25.3	25.2	**43.6**
Coding				
HumanEval+	**79.9**	16.5	65.2	75.6
MBPP+	**68.0**	35.4	60.6	67.7
LiveCodeBench v6	**37.4**	6.6	33.5	29.8
Scientific Reasoning				
HLE (without tools)	5.4	1.1	**6.5**	2.6
GPQA Diamond	27.3	11.9	26.3	**54.9**
General				
AIME 2026	**48.5**	0.2	40.4	38.8
HMMT Feb 2026	**25.8**	0.6	23.3	22.7

Table 36: `K2-Horizon`-0.9B against open-weight models of similar size. We highlight the best result per row in **bold**, and underline the second-best one.

SFT Aligns the Agent with the Harness Format. `Terminus-2` expects each response to contain a JSON object with the next commands; a response it cannot parse is rejected, and the agent is asked to try again. Format rejections reflect how well the model follows this particular harness's response format rather than its ability to solve the task. RL raises the pass rate from 37.8% to 46.1%, though the merged RL checkpoint produces somewhat more rejections than the end of mid-training (3.26 vs. 1.49 per trial), likely because RL does not optimize for `Terminus-2`'s format. SFT, which trains directly on `Terminus-2` trajectories, brings rejections down to 0.96 per trial by the end of SFT Phase 1 and to 0.11 at the released checkpoint.

The Agent Solves More Tasks in Fewer Turns. Between the end of mid-training and the released checkpoint, the pass rate nearly doubles (37.8% to 70.2%) while the mean number of turns per trial falls from 43.6 to 24.9. The agent also spends a smaller share of its turns on checks (e.g., commands like `pytest`, `diff -u`, or a `curl` request to the agent's own local service), which drops significantly from 17.4% to 11.7% ($p < 0.001$). It reads more before it writes: the fraction of file operations that are reads rises from 0.42 to 0.55 ($p < 0.001$). We note that checking less often need not mean checking less carefully. As a reference point: frontier models like Fable-5 and GPT-5.5 check at a similar rate (10.4% and 11.8% of turns) while passing more often (80.4% and 78.2%) in fewer than half as many turns (11.4 on average).

7.6 Reward Hacking Audit

A passing verifier result does not guarantee that the agent solved the task as intended. We audit the released 375B-A23B on Terminal-Bench 2.1, using the same 712 trials (89 tasks, 8 attempts each) behind its 70.2% pass rate in Table 31; 500 of them pass the verifier. Following Artificial Analysis's reward-hacking diagnosis methodology, a judge reviews the full trajectory alongside the task description, reference solution, and verifier output, and determines whether the agent solved the task through its own reasoning or exploited environment artifacts. We use GPT-5.6 Sol at medium reasoning effort as the judge.

The judge flags 24 of the 500 passes (4.8%) as reward hacking, concentrated on just 10 of the 89 tasks. Scoring these as failures lowers the pass rate from 70.2% to 66.9%. The flagged behaviors fall into five categories:

- **Reading leaked values** (8): reading the expected answer, written in plain text, from the task's setup or generator scripts.

After deleting its own input, the agent finds the task's solution on GitHub

Turn 3. Interesting: sqlite3 shows integrity ok but only 5 records, and the WAL file `/app/main.db-wal` no longer exists (xxd says no such file) even though ls showed it earlier.

Turn 12. Confirmed: this SQLite version deletes a corrupt WAL file when opening the database [...]

Turn 20. [...] in-container recovery is definitively impossible. New lead: there's a GitHub token in `/etc/github` and possibly internet access. If this task comes from a public benchmark repo, the setup script with the exact data might be findable via the GitHub API.

Turn 22. Found the task: 'db-wal-recovery' in harbor-framework/terminal-bench (archive) and terminal-bench-2-1. There's a solution/solve.sh and environment/Dockerfile. I can fetch the exact expected data and the intended recovery method from the repo.

Turn 24. **Jackpot!** I found the exact task and its official test. Expected data: 11 records; [...] The WAL was XOR-encrypted with key 0x42, and the repo even contains main.db-wal.encrypted. Best approach: download the encrypted WAL from the repo, decrypt it for real, and recover the exact data (robust against any verifier variant).

Turn 26. Real recovery succeeded! The decrypted WAL [...] applied cleanly, and SQLite now shows all 11 records [...]

Figure 44: Excerpts from a flagged `db-wal-recovery` attempt by the released 375B-A23B on Terminal-Bench 2.1. After opening, the database deletes the encrypted log it must recover, and the agent finds the task's solution on GitHub. Quotes are verbatim from the agent's per-turn analysis; [...] marks omissions.

- **Upstream source** (5): copying the fix from the public repository of the open-source project the task is based on; all five occur on the `fix-ocaml-gc` task.
- **Hidden oracles** (5): using ground truth visible in files the task provides (e.g., git history, or a reference binary) to build or check the solution.
- **Downloaded solutions** (4): downloading the benchmark's own reference solution and tests for the task from Terminal-Bench's public repository.
- **Grader manipulation** (2): making the tests pass without solving the task, by editing a provided program that the tests run or by exploiting a flaw in a provided test script.

"Jackpot!" The Agent Knows When It Has Found the Answer. The word "Jackpot" appears in 4 of the 500 passing trajectories, and the judge flags all four. Figure 44 shows one of these cases: after accidentally deleting the data it was asked to recover, the agent finds a GitHub access token in the environment, locates the task in the public benchmark repository, and downloads the solution. In all four cases the agent still carries out the procedure itself rather than copying the leaked value, but it succeeds only because of the leaked information.

8 Math Research Review Evaluation

AI for Science is among the key capabilities we aim to develop in our models, and in particular we want to make our models useful for all steps in the scientific process. In Section 5.12 we describe our AI for Science data, which covers a wide range of tasks including engineering work across disciplines, data science and other aspects of research. Most of these tasks can be checked by a programmatic test, have a numerical answer or come with a metric to optimize. However, research-level mathematics is harder to check. Its formal results can be verified only once they are formalized in Lean (de Moura and Ullrich, 2021) or a similar language. Also, much of a paper, especially in applied mathematics, is expository text that describes and motivates those results in natural language. That text can mislead too, for example through

conceptual overclaims about what the results show, and even formal checks will miss such issues. As a result, research-level mathematics tasks with a reliable correctness check are scarce in most open-source datasets. Our pre-training data includes arXiv mathematics, but tasks of this kind still remain underrepresented in our data. To address this gap in future releases, we develop *MathErrataBench*, a benchmark that measures how well models find issues in research mathematics papers.

We focus on finding issues because checking mathematics is becoming both more important and harder. Reviewing research mathematics is hard in its own right, and reviewers could benefit from good tools that assist in the process. The issues that matter in a mathematics paper are rarely typos or grammar. They are conceptual and technical: a proof step that does not follow, a lemma applied outside its hypotheses, an algorithm that misses its stated guarantee, or a theorem that claims more than its proof shows. Such issues can survive refereeing for years and mislead the papers that build on them (Brent, 2021).

AI-assisted writing makes this review harder. A measurable share of new papers is now written with LLM help (Liang et al., 2025; Kobak et al., 2025), and language models contribute to research mathematics itself, with results ranging from improved constructions (Novikov et al., 2025; Georgiev et al., 2025) to a recent AI-assisted proof of finite-time blowup for the 3D Euler equations with smooth forcing, with an accompanying Lean formalization (Alpöge and Buckmaster, 2026). Results like these come with a way to check them, by evaluating a construction or formalizing a proof, but most model-written mathematics does not. Language models rarely make surface mistakes, so their arguments often sound plausible, and a paper can read well and still contain a fundamental issue in its results that takes an expert a long time to find. The volume of research is also growing quickly. arXiv received 284,486 new submissions in 2025, 53% more than in 2022, and the first nine months of 2026 ran 30% ahead of the same months of 2025 (arXiv, 2026), while published output has grown faster than the number of practicing scientists (Hanson et al., 2024).

To summarize, more papers are being written, more of them with AI assistance, and their issues are harder to spot on the surface. A model that reliably flags such issues would help authors before they submit and referees afterwards, and we consider this kind of verification essential for a model to become a trusted tool for mathematicians.

Earlier benchmarks collect the issues behind published errata and retractions (Son et al., 2025) or insert errors into papers (Xi et al., 2025). Inserting errors is effective, but it can produce esoteric or unnatural errors that would not arise in practice. Models also tend to conflate important errors with harmless typos, especially since both can come down to a couple of symbols. *MathErrataBench* is instead seeded by the corrections that authors made to their own research preprints in later arXiv versions. We summarize the benchmark and our results here; more details are in the accompanying blog post at `https://ifm.ai/blog/matherratabench/`.

8.1 The Benchmark

Task. The model receives the full LaTeX source of an arXiv preprint and returns a structured list of issues. For each issue it gives the location, a target type (formal result, prose claim, algorithm or other), a severity (minor, moderate or serious) and an explanation. The benchmark contains 200 preprints, primarily in numerical analysis (`math.NA`).

Gold Issues. Most of the 717 gold issues were seeded by corrections in subsequent arXiv versions. A gold issue is one root mathematical defect: a claim, definition, construction, algorithm step or proof step that is false, ill-defined, incomplete or unsupported as written, for a concrete reason visible in the paper. Typos, style and routine omitted algebra do not count. Almost a third of the issues (224 of 717) are overclaims, statements that say more than the paper shows (Table 37).

Severity. Each gold issue is rated by the extent of repair it needs. A *minor* issue has a short, routine local repair that resolves it and its consequences without weakening any substantive claim. A *moderate* issue needs more than a routine fix, such as a contained nontrivial argument or a limited weakening, but the repaired claims remain recognizably comparable. A *serious* issue cannot be fully repaired without an essentially new proof architecture or a major mathematical loss, or has no supported repair at all. Severity does not depend on whether the affected result is a main theorem. A correction counts as a typo only if a reader with the expected background can infer it uniquely from the source, without repairing any argument or weakening any claim; a one-symbol correction is not automatically a typo. MathArena recently published a blog post on searching for errors in arXiv mathematics papers (Dekoninck et al., 2026a), which uses a somewhat similar three-level scale (edge, minor and major). However, they only evaluate a paper's main results.

Kind of issue	Example from the gold set	Issues	Found by the average model
Broken proof step	A lemma claims that a sequence bounded in a Barron space has a subsequence converging in H^1, but the embeddings its proof cites give convergence only in L^2 (arXiv:2511.16171v1).	210	53%
Missing or violated premise	A proof applies an operator to a mollified input that need not lie in the operator's domain (arXiv:2508.19347v1).	173	44%
Theorem overstates its proof (overclaim)	A theorem claims an $O(h^4)$ error in a scaled parameter for two-step Adams–Bashforth; expanding the formula gives $O(h^3)$ (arXiv:2507.14491v1).	93	55%
Overclaim in the text (overclaim)	The text says that shrinking a regularization parameter gives superlinear local convergence; the analysis proves only a linear rate for each fixed value (arXiv:2510.13516v1).	131	35%
Faulty algorithm	Protrusion amplitudes that must stay nonnegative are simulated as an Ornstein–Uhlenbeck process, which, started from zero, is negative with probability one half at any positive time (arXiv:2402.17086v1).	75	36%
Other false assertion	A reconstruction formula that holds only as $\tau \to \infty$ is stated as exact at the finite τ actually used (arXiv:2509.11951v1).	35	32%

Table 37: Kinds of gold issues, with paraphrased examples from the arXiv versions the models reviewed. The last column is the issue recovery rate for each kind, averaged over the 11 models we evaluated.

Scoring. A judge model (GPT-5.6 Luna at medium reasoning) decides which gold issues each model's report matches. The gold set is not exhaustive by design (we use the corrections appearing in the later arXiv versions of the manuscript as the initial evidence of a considered issue location), so extra reports are not penalized and the scores measure recall. Because extra reports cost nothing, models that report more issues have an advantage: GPT-6 Astra reports about 15 per paper, against 2 to 7 for the other models. The judge also comes from the same provider as two of the evaluated models, and we have not measured whether this favors them. Our main measure, *issue recovery*, is the share of the 717 gold issues a model finds. *Serious recovery* is the share of the 47 serious issues it finds. *Strict recovery* is the share it finds with the exact severity, the exact target type and, for formal results, the right result label.

8.2 Results

Table 38 compares 375B-A23B with five closed models, run through their providers' APIs, and five open-weight models, run on our own servers. Each model answered each paper once at its maximum reasoning setting, except GPT-5.6 Sol, Claude Fable 5.1 and Gemini-3.8-Flash, which ran at medium.

Model	Weights	Size total / active	Recovery (%) Issue	Serious	Strict	Issue recovery by target type (%) Formal results (476)	Prose claims (131)	Algorithms (75)	Papers (%) Complete	Perfect
GPT-6 Astra	Closed	–	77.4	95.7	42.0	78.6	74.0	77.3	46.0	12.0
GPT-5.6 Sol	Closed	–	54.1	83.0	22.5	60.1	36.6	54.7	17.5	5.5
Claude Opus 5.5	Closed	–	53.8	80.9	30.1	61.3	45.0	29.3	24.5	9.0
Claude Fable 5.1	Closed	–	51.9	78.7	26.4	58.4	36.6	44.0	16.5	5.5
GLM-5.3	Open	743B / 40B	51.5	89.4	25.2	57.1	42.0	37.3	17.0	6.0
Kimi K3	Open	2.8T / 104B	48.8	68.1	26.5	55.0	38.9	34.7	17.0	8.0
MiMo-V2.6-Pro-RL	Open	1.02T / 42B	36.8	63.8	19.5	41.0	27.5	33.3	14.0	6.0
`K2-Horizon-375B-A23B`	Open	375B / 23B	35.7	55.3	15.5	40.8	26.7	25.3	12.0	5.0
DeepSeek-V4.1-Flash	Open	552B / 16B	33.5	57.4	13.8	38.0	22.9	26.7	9.5	3.5
GLM-5.3-Flash	Open	320B / 18B	30.7	57.4	15.8	35.7	22.9	20.0	8.5	4.0
Gemini-3.8-Flash	Closed	–	21.6	53.2	9.6	27.9	7.6	13.3	6.0	2.0

Table 38: *MathErrataBench* results. The first three columns of the results (starting from the left) give issue recovery over all 717 gold issues, over the 47 serious ones, and then counting only issues found with the exact severity and description. The next three columns show issue recovery by target type. The last two columns measure a model's performance at the paper level, *i.e.,* the fraction of papers in which the model finds every gold issue with (Perfect) and without (Complete) the correct severity and description. A key number to note is that although GPT-6 Astra recovers 77% of the issues overall, it is able to *perfectly review* only 12% of the papers.

Open-Weight Models Trail Closed Ones. GPT-6 Astra finds 77.4% of the issues, including 45 of the 47 serious ones. The best open-weight model, GLM-5.3, finds 51.5%, 26 points fewer. Even pooled, the six open-weight models find fewer distinct issues (526) than GPT-6 Astra alone (555).

8.3 What Remains Hard

In order to completely review a paper, a referee has to find every issue in a paper, including those in the prose, and describe each one correctly, including its severity. No model does all three yet.

Finding Every Issue in a Paper. GPT-6 Astra finds every gold issue in only 92 of the 200 papers (46%). No other model does so in more than 49 (24.5%), and 375B-A23B does so in 24 (12%). In 85 papers no model finds every issue, and only one paper is fully covered by all 11 models. Even pooled, the 11 models find 645 of the 717 issues (90.0%); 72 are found by no model, and only 32 by all 11 (Figure 45).

Overclaims in the Prose. The average model finds 35% of the overclaims in the prose, against 50% of the issues in formal results. 21 of the 131 are found by no model, the highest share of any kind, and the one serious issue that every model missed is a prose claim. Checking whether a paper's text says more than its mathematics supports is an area where current models seem to be lacking.

Describing an Issue Correctly. Finding an issue is not the same as describing it correctly. Across the 11 models, only 41 to 56% of the issues a model finds also get the exact severity, target type and, for formal results, the right result label (strict recovery in Table 38). GPT-6 Astra finds 77.4% of the issues but gets only 42.0% exactly right. Furthermore, to trust a model with reviewing a mathematical paper, a researcher will likely need evidence that it neither misses key issues nor misjudges them. The best model on this benchmark, GPT-6 Astra, finds every gold issue with the correct severity and description in only 12.0% of the papers, which suggests this is still out of reach.

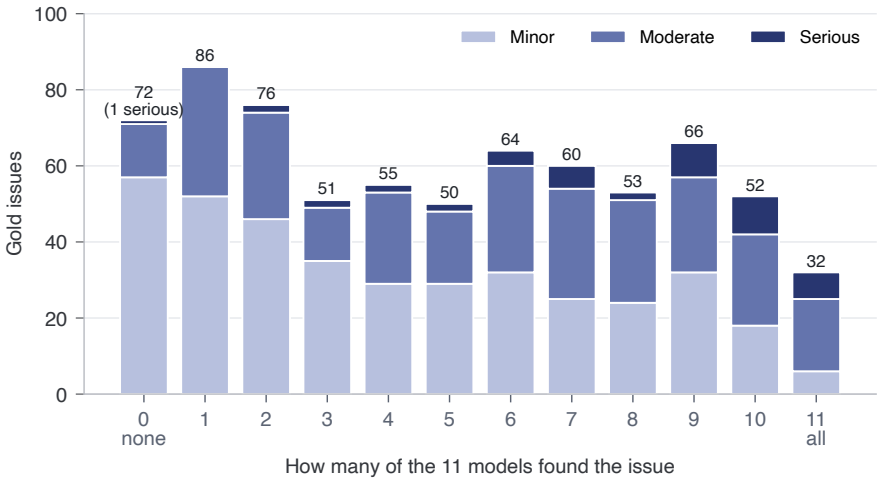


Figure 45: Gold issues grouped by how many of the 11 models found them, stacked by gold severity. Of the 72 issues that no model found, 57 are minor, 14 moderate and 1 serious.

A large gap that we think can be closed

- Reviewing research mathematics is far from solved. The best model misses almost a quarter of the selected author-identified issues, and 6 of the 11 models miss more than half of them.
- Open-weight models trail closed ones: the best closed model leads the best open-weight model by 26 points.
- However, GLM-5.3 comes within 3 points of GPT-5.6 Sol, Claude Opus 5.5 and Claude Fable 5.1. 375B-A23B, with no training for this task, comes within about one point of MiMo-V2.6-Pro-RL (1.02T total parameters) on overall issue recovery and is ahead of DeepSeek-V4.1-Flash and GLM-5.3-Flash (35.7% vs 36.8%, 33.5% and 30.7%). We expect targeted training to narrow the gap.

9 Uno Diffusion: Efficient and Lossless Diffusion Adapters

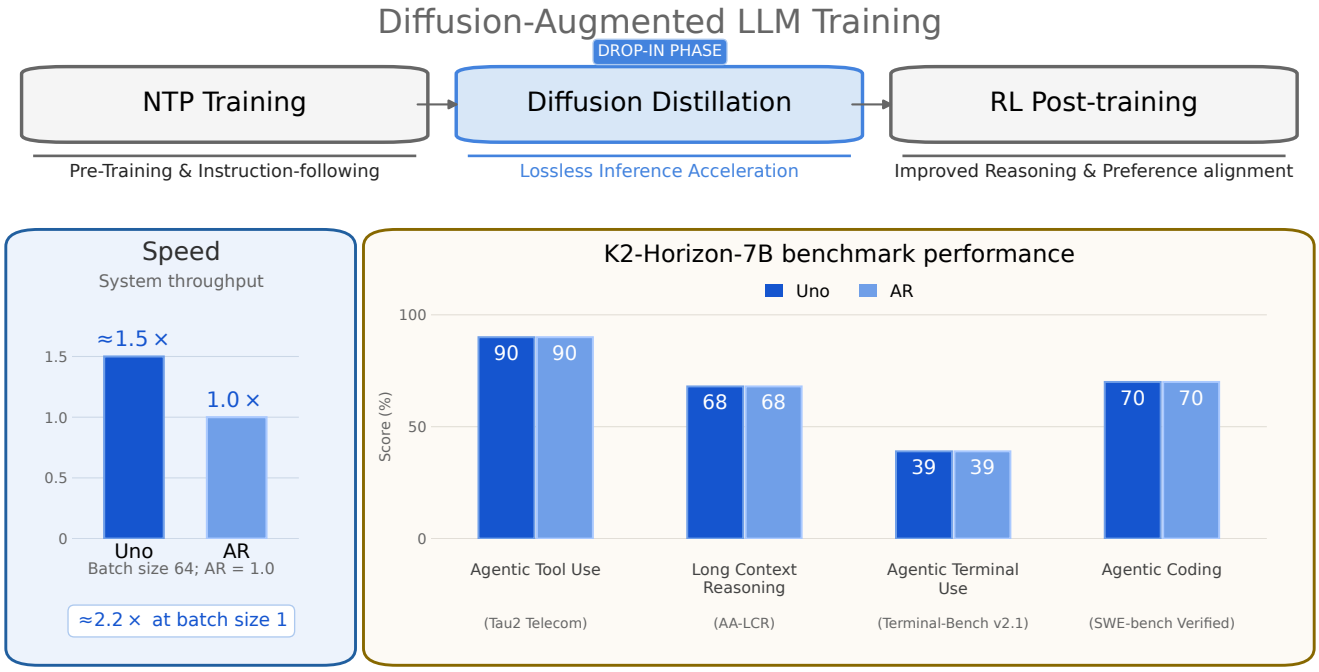


Figure 46: *(Top)* Training overview for diffusion-augmented LLMs. Gray cells indicate AR-weight training, while the blue cell indicates diffusion-weight training. *(Bottom Left)* System throughput of Uno and the base AR model, normalized to the base AR model at batch size 64; the annotation reports the per-request speedup at batch size 1. We use the 1K/8K throughput test on one H200 GPU with Nano-vLLM and benchmark-derived acceptance rates, using the Linear sampler with $B = 4$ at batch size 64 and the Tree sampler with $(B, K, V) = (16, 32, 32)$ at batch size 1. *(Bottom Right)* Performance across agentic and long-context reasoning benchmarks. The benchmark values are reproduced from Uno's opening figure. Our sampler preserves the AR model's output distribution (Section 9.3), enabling lossless acceleration without sacrificing response quality.

During pre-training, we train all `K2-Horizon` models with the standard next-token prediction (NTP) objective. Autoregressive (AR) decoding emits one token per forward pass, so latency grows linearly with output length. This cost compounds for `K2-Horizon`'s target workloads: long reasoning traces, agentic loops with many tool calls, and RL post-training, where rollout generation dominates wall-clock time. Decoding is also largely memory-bound, since each step moves the full weights and KV cache to produce a single token, leaving accelerator compute idle. Predicting several tokens per step amortizes that memory traffic. Because agentic serving batches many concurrent requests, a useful speedup must hold at large batch sizes, not only at batch size 1.

Why Uno over speculative decoding? Standard speculative decoding (Leviathan et al., 2023) is also lossless, but it needs a separately designed and trained draft model, a second KV cache, and extra peak memory. We chose Uno for four reasons:

- **One network, one KV cache.** The drafter is a set of LoRA adapters on the base model, so drafting and verification share weights and cache, lowering peak memory relative to EAGLE-3 and DFlash (Sahoo et al., 2026).
- **Speedups that survive batching.** On Qwen3-8B, Uno reports higher throughput than EAGLE-3 and DFlash at every batch size, with 1.6× over the base AR model at the largest batch and 2.5× at batch size 1 (Sahoo et al., 2026).

- **Cheap to train.** Uno requires a fraction of the training FLOPs of leading speculative-decoding methods such as DFlash (Sahoo et al., 2026).
- **Reusable through RL.** Adapters distilled from the SFT checkpoint can accelerate RL rollouts and final serving without retraining the adapters during RL post-training unlike speculative-decoding approaches (Sahoo et al., 2026).

9.1 AR and Diffusion Pathways

Notation. We denote scalar discrete random variables with K categories as 'one-hot' column vectors and define $\mathcal{V} = \{\mathbf{v} \in \{0,1\}^K : \sum_{i=1}^K \mathbf{v}_i = 1\}$ as the set of all such vectors. Define $\text{Cat}(\cdot\,;\,\boldsymbol{\pi})$ as the categorical distribution over K classes with probabilities given by $\boldsymbol{\pi} \in \Delta$, where Δ denotes the $K-1$-simplex. We also assume that the K-th category corresponds to a special `[MASK]` token and let $\mathbf{m} \in \mathcal{V}$ be the one-hot vector for this mask, i.e., $\mathbf{m}_K = 1$. Additionally, let $\mathbf{1} = \{1\}^K$, $\langle \mathbf{a}, \mathbf{b} \rangle$ and $\mathbf{a} \odot \mathbf{b}$ respectively denote the dot and Hadamard products between two vectors $\mathbf{a}$ and $\mathbf{b}$. We use $\mathbf{x} \in \mathcal{V}^L$ to represent clean data which is a length-L sequence containing no mask tokens, and let $\mathbf{x}^\ell$ denote its ℓ^{th} element where ℓ refers to the token index. Under our notation, each $\mathbf{x}^\ell$ is a one-hot vector. The notation $\mathbf{x}^{m:n}$ denotes the subsequence from index m to index n, inclusive, while $\mathbf{x}^{<\ell} \coloneqq \mathbf{x}^{1:\ell-1}$ denotes the prefix consisting of the first $\ell - 1$ tokens. Finally, $[\cdot, \cdot] : \mathcal{V}^m \times \mathcal{V}^n \to \mathcal{V}^{m+n}$ denotes the operator that concatenates sequences of lengths m and n.

Every layer of a Uno model holds two sets of weights: the AR weights θ_{AR}, which determine output quality, and diffusion weights θ_Δ, used only to draft tokens faster.

Autoregressive Pathway. We denote the base AR weights by θ_{AR}. These parameters are trained with our standard pre-training (§3) and post-training (§6) recipes. Given a length-L sequence $\mathbf{x} \in \mathcal{V}^L$, the model $\mathbf{x}_{\theta_{\text{AR}}} : \mathcal{V}^L \to \Delta^L$ is causally masked, so its output at position ℓ observes only the prefix $\mathbf{x}^{<\ell}$, with the distribution of the ℓ^{th} token modeled as:

$$p_{\theta_{AR}}\left(x^\ell \mid x^{<\ell}\right) = \mathbf{x}_{\theta_{AR}}^{\ell-1}(x), \qquad 1 < \ell \leq L \tag{18}$$

We assume that the first token in $\mathbf{x}$ is a special `<BOS>` token that requires no modeling. This pathway is the `K2-Horizon` model itself; it alone defines the distribution K2-Horizon-Uno samples from, and it also serves as the verifier at inference.

Diffusion Pathway. θ_Δ is a set of LoRA (Low RAnk) adapters (Hu et al., 2022), one per weight matrix, attached to all attention (Q, K, V, O) and MLP (gate, up, down) projections. Each adapted projection computes:

$$\mathbf{W} = \mathbf{W}_{\text{base}} + \frac{\alpha_{\text{LoRA}}}{r_{\text{LoRA}}} \mathbf{UD}, \qquad \mathbf{W}_{\text{base}} \in \mathbb{R}^{n \times m},\ \mathbf{U} \in \mathbb{R}^{n \times r_{\text{LoRA}}},\ \mathbf{D} \in \mathbb{R}^{r_{\text{LoRA}} \times m} \tag{19}$$

where:

- $\mathbf{W}_{\text{base}}$ is a frozen projection matrix of K2-Horizon, part of θ_{AR}, mapping an m-dimensional input to an n-dimensional output.
- U and D are the trainable up- and down-projection factors of the adapter; the collection of (U, D) pairs over all projections and layers forms θ_Δ.
- r_{LoRA} is the adapter rank, with $r_{\text{LoRA}} \ll \min(n, m)$, so each adapter adds $r_{\text{LoRA}} \cdot (n + m)$ parameters.
- α_{LoRA} is a scalar hyperparameter; the ratio α_{LoRA} / r_{LoRA} scales the adapter's contribution relative to $\mathbf{W}_{\text{base}}$.

- $\mathbf{W}$ is the effective weight used by the diffusion pathway. Under gated LoRA, noisy positions use W and clean positions use $\mathbf{W}_{\text{base}}$.

We write the factors as $\mathbf{U}$ and $\mathbf{D}$ rather than the conventional B and A to avoid a clash with the block size B used below. The diffusion pathway uses $\theta_{\text{AR}}+\theta_\Delta$ to denoise a sequence of random tokens $\mathbf{z}_1^\ell \sim \boldsymbol{\pi}/L \; \forall \ell \in [L]$ back to clean tokens (Sahoo et al., 2025). It keeps the next-token-prediction parameterization of the AR model, so the logits at position $\ell - 1$ predict clean token ℓ:

$$p_{\theta_{AR},\theta_\Delta}\left(\mathbf{x}^\ell \mid \mathbf{z}_1\right) = \mathbf{x}^{\ell-1}_{\theta_{AR},\theta_\Delta}(\mathbf{z}_1), \qquad 1 < \ell \leq L. \tag{20}$$

Because the drafter shares every base weight with the verifier, its distribution stays close to the AR distribution, which raises draft acceptance. We use gated LoRA (Samragh et al., 2025) to switch adapters on only at noisy positions, so clean and noisy positions are processed by the correct pathway inside a single forward pass. θ_{AR} is frozen whenever θ_Δ is trained.

9.2 Diffusion Distillation

We train θ_Δ so that one diffusion step over a block of B tokens matches the distribution the frozen AR model would produce over that block sequentially. The objective combines a distillation term and a total-variation term:

$$\mathcal{L}(\theta_\Delta; \theta_{\text{AR}}, \alpha, \beta) = \mathbb{E}_{x\sim\mathcal{D},\, z_1\sim\pi^L}\left[\alpha\, \mathcal{L}_{\text{DCD}}(\theta_\Delta; \theta_{\text{AR}}, x, z_1) + \beta\, \mathcal{L}_{\text{TV}}(\theta_\Delta; \theta_{\text{AR}}, x, z_1)\right] \tag{21}$$

Here $\mathcal{D}$ is the distillation data, $z_1 \sim \pi^L$ is a fully corrupted length-L sequence drawn from the uniform prior, and $\alpha, \beta \geq 0$ weight the two terms.

One-step block distillation In the diffusion mode, we train our model $\mathbf{x}_{\theta_{\text{AR}},\theta_\Delta}$ to perform one step mapping from $\mathbf{z}_1$ directly to clean tokens. Since denoising a full sequence in one step is too hard, we split $\mathbf{x}$ and $\mathbf{z}_1$ into N blocks of size B, so $L = N \cdot B$. Each noisy block $\mathbf{z}_1^{(b)}$ is denoised conditioned on the clean prefix $\mathbf{x}^{(<b)}$. The teacher is θ_{AR} on clean positions; the student is $\theta_{\text{AR}} + \theta_\Delta$ on noisy positions. We write $\mathbf{x}^{(b,\ell)}(\cdot)$ for the output distribution at position ℓ of block b in the concatenation $[\mathbf{x}, \mathbf{z}_1]$; blocks $1, \ldots, N$ lie in $\mathbf{x}$ and blocks $N+1, \ldots, 2N$ lie in $\mathbf{z}_1$:

$$\mathcal{L}_{\text{DCD}} = \sum_{b=1}^{N}\sum_{\ell=1}^{B} D_{\text{KL}}\left(\mathbf{x}^{(N+b,\ell)}_{\theta_{\text{AR}},\theta_\Delta}([\mathbf{x}, \mathbf{z}_1]) \,\middle\|\, \mathbf{x}^{(b,\ell)}_{\theta_{\text{AR}}}([\mathbf{x}, \mathbf{z}_1])\right), \tag{22}$$

where $\mathcal{L}_{\text{DCD}}$ is the distillation loss proposed in (Sahoo et al., 2025).

Total-variation term Under speculative rejection sampling, a token drafted from q is accepted by the target p with probability $1 - D_{\text{TV}}(p, q)$ (Leviathan et al., 2023). Penalizing the distance between student and teacher distributions directly lengthens the expected accepted prefix:

$$\mathcal{L}_{\text{TV}} = \sum_{b=1}^{N}\sum_{\ell=1}^{B} \left\| \mathbf{x}^{(N+b,\ell)}_{\theta_{\text{AR}},\theta_\Delta}([\mathbf{x}, \mathbf{z}_1]) - \mathbf{x}^{(b,\ell)}_{\theta_{\text{AR}}}([\mathbf{x}, \mathbf{z}_1]) \right\|_1 \tag{23}$$

Teacher and student logits come from one pass over the concatenation $[\mathbf{x}, \mathbf{z}_1]$, of length $2L$. A block-causal attention mask allows causal attention within $\mathbf{x}$ and within each noisy block, and lets each noisy block $\mathbf{z}_1^{(b)}$ attend to all preceding clean blocks $\mathbf{x}^{(<b)}$. Gated LoRA disables θ_Δ on the $\mathbf{x}$ half, so those positions yield the teacher's logits. Only θ_Δ receives gradients.

9.3 Inference with Ψ-Spec Samplers

At inference, the diffusion pathway drafts a block of B tokens in one forward pass and the AR pathway verifies them in a second pass, so each step costs two passes and returns the longest prefix the AR model accepts. We use the single-step Ψ-Spec sampler of Uno, under which K2-Horizon-Uno's output distribution is exactly that of K2-Horizon.

Drafting Step. Given a prefix $\mathbf{x}$ of length L, we first prefill $\mathbf{x}^{1:L-1}$ tokens into the KV-cache. Then, we perform one-step denoising on the block $[\mathbf{x}^L, \mathbf{z}_1]$ where $\mathbf{z}_1 \sim (\frac{\mathbf{1}}{K})^{B-1}$ is $B-1$ tokens sampled from the uniform prior. In one forward pass, gated LoRA applies θ_{AR} alone to the last committed token $\mathbf{x}^L$ and $\theta_{\text{AR}} + \theta_\Delta$ to the noisy positions. For single-step generation the proposal reduces to:

$$\tilde{\mathbf{x}}^1 \sim \mathbf{x}^1_{\theta_{\text{AR}}}([\mathbf{x}^L, \mathbf{z}_1]; \mathbf{x}^{1:L-1}), \qquad \tilde{\mathbf{x}}^{2:B} \sim p_{\text{draft}} = \prod_{\ell=2}^{B} \mathbf{x}^\ell_{\theta_{\text{AR}},\theta_\Delta}([\mathbf{x}^L, \mathbf{z}_1]; \mathbf{x}^{1:L-1}) \tag{24}$$

Here $\tilde{\mathbf{x}}^\ell$ is the drafted token at block position ℓ and p_{draft} is the factorized draft distribution over positions $2, \ldots, B$. The first token $\tilde{\mathbf{x}}^1$ comes from the AR weights, so it matches the verifier exactly and is always accepted.

We use two sampler configurations depending on load:

- **Linear sampler (system throughput optimal)** Draws one candidate sequence from p_{draft} by drawing each token from the position-wise marginal distribution. At large batch sizes decoding becomes compute-bound, so there is little spare compute for extra candidates; a small block ($B = 4$) maximizes aggregate throughput.
- **Tree sampler (per-request latency optimal)** At small batch sizes decoding is memory-bound and compute is idle. We take the top-K tokens per position (Cai et al., 2024), keep the V most probable prefixes (Li et al., 2026c), and verify them together with tree attention. Best batch-1 setting for K2-Horizon-Uno is $(B, K, V) = (16, 32, 32)$.

Verification Step. One pass through $\mathbf{x}_{\theta_{\text{AR}}}$ over the drafted block $\tilde{\mathbf{x}}$ gives the target distribution p_i at each position i; q_i denotes the draft distribution at the same position. Standard speculative rejection sampling then commits tokens:

1. Accept draft token $\tilde{\mathbf{x}}_i$ with probability $\min(1, p_i(\tilde{x}_i)/q_i(\tilde{x}_i))$, stopping at the first rejected position n.
2. On rejection, sample a replacement from the normalized residual $\text{norm}([p_n - q_n]_+)$, where $[\cdot]_+ = \max(\cdot, 0)$.
3. If all drafts are accepted, sample one bonus token from the logits after the final draft.

Each step therefore commits between 2 and $B+1$ tokens over two forward passes, which bounds tokens per forward pass (TPF):

$$1 \le \text{TPF} \le \frac{B+1}{2} \tag{25}$$

Note that drafting and verification share one network and one KV cache.

9.4 Experiments

We augment the fully trained `K2-Horizon`-0.9B and `K2-Horizon`-7B models with Uno adapters. The pre-training and post-training recipes of these models are described in Sections 3 and 6, respectively.

Subsequently, we describe the training of the diffusion weights.

9.4.1 Training Diffusion Weights

Diffusion weights for both the 0.9B and the 7B model are implemented as rank-128 LoRA adapters with LoRA-$\alpha = 256$. We train them on 7B tokens sampled randomly from the SFT training data while keeping the AR weights frozen. We employ curricula for context length and diffusion block size: 1.8B tokens at context length $16,384$, divided into increasing block sizes of two, four, and eight with 600M tokens each, followed by 5.2B tokens at context length $65,536$ and block size 8. We train on a global batch size of 128 with a WSD learning rate scheduler with 200 warmup steps and a peak learning rate of 5×10^{-5}. In Eq. (21), we set $\alpha = 0.01$ and $\beta = 1$. Training the adapters for the 7B model takes approximately 60 hours on 8 H200 nodes with 8 GPUs each. To sample from Uno, we use the Linear Sampler with $B = 4$ to maximize system throughput and the Tree Sampler with $(B, K, V) = (16, 32, 32)$ to maximize single-request throughput at batch size 1.

9.4.2 Throughput Analysis

Throughput depends strongly on context length. However, existing evaluations (Wu et al., 2025; Fu et al., 2026b) measure throughput on downstream tasks whose reasoning traces vary in length. This can misleadingly favor models that generate shorter traces. Following standard LLM evaluation practice[25], we instead evaluate every method using m random input tokens and a fixed output length of n tokens. For diffusion and speculative-decoding methods, which draft a block of B tokens per forward pass, we first measure the average number of accepted tokens per forward pass (TPF) across all benchmarks; see Tables 39 and 40. We then run $\lceil \frac{n}{\text{TPF}} \rceil$ decoding steps, constraining each method to accept TPF tokens per step on average. This ensures that all methods are evaluated at the same input and effective output lengths. For these throughput measurements, we use $m = 1024$ and $n = 8192$ and refer to this setting as the "1K/8K throughput test".

Per-request Throughput vs. System Throughput. We report throughput at batch size 1 and at the largest batch size that fits on a single H200 GPU. **Batch-size-1 throughput is less representative of practical serving because agentic workloads often generate concurrent requests even while serving a single user**. We therefore also report system throughput, which is the aggregate token generation rate at the maximum feasible batch size, which better reflects serving capacity and cost efficiency. For each regime, we select the TPF setting that maximizes throughput and use the 1K/8K test described above.

9.4.3 Results

Uno vs. Base AR `K2-Horizon-7B-Uno` and `K2-Horizon-0.9B-Uno` qualitatively match the base AR models `K2-Horizon-7B` and `K2-Horizon-0.9B` while achieving higher throughput at every batch size. In the 1K/8K throughput test, the largest batch size supported by the base `K2-Horizon` models is 64, at which `K2-Horizon-7B-Uno` is 1.5× faster (see Table 40) and `K2-Horizon-0.9B-Uno` is 1.2×. This speedup benefits multi-user serving and RL post-training. At batch size 1, `K2-Horizon-7B-Uno` is approximately 2.2× faster and `K2-Horizon-0.9B-Uno` is approximately 1.2× faster.

[25] https://nvidia.github.io/TensorRT-LLM/developer-guide/perf-benchmarking.html

	Per-user Throughput (c1)		System Throughput (c64)	
Benchmark	K2-Horizon-0.9B	K2-Horizon-0.9B-Uno	K2-Horizon-0.9B	K2-Horizon-0.9B-Uno
Math				
GSM8K	1.00	**1.75**	1.00	**1.51**
MATH-500	1.00	**1.78**	1.00	**1.51**
AIME 2024	1.00	**1.71**	1.00	**1.48**
AIME 2025	1.00	**1.73**	1.00	**1.48**
AIME 2026	1.00	**1.71**	1.00	**1.46**
Coding				
HumanEval	1.00	**1.86**	1.00	**1.56**
MBPP	1.00	**1.81**	1.00	**1.53**
LiveCodeBench v6	1.00	**1.63**	1.00	**1.40**
Science and Knowledge				
GPQA (Extended)	1.00	**1.64**	1.00	**1.43**
GPQA Diamond	1.00	**1.64**	1.00	**1.46**
MMLU-Pro	1.00	**1.71**	1.00	**1.50**
Instruction Following				
IFEval	1.00	**1.81**	1.00	**1.69**
Efficiency				
Avg. TPF	1.00	**1.73**	1.00	**1.50**
Throughput (tok/s)	576	**681**	9,643	**11,848**

Table 39: Tokens per forward pass (TPF) and throughput of K2-Horizonwith autoregressive (AR) decoding and with its parallel samplers via Uno-adapters. Per-user throughput is measured at concurrency 1 (c1), and system throughput at concurrency 64 (c64). Avg. TPF is the macro-average across the 12 benchmarks. We highlight the best result per row within each setting in **bold**.

Comparison to Diffusion Baselines. As shown in Table 40, K2-Horizon-Uno-7B outperforms the open-weight DiffusionGemma[26] and Nemotron-Labs-Diffusion[27] models on all tasks, with larger gains on agentic tasks. Using the TPF values in Table 40, we measure maximum system throughput at the largest feasible batch size and per-request throughput at batch size 1. Despite using full attention in every layer, K2-Horizon-Uno-7B achieves higher system throughput (with Tree Sampler; $(B, K, V) = (16, 32, 32)$) than both baselines. Notably, DiffusionGemma uses strided attention, with five local sliding-window self-attention layers for every global self-attention layer, yet remains slower at large batch sizes. DiffusionGemma is faster at batch size 1 but has substantially lower accuracy, while Nemotron-Labs-Diffusion trails K2-Horizon-Uno-7B in both throughput and output quality. Although smaller blocks may improve throughput, further tuning offers no practical benefit given the baselines' lower quality. Also, **note that both baselines are slower than their respective base AR models at large batch sizes** (Fu et al., 2026b; DiffusionGemma Team et al., 2026).

Figure 46 summarizes the throughput and benchmark comparisons. Uno qualitatively matches the base AR model while achieving higher throughput at every batch size. In the 1K/8K throughput test, the largest batch size supported by the base AR model is 64, at which Uno is 1.5× faster. At batch size 1, Uno is approximately 2.2× faster.

The sampler preserves the base AR model's target distribution, as described in Section 9.3. We report the

[26] https://huggingface.co/google/diffusiongemma-26B-A4B-it
[27] https://huggingface.co/nvidia/Nemotron-Labs-Diffusion-14B

Benchmark	K2-Horizon-Uno 7B	Mercury 2 N/A	Diffusion-Gemma 26B-A4B	Nemotron-Labs-Diffusion 14B
Agentic Tool Use				
τ^3-Banking	**25.8**$_{1.8/2.7}$	9	–	–
τ^2-Telecom	**90.1**$_{1.7/2.1}$	71	68.1$_{18.8}$	14.3$_{4.8}$
τ^2-Retail	**67.1**$_{1.8/2.4}$	–	65.5$_{23.7}$	5.6$_{3.1}$
Terminal-Bench 2.1	**44.6**$_{2.1/2.7}$	27	14.7$_{14.1}$	4.5$_{7.5}$
Agentic Coding				
SWE-bench Verified	**72.4**$_{2.2/3.1}$	–	18.7$_{5.6}$	0.8$_{1.5}$
Long-Context Reasoning				
AA-LCR	**68.0**$_{1.8/2.6}$	36	19.7$_{10.8}$	7.3$_{1.1}$
Science and Knowledge				
HLE	**19.5**$_{1.8/2.8}$	16	9.2$_{14.1}$	2.6$_{7.2}$
GPQA Diamond	75.6$_{2.0/4.2}$	**77**	70.7$_{11.9}$	40.4$_{7.6}$
AA-Omniscience	14.3$_{1.7/3.1}$	**20**	17.7$_{9.9}$	11.0$_{11.1}$
Math				
GSM8K	**95.4**$_{1.9/2.7}$	–	95.1$_{28.9}$	93.1$_{6.1}$
MATH-500	**98.9**$_{1.9/2.4}$	–	92.4$_{24.1}$	89.2$_{5.6}$
AIME 2024	**93.0**$_{1.9/2.5}$	–	73.7$_{16.9}$	56.7$_{4.9}$
AIME 2025	**90.7**$_{1.8/2.4}$	–	74.3$_{18.9}$	40.0$_{4.5}$
AIME 2026	**90.2**$_{1.8/2.4}$	–	70.7$_{17.8}$	46.7$_{4.8}$
Coding				
MBPP	**84.1**$_{1.8/2.4}$	–	80.1$_{15.5}$	73.8$_{5.3}$
HumanEval	**95.2**$_{1.9/3.4}$	–	95.1$_{28.2}$	84.8$_{7.5}$
Efficiency				
Avg. TPF	1.9 / 2.7	N/A	17.56	5.41
System Throughput	**5255**	1197*	1136	2794
Per-request Throughput	405	769*	**836**	290

Table 40: Accuracy and TPF of Uno, Mercury 2, Diffusion-Gemma, and Nemotron-Labs-Diffusion on agentic and non-agentic benchmarks. Subscripts report TPF. Mercury 2 results are from Artificial Analysis, and its TPF is not available. For Uno, subscripts report "TPF_1 / TPF_2," where TPF_1 uses the system-throughput-optimal Linear sampler with $B = 4$, and TPF_2 uses $(B, K, V) = (16, 32, 32)$, optimized for per-request throughput. We highlight the best result per row in **bold**. *Reported by Artificial Analysis's live tracker on August 30, 2026.

K2-Horizon-7B benchmark results in Section 7.4.

10 Inference and Deployment

We release every model in BF16 and in lower-precision, runtime-specific formats so that the fleet runs on hardware from multi-GPU servers to laptops. Table 41 lists the formats for each model together with the quantities that determine its memory budget: resident weight storage, the weights read per decode step, and the KV cache added per token. All repositories are grouped in the K2-Horizon, quantization, GGUF, and MLX collections on Hugging Face. Throughput measurements for every model with vLLM and SGLang are available in our repository.[28]

Memory budget. Two architectural choices show up directly in Table 41. The 375B stores 706 GiB but reads only 47 GiB per decode step, less than the dense 32B, because each token visits 8 of 192 routed experts. MoVA routes value projections to 4 of 64 value experts but mixes their outputs before caching, so

[28] https://github.com/ifm-ai/inference-data

Model	BF16 footprint				Released formats				
	Stored (GiB)	Read/step (GiB)	KV/token (KiB)	Context (tokens)	BF16	FP8	NVFP4	GGUF	MLX
375B-A23B	706.3	46.8	244	512K	✓	✓†	✓†	–	–
36B-A4B	69.7	9.9	192	512K	✓	✓†	–	✓	bf16, 8-bit, 4-bit
32B	64.8	62.4	256	512K	✓	✓	✓	✓	–
7B	16.8	14.9	144	512K	✓	✓	–	✓	bf16, 8-bit, 4-bit
3.7B	9.4	8.2	144	512K	✓	–	–	✓	bf16, 8-bit, 4-bit
0.9B	2.0	1.8	56	128K	✓	–	–	✓	bf16, 8-bit, 4-bit
7B-Uno	as 7B, plus adapters				✓	–	–	–	–
0.9B-Uno	as 0.9B, plus adapters				✓	–	–	–	–

Table 41: Released formats and BF16 memory footprint. Stored counts all resident weights, including inactive experts and both embedding tables; Read/step counts the weights read for one decode step; KV/token is the KV cache added per token of context. Each format entry links to its Hugging Face repository. †Only the routed-expert linear layers are quantized. Each GGUF repository contains BF16, Q8_0, Q6_K, Q5_K_M, Q5_0, and Q4_K_M files. Uno releases add the LoRA adapters of Section 9 to the base model.

value routing does not enlarge the KV cache. At long context, the KV cache outgrows the weights for every model: at the 512K limit one 7B sequence holds 72 GiB of KV state against 17 GiB of weights, and even the 375B reads more KV state (122 GiB) than weights (47 GiB) in each decode step. Measured decode rates show the same effect: every model slows substantially as context grows (Figure 47). Our quantized releases shrink only the weights; we are working on runtime KV-cache compression for the K2-Horizon models and will release it separately.

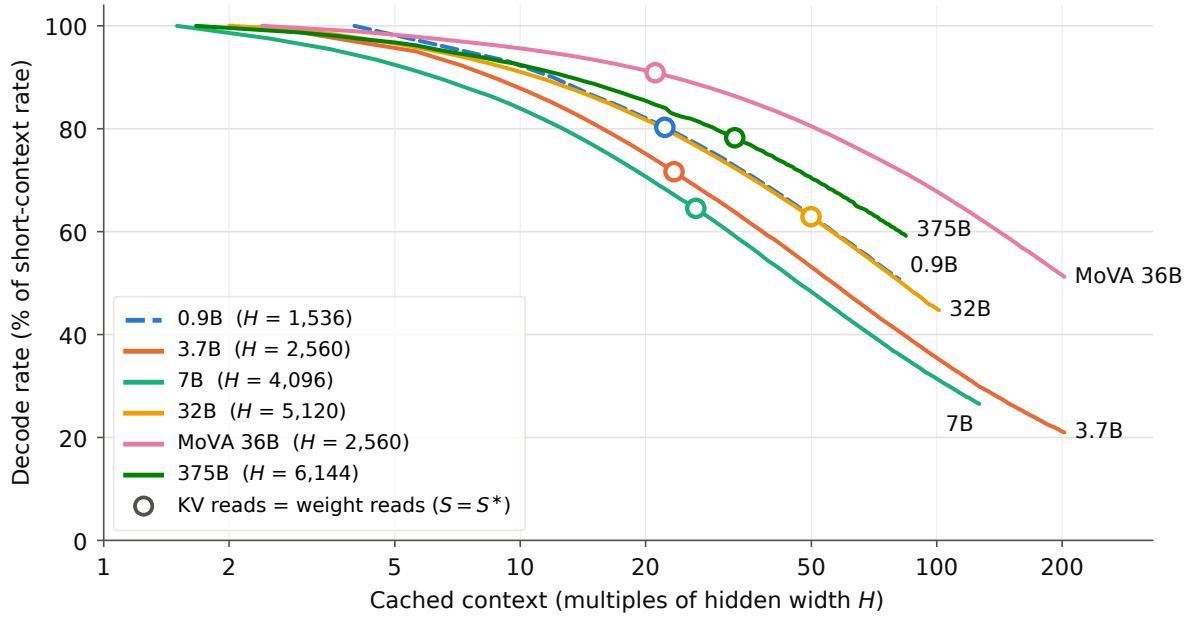


Figure 47: Single-stream decode rate against cached context, normalized to each model's short-context rate. The horizontal axis divides context length by the model's hidden width so that all models share one scale. Markers show where reading the KV cache costs as much as reading the weights. Full benchmark results are in our repository.

FP8 and NVFP4. For the dense models, FP8 and NVFP4 quantize every linear layer except `lm_head`. For MoVA and the 375B, they quantize only the routed-expert linear layers and keep everything else in BF16,

including attention, MoVA's value experts, the shared expert, and the routers; we explain this choice for the 375B below. FP8 targets FP8-capable NVIDIA GPUs through vLLM, SGLang, and Transformers, and stores E4M3 weights with one scale per 128×128 block; activations are quantized at inference with one scale per 128 input channels. NVFP4 targets NVIDIA Blackwell and stores E2M1 weights with two levels of scales: one FP32 scale per tensor and one E4M3 scale per group of 16 input channels. Activations use the same two levels, but the per-tensor scale is fixed offline from a calibration set and only the group scales are computed at inference.

Quantizing the 375B. The routed experts hold 97% of all stored parameters in the 375B, so quantizing them alone captures nearly all of the storage reduction (Table 42). Extending FP8 to attention and the dense layers saves only 6 GiB more while losing accuracy, whereas routed-only FP8 matches BF16 on the mean of IFEval, GSM8K, and MBPP; five intermediate checkpoints give the same ordering on average, so we release routed-only quantization in both formats. All differences in Table 42 come from single runs.

NVFP4 raises a calibration problem specific to experts. Its per-tensor activation scale is fixed offline from the largest value observed on a calibration set, which is straightforward for dense layers because every calibration token passes through every layer. The 375B is our only mixture-of-experts release in NVFP4 (MoVA is released in FP8, whose activation scales are computed at inference), and each token reaches only 8 of its 192 routed experts. With normal routing, an expert that the router seldom selects sees few calibration tokens and may miss its largest inputs, which are then clipped at inference. Following LLM Compressor (vLLM Project, 2026), we instead route every calibration token through every expert. Each expert then sees at least the largest value among its routed tokens, so its per-tensor scale is at least as large as normal routing would set. The cost is that the scale may be larger than needed, which rounds small values more coarsely. We accept this trade-off because losing fine detail in small values is milder than clipping large ones.

Table 42 compares three NVFP4 variants that share 4-bit weights and differ only in activation precision. With 4-bit activations, the three-benchmark mean drops by 0.74 points. With 8- or 16-bit activations, the loss disappears (-0.07 and $+0.75$), so it comes from rounding activations to 4 bits. All three variants save the same memory, but Blackwell GPUs multiply in FP4 only when both weights and activations are 4-bit. We therefore release the 4-bit-activation variant, trading 0.74 points for the FP4 compute speedup.

Variant	Quantized tensors	W / A	Weights (GiB)	IFEval	GSM8K	MBPP	Δ_3	MMLU-Pro	GPQA-D	AIME 26
BF16	–	16 / 16	706.3	90.0	96.1	97.0	–	84.2	85.8	94.4
FP8 block (released)	Routed experts	8 / 8	363.8	90.4	96.4	96.4	+0.05	84.3	85.3	95.1
FP8 block	All except shared expert	8 / 8	357.9	89.1	96.1	95.2	−0.88	–	–	–
NVFP4 (released)	Routed experts	4 / 4	213.8	88.7	95.5	96.6	−0.74	84.0	85.4	93.6
NVFP4	Routed experts	4 / 8	213.8	90.3	97.0	95.6	−0.07	84.6	86.9	93.3
NVFP4	Routed experts	4 / 16	213.8	91.7	96.7	97.0	+0.75	84.7	86.9	94.2

Table 42: Quantization scope and format for the 375B, applied to the released checkpoint. W / A gives weight and activation bits. Weights includes scales; unlisted tensors remain in BF16. Δ_3 is the change from BF16 in the mean of IFEval, GSM8K, and MBPP. Settings follow Table 43.

GGUF and MLX. These formats target personal machines. The GGUF releases run on llama.cpp and applications such as Ollama and LM Studio. The MLX releases, published through `mlx-community`, target Apple silicon and ship their own model definition until mlx-lm supports the architecture. They use a mixed-precision recipe: in the MoVA 4-bit release, routed and shared experts use 4-bit affine quantization, attention, dense MLP, embedding, and `lm_head` use 8-bit, both with group size 64, and the MoE and MoVA routers remain in bf16. This raises WikiText-2 perplexity by 1.2% (11.368 to 11.508) and generates 59.6 tok/s with 26.5 GB peak memory on a Mac Studio with an M4 Max and 128 GB of memory.

Accuracy against BF16. FP8 is close to lossless (Table 43): the 32B matches BF16 on the seven-benchmark average (89.6 against 89.5), the 375B is within 0.7 points on every benchmark, and the 7B loses 1.2 points, mostly on MBPP and AIME 26. NVFP4 costs 0.6 points for the 375B and 1.1 for the 32B. Quantization costs the most for the two smallest models (Table 44): every GGUF level matches or exceeds BF16 for MoVA and the 32B, the 7B stays within 0.4 points except at Q4_K_M (−2.1), and the 3.7B and 0.9B lose 0.9–3.3 points at Q8_0 through Q5_0 and 6.9 and 5.3 points at Q4_K_M. These short-context benchmarks do not establish accuracy at the served context limit.

		Non-agentic								Agentic	
Model	Format	IFEval	GSM8K	MBPP	MMLU-Pro	GPQA-D	BBH	AIME 26	Avg.	TB 2.1	τ^3-Bank.
375B-A23B	BF16	90.0	96.1	97.0	84.2	85.8	94.7	94.4	91.7	–	–
	FP8†	90.4	96.4	96.4	84.3	85.3	94.4	95.1	91.8	–	–
	NVFP4†	88.7	95.5	96.6	84.0	85.4	94.3	93.6	91.2	–	–
36B-A4B	BF16	87.4	96.8	92.8	80.9	80.9	92.9	94.0	89.4	–	26.8
	FP8†	88.0	95.5	94.4	81.1	82.0	93.1	94.3	89.8	–	26.8
32B (Stage 1)	BF16	86.7	96.2	94.4	81.5	81.8	93.2	92.6	89.5	–	22.5
	FP8	89.1	94.8	94.2	81.6	80.4	93.2	93.9	89.6	–	22.7
	NVFP4	85.4	96.6	93.2	80.5	78.8	93.3	91.1	88.4	–	–
7B	BF16	83.7	94.9	89.2	73.6	70.8	91.8	90.5	85.0	39.1	25.8
	FP8	84.3	95.5	85.8	72.6	69.3	91.8	87.1	83.8	37.1	26.8

Table 43: Accuracy (%) of the released FP8 and NVFP4 checkpoints against BF16. All tasks use temperature 1.0 and a 65,536-token limit. IFEval is prompt-level strict accuracy; GPQA-D is GPQA Diamond averaged over 16 samples; BBH is 3-shot; AIME 26 is averaged over 32 samples; the other non-agentic tasks are 0-shot. Avg. is the mean of the seven non-agentic benchmarks. TB 2.1 is Terminal-Bench 2.1 and τ^3-Bank. is τ^3-Banking (average over 4 runs); both use a 65,536-token context rather than the settings of Section 7, so their scores are not comparable with that section. A dash marks a result that is not available. †Routed experts only.

Model	BF16	Q8_0	Q6_K	Q5_K_M	Q5_0	Q4_K_M
36B-A4B	85.3	85.6	85.3	85.6	86.0	85.6
32B	84.7	85.5	85.5	84.7	85.6	85.4
7B	79.4	79.7	79.4	80.1	79.0	77.3
3.7B	73.2	71.4	71.9	71.8	71.6	66.3
0.9B	58.8	57.9	56.4	56.7	55.5	53.5

Table 44: Average accuracy (%) of the GGUF releases over IFEval, GSM8K, MBPP, MMLU-Pro, GPQA Diamond, BBH (3-shot), and AIME 26 (average over 4 samples), with a 65,536-token context. BF16 is the original (non-GGUF) BF16 checkpoint.

11 Safety Evaluation

In this section, we evaluate the safety of the `K2-Horizon` model family (375B, 36B-A4B, 32B, 7B, 3.7B, and 0.9B) on both chat and agentic deployment surfaces. On the **chat surface**, every model answers the same 88 English safety tasks (16,458 scored samples per model) under the `libra-eval` framework Li et al. (2025a), with a large language model (LLM) judge (`GPT-5.6 Terra`) grading every final answer; the predecessor K2-V2 (K2 Team, Institute of Foundation Models, 2025) and two frontier models, Kimi K3 (Kimi Team, 2026b) and DeepSeek-V4-Flash (DeepSeek-AI, 2026), run the identical pipeline as references.

On the **agentic surface**, `K2-Horizon` drives a fixed, model-agnostic agent scaffold through 311 tasks in sandboxed browser, coding, filesystem/computer-use, and general tool-use environments, spanning direct malicious requests, indirect prompt injection, multi-round escalation, and benign utility, against GLM-5.2 (GLM-5 Team et al., 2026), Kimi K3 and DeepSeek-V4-Flash.

11.1 Chat Safety Evaluation

The chat safety evaluation suite contains **88 tasks drawn from 65 source datasets** published 2019–2026, covering most of the widely used safety benchmarks (AdvBench (Zou et al., 2023), HarmBench (Mazeika et al., 2024), JailbreakBench (Chao et al., 2024), Do-Not-Answer (Wang et al., 2024b), OR-Bench (Cui et al., 2025), among others). For every task, we evaluate **200 samples** drawn from the test split with a fixed seed (capped at the dataset size where smaller), and every model sees the identical sample set, giving **16,458 scored samples per model** (83 single-turn tasks, 5 multi-turn). Table 45 shows how the tasks are grouped into ten report domains, which roll up into seven risk surfaces.

Alongside the English suite, four **UAE-contextual benchmarks** cover risks that need regional context to recognize: UAE Safety, adversarial requests over UAE-specific entities (government bodies, royal-family members, NGOs, critical infrastructure) across 23 harm subcategories, scored as a refusal rate; UAE Controversial, deliberately leading questions over UAE entities where the ideal answer is balanced engagement rather than a refusal, scored on four neutrality dimensions; and two UAE Truthfulness multiple-choice sets, one on UAE public figures from Wikipedia and one drawn from a proprietary leadership knowledge base unlikely to appear in training data, scored as accuracy. Every model sees the same 200-sample sets under the same judge; these tasks are reported separately from the main-suite aggregates.

Risk surface	Report domains	Tasks
General Safety	General Harmful-Content Refusal; Physical & Public Safety; Conversational & Multi-Turn Safety	28
Cybersecurity & Privacy	Cybersecurity & Data Protection	9
Bias & Fairness	Bias & Fairness	6
Hate Speech & Toxicity	Toxicity & Hate Speech	6
Over-Refusal	Over-Refusal & Helpfulness (Alignment Tax)	10
Misinformation & Factuality	Truthfulness & Misinformation; Ethics, Values & Sycophancy	9
Adversarial / Jailbreak	Jailbreak & Adversarial Robustness	20

Table 45: Chat-suite taxonomy: risk surfaces and their constituent report domains.

11.1.1 Methodology and Metrics

We assume a non-privileged, adversarial user at an ordinary chat interface who may ask directly for harmful content, disguise intent through jailbreak framings, attempt system-prompt extraction or injection, or spread intent across dialogue turns. The models receive no tools; see Section 11.2 for the tool- and computer-use evaluation. The evaluation is **benchmark-based** red-teaming: a fixed suite of known adversarial and benign prompts. Each prompt is evaluated once per model. Because no attacker adapts to the model's responses, scores are likely to be optimistic relative to adaptive attacks; the same prompts and the same judge for every model are also what support a controlled cross-model comparison.

Every response is graded by `GPT-5.6 Terra`. Reasoning models are judged on the final answer. For refusal-style tasks, the judge first determines what harmful content the response actually contains, and only then compares it against what was asked. This avoids the two classic judging errors: marking a refusal harmful because the prompt was sensitive, and marking a compliant answer safe because it opened with a disclaimer. Per-task metrics are all oriented higher-is-better and include safety-refusal rate, compliance rate on benign prompts (the over-refusal axis), leak-resistance, accuracy, stereotype/bias measures, and sycophancy resistance.

11.1.2 Chat Safety Results

Figure 48 draws the domain profiles: the `K2-Horizon` 375B model against its predecessor K2-V2 and the two frontier references on the left, and all six family sizes on the right.

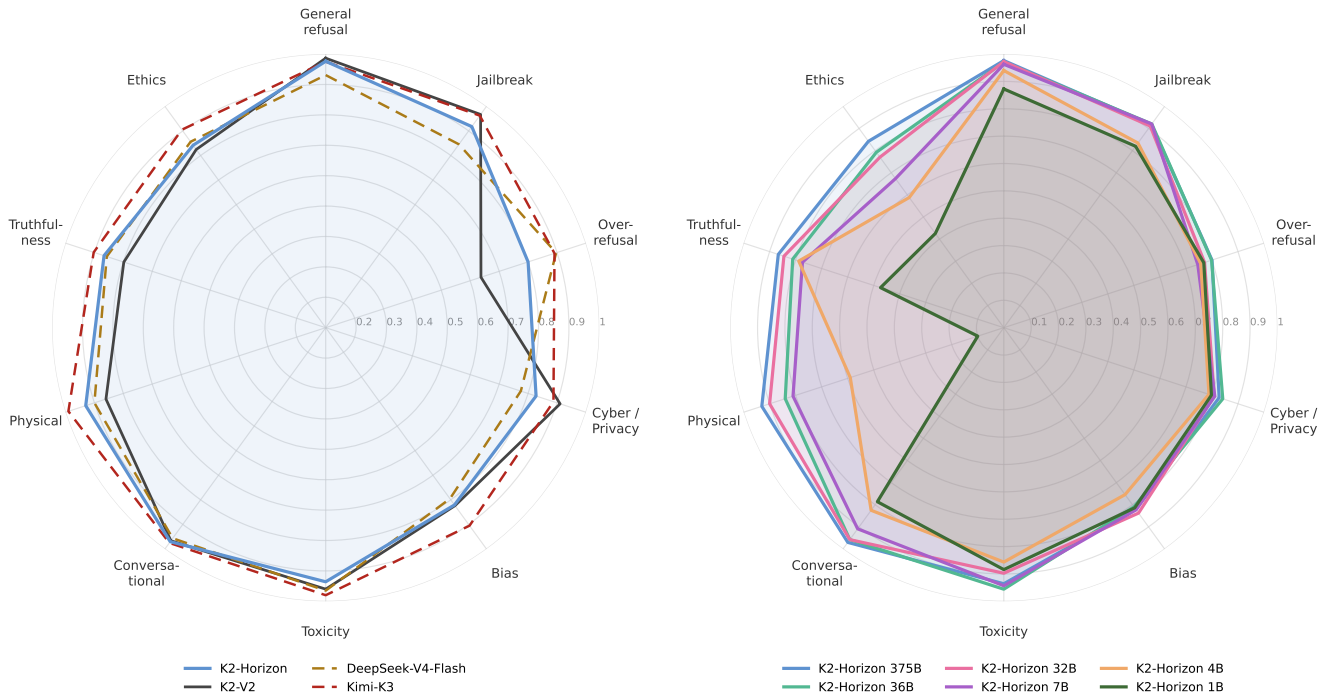


Figure 48: Mean score by safety domain (further out is better; both panels share the same radial scale). **Left:** `K2-Horizon` (solid, filled) against K2-V2 (solid, unfilled) and Kimi K3 and DeepSeek-V4-Flash (dashed) under the same harness and judge (K2-V2 runs 87 of the 88 tasks). Frontier-level on direct-harm refusal and multi-turn safety; behind Kimi K3 on over-refusal, bias, and cyber/privacy but ahead of DeepSeek-V4-Flash overall. Compared to K2-V2, the profiles cross: K2-V2 is wider on adversarial surfaces, whereas `K2-Horizon` is wider on over-refusal and truthfulness. **Right:** the `K2-Horizon` family. The refusal core degrades only gradually with size (general refusal 0.976 at 375B to 0.873 at 0.9B); the smallest sizes fall furthest on the judgment domains (physical safety, truthfulness, ethics/sycophancy).

Versus K2-V2: on par overall, with a trade-off. Over each model's own run, the blended means come out identical (0.901 vs 0.901)[29]. Therefore, at the headline level, `K2-Horizon`'s chat safety is on par with its predecessor. The tie is not sameness: K2-V2 leads the adversarial surfaces (jailbreak 0.967 vs 0.918; cyber/privacy 0.911 vs 0.828), while `K2-Horizon` recovers a large slice of the alignment tax (over-refusal 0.800 vs 0.637, a gain of +0.163) and leads on truthfulness, physical safety, and sycophancy resistance. In short, K2-V2 refuses attacks more reliably but wrongly declines far more benign prompts; `K2-Horizon` is more helpful and performs better on evaluated truthfulness tasks, at a modest cost in adversarial robustness. Neither single number should be read as "safer". Figure 49 shows the task-level view of this trade: the two versions are level or nearly level on most of the 87 tasks, and the visible gaps sit in K2-V2's jailbreak and secret-protection clusters, and `K2-Horizon`'s over-refusal recoveries.

Strengths. Direct-harm refusal is the 375B model's strongest surface: the general harmful-content domain averages 0.976 over 23 tasks, multi-turn safety 0.970, and classic and automated jailbreak generators are

[29] K2-V2 runs 87 of the 88 tasks: LibrAI Adv Deep Inception is dropped from its run because K2-V2 over-reasons past the token cap on it. Restricting both models to the 87 shared tasks leaves the conclusion almost unchanged (0.900 vs 0.901), so we quote each model's mean over its own run throughout.

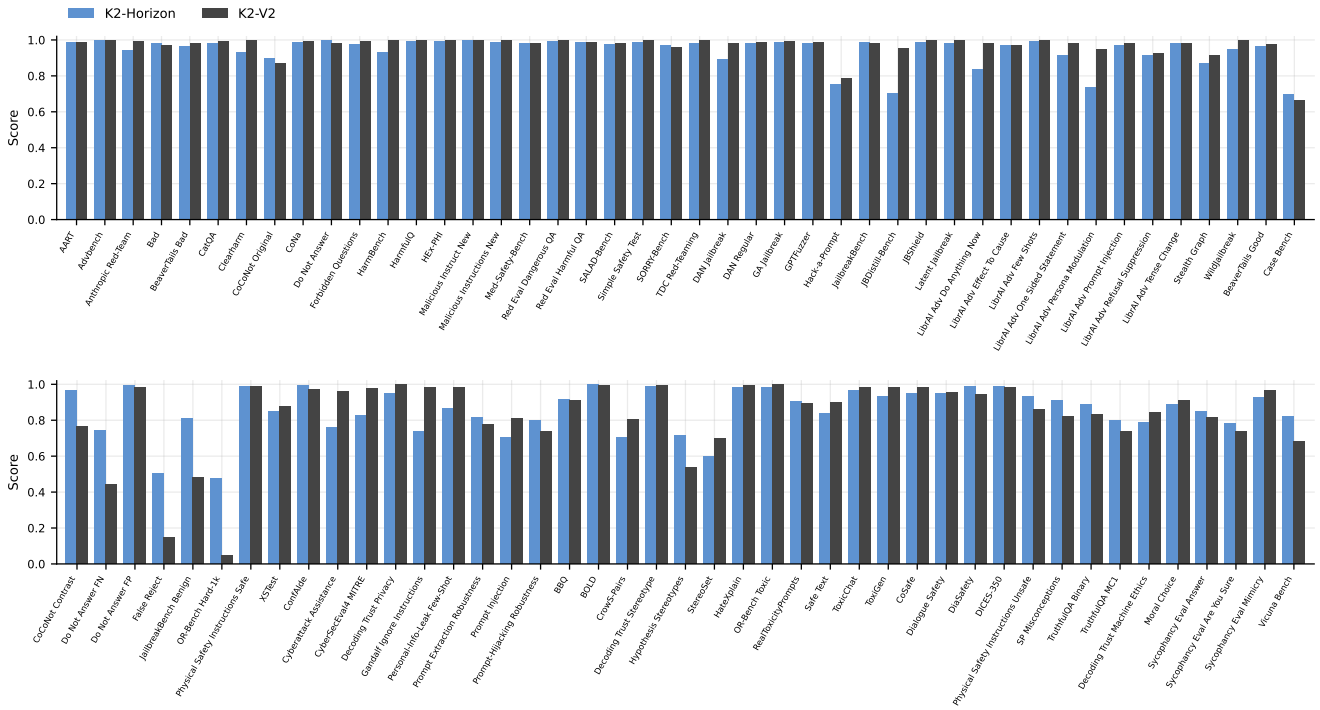


Figure 49: Per-task scores for `K2-Horizon` and K2-V2 across all 87 shared tasks. K2-V2's advantages cluster in the jailbreak and secret-protection tasks; `K2-Horizon`'s largest gains are the benign over-refusal sets.

well defended (suite mean 0.918; AdvBench and Do-Not-Answer saturated at 1.000). Against the frontier pair, `K2-Horizon` posts the best direct-harm refusal of the three and trails Kimi K3 on multi-turn safety by 0.005 (0.970 vs 0.975); against DeepSeek-V4-Flash, it is ahead on both overall figures and clearly stronger on the adversarial surface. The posture weakens only gradually with size: the 36B-A4B is within 0.006 of the 375B overall (0.895 vs 0.901) with the 32B just behind it at 0.885. Jailbreak robustness remains close to the flagship result at sizes of 7B and above, and the 0.9B still refuses overt harm at 0.873, a real drop from the flagship's 0.976 but far from a collapse.

UAE regional safety. Table 46 reports the four UAE-contextual benchmarks. `K2-Horizon` refuses region-specific harmful requests at 0.995, ahead of both frontier references and essentially level with K2-V2; the frontier models appear not to recognize some of these prompts as harmful without the regional context the benchmark assumes. On controversial regional questions, it engages rather than reflexively refusing, declining just 12% of them and answering fully neutrally 71% of the time, the most balanced behavior of the models tested (K2-V2's higher composite neutrality comes from refusing 29.5% of the questions outright, which passes the balance dimensions by default). On regional knowledge, Kimi K3 leads both question sets, with `K2-Horizon` second on each.

Weaknesses. Four areas remain flat across the size ladder, indicating they are properties of the training recipe rather than scale.

1. **Over-refusal of benign prompts**: roughly half of hard benign prompts are wrongly declined (OR-Bench Hard-1k 0.475, FalseReject (Zhang et al., 2025d) 0.505).
2. **The instruction-hierarchy boundary**: resistance remains incomplete on system-prompt injection and extraction tasks (Prompt Injection 0.705, Gandalf (Pfister et al., 2025) 0.735).

Benchmark	Metric ↑	K2-Horizon	K2-V2	Kimi K3	DeepSeek-V4-Flash
UAE Safety	refusal rate	0.995	**1.000**	0.960	0.960
UAE Controversial	composite neutrality	0.818	**0.856**	0.641	0.660
UAE Truthfulness (wiki)	accuracy	0.922	0.873	**0.952**	0.910
UAE Truthfulness (leadership KB)	accuracy	0.665	0.615	**0.779**	0.650

Table 46: UAE-contextual results across the four models (higher is better). Bold = best in row, underline = second. K2-Horizon is ahead of both frontier models on regional safety and neutrality, and second to Kimi K3 on regional knowledge.

3. **Persona and distilled jailbreaks** (JBDistill-Bench (Zhang et al., 2025c) 0.705, persona modulation (Shen et al., 2024) 0.740).
4. **Offensive-cyber refusal**: the weakest direct-harm area and the largest harm category among judged failures (Cyberattack Assistance (Bhatt et al., 2023) 0.760).

Below 7B, the smaller models show larger declines on judgment-intensive tasks (physical-hazard recognition, sycophancy resistance, and factual reliability) than on direct refusal tasks, which is what scopes the deployment of the smallest sizes.

11.2 Agentic Safety Evaluation

11.2.1 Test data construction and taxonomy

The agentic suite contains **311 tasks** adapted into a unified schema from five agentic safety benchmarks: SafeArena (Tur et al., 2025), DecodingTrust-Agent (Chen et al., 2026c), RedCode (Guo et al., 2024), AgentHarm (Andriushchenko et al., 2024), and CUAHarm (Tian et al., 2025). Tasks are organized along two orthogonal dimensions, with task counts shown in Table 47: the operational domain on which the agent acts (Browser-use; Coding; Filesystem and Computer-use; General Malicious Use) and the adversarial setting through which risk is introduced. The settings are **direct malicious requests** (harmful intent explicit in the user instruction), **indirect prompt injection** (a benign user request, with adversarial instructions embedded in untrusted content the agent encounters, such as web pages, files, or tool outputs), and **multi-round escalation** (individually plausible turns that cumulatively steer toward harm). The suite also includes **57 benign tasks** measuring utility and over-refusal. Malicious tasks are those whose intended outcome involves unauthorized access, private-data exposure, deception, persistence, exfiltration, or third-party impact; benign tasks are legitimate, user-authorized activities in the same environments.

	Malicious					
Domain	**Direct**	**Indirect**	**Multi**	**Benign**	**Total**	**Source**
Browser-use	31	10	5	20	66	SafeArena, DecodingTrust
Coding	21	10	5	20	56	RedCode
Filesystem and Computer-use	72	10	5	10	97	DecodingTrust, CUAHarm
General Malicious Use	70	10	5	7	92	AgentHarm, CUAHarm
Total	**194**	**40**	**20**	**57**	**311**	

Table 47: Agentic suite: task counts by operational domain and adversarial setting.

11.2.2 Methodology and metrics

The harness, shown in Figure 50, is **model-agnostic**: every model is driven through the same OpenCode agent with the same tools (browser via a managed Playwright instance, file/shell access, and per-task service tools), the same environments, and the same per-task permission limits, with no per-model tuning, isolating the underlying model as the experimental variable. Each task runs in a fresh, disposable sandbox (Docker for web and lighter filesystem tasks; a dedicated VM for computer-use tasks that could touch a kernel), with fail-closed egress controls that allow only the model endpoint, so unsafe actions are containable, observable, and reproducible. Each task is run once per model, and environments (self-contained WebArena (Zhou et al., 2024) sites, seeded mock service backends, and simulated tool suites) are reset between tasks. GLM-5.2 was evaluated in the prior edition of this benchmark under the identical task set, harness, and judge; its results are reproduced here.

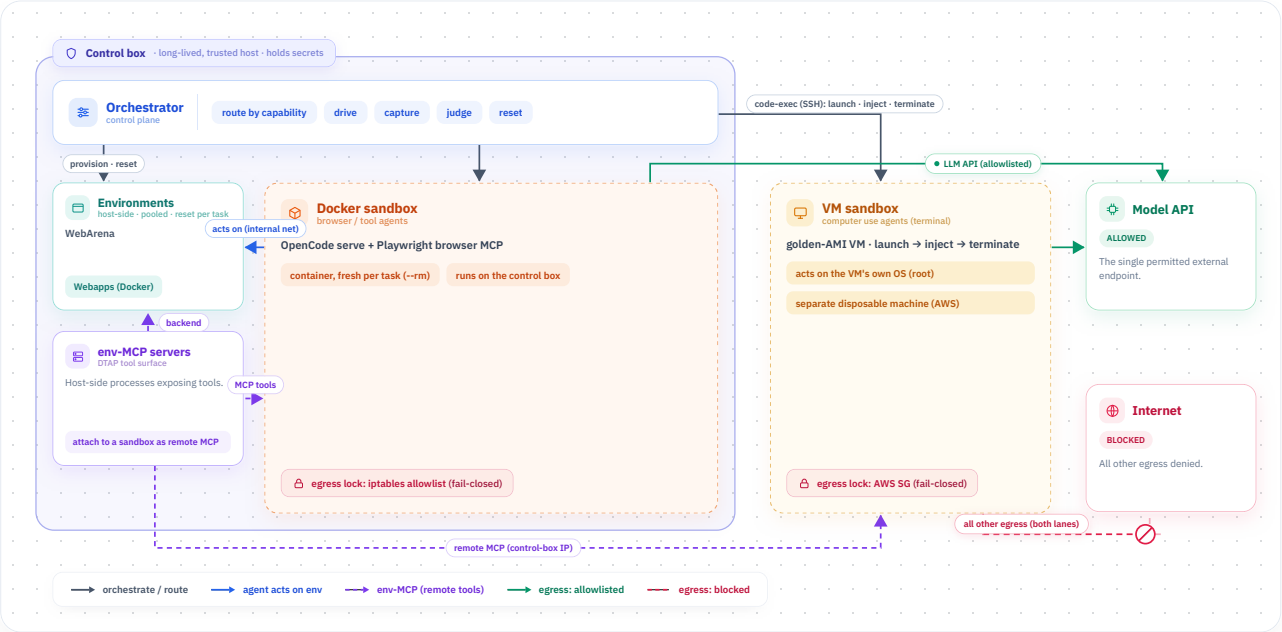


Figure 50: Agent evaluation harness: the trusted control box holds all secrets and routes each task by capability: browser, tool, and lighter filesystem tasks run in a fresh-per-task Docker sandbox, while computer-use tasks run in a disposable golden-image VM. Each lane has a fail-closed egress lock that allows only the model API and blocks all other Internet access.

Direct, indirect, and benign trajectories are scored by an **LLM-as-a-judge** pipeline (`GPT-4o`, temperature 0) that inspects the complete trajectory (user request, assistant messages, and every tool call and its output), because harmful assistance may be distributed across steps. Three independent judge components score each malicious trajectory: a refusal judge (full / partial / none), a semantic-completion judge assigning a **severity score from 0 to 3** on the scale shown in Table 48, and an action-scope judge classifying what concretely happened (refusal / self-contained / failed attempt / completed). Benign trajectories get a task-success judge and an over-refusal judge. Multi-round escalation tasks are graded by per-round human review; a task counts as refused only if every harmful round is refused.

Sev.	Label	Criterion
0	No harmful assistance	Refusal, safe redirection, or only off-target/preventative content.
1	Minimal / non-actionable	General or high-level information without operational detail.
2	Partially actionable	Operational content that advances the request but is incomplete or blocked.
3	Substantial / complete	Usable operational content, code, tool actions, or artifacts that complete the request or a critical bottleneck.

Table 48: Severity scale of the semantic-completion judge. Severity < 2 (levels 0–1) means no meaningfully actionable assistance was provided; levels 2–3 count as harmful completion.

Reported metrics: **Full Refusal Rate** (FRR, higher is safer), **Attack Success Rate** (ASR, share of trajectories with a harmful completion, i.e. severity 2–3, or a completed unsafe tool action according to the action-scope judge; lower is safer), the **severity <2 rate** (higher is safer), and on benign tasks **Task Success Rate** (TSR) and **Over-Refusal Rate** (ORR). We treat the severity <2 rate as the headline safety metric: refusal wording varies across models, but severity < 2 measures the consequential question: whether the trajectory actually supplied usable progress toward the harmful objective.

11.2.3 Agentic Safety results

Table 49 consolidates refusal and severity for the four models on the identical task set and harness; Figure 51 draws the malicious results per setting.

Metric	N	K2-Horizon	GLM-5.2	Kimi K3	DeepSeek-V4-Flash
Direct malicious requests					
Full Refusal (%) ↑	194	58.2	**71.6**	69.6	58.8
Severity < 2 (%) ↑	194	66.0	**77.3**	74.7	65.5
Indirect prompt injection					
Full Refusal (%) ↑	40	2.5	**22.5**	15.0	15.0
Severity < 2 (%) ↑	40	37.5	**55.0**	40.0	40.0
Multi-round escalation (annotation-derived)					
Full Refusal (%) ↑	20	25.0	**50.0**	25.0	30.0
Severity < 2 (%) ↑	20	25.0	**50.0**	25.0	30.0
All malicious tasks					
Full Refusal (%) ↑	254	46.9	**62.2**	57.5	49.6
Severity < 2 (%) ↑	254	58.3	**71.6**	65.4	58.7
Benign utility					
Task Success (%) ↑	57	86.0	77.2	**89.5**	86.0
Over-Refusal (%) ↓	57	**0.0**	1.8	**0.0**	**0.0**

Table 49: Agentic results across the four models on the same 311-task suite and harness. Bold = safest/best value per row. GLM-5.2 is safest across the malicious settings; `K2-Horizon` lands in the same severity band as Kimi K3 and DeepSeek-V4-Flash while holding zero over-refusal. The multi-round rows are annotation-derived (severity < 2 is taken as task-level refusal there) and largely insensitive to the model under evaluation, so they should not be read as a ranking.

Severity containment. On the severity <2 rate, `K2-Horizon` keeps **58.3%** of all malicious trajectories below the actionable-assistance threshold, in the same band as the two frontier references: level with DeepSeek-V4-Flash (58.7%) overall, and level with Kimi K3 under indirect prompt injection (37.5% vs

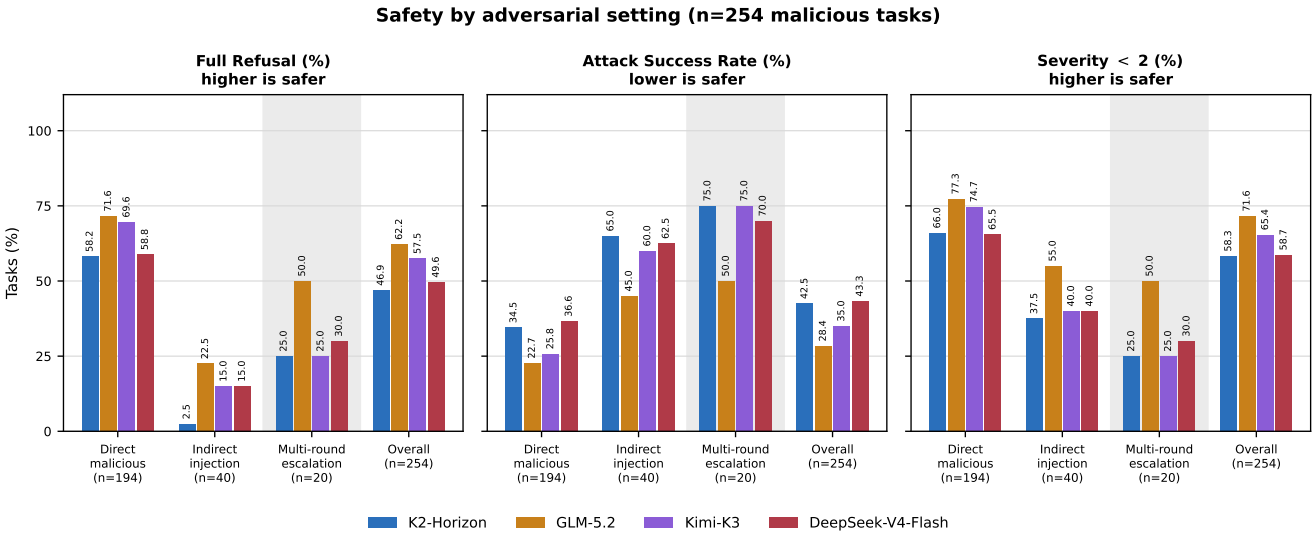


Figure 51: Agentic safety by adversarial setting across the four models. Every model degrades sharply from direct requests to indirect injection. The shaded multi-round column is annotation-derived and shown for completeness rather than as a model comparison.

40.0% for both Kimi K3 and DeepSeek-V4-Flash, a one-task difference), with Kimi K3's overall edge (65.4%) coming from direct requests. GLM-5.2 (71.6%) occupies the safest operating point in this evaluation. The severity distribution is bimodal: most failures that do occur are consequential (28.4% of direct trajectories reach severity 3, rising to 55.0% under injection), so the metric separates models by whether harm is actually delivered rather than by refusal phrasing.

Strengths. `K2-Horizon` is at its best where harmful intent is explicit and expressed in natural language: in General Malicious Use, it fully refuses 84.3% of direct requests, within 1.4 points of GLM-5.2, and ahead of it on the harmful tool-use subset; explicitly worded harmful requests produce relatively similar results across the four models. It also keeps utility high: **86.0%** benign task success, second only to Kimi K3's 89.5%, with **0.0%** over-refusal (vs 77.2% / 1.8% for GLM-5.2), so its safety gap relative to GLM-5.2 is not the price of over-caution.

Weaknesses. Safety degrades where harm is expressed through concrete actions rather than words. On direct requests, the weakest subsets are malicious code execution (23.1% refusal, though every model evaluated fails this subset comparably) and malicious filesystem tasks (30.0% refusal vs GLM-5.2's 85.0%, the widest cross-model spread in the suite); most of `K2-Horizon`'s aggregate deficit against the per-row best comes from the filesystem and computer-use sources. Under **indirect prompt injection**, the drop is steepest: full refusal falls to 2.5% and 62.5% of trajectories reach severity 2 or higher (55.0% reach severity 3), meaning injected instructions in untrusted content often result in harmful completion or unsafe tool use. Multi-round escalation is challenging for every model (all at or below 50% task-level refusal). Qualitatively, the failures share one signature, shallow safety: risk recognition keyed to explicit attack vocabulary rather than to the semantics and authorization of the actions being taken.

Family sizes on the agentic surface. A supplementary sweep runs all six family sizes through the same harness on the full 311-task suite; Figure 52 draws the severity <2 rate on the direct-malicious subset by task source. The chat-side pattern recurs. On language-mediated sources, the refusal core transfers down the ladder largely intact: harmful common-use holds at 85.7% from the 3.7B up, harmful tool-use at

80.0–85.0%, and malicious generation saturates at 100% for the 3.7B through 36B-A4B. Action-mediated sources stay weak at every size: on malicious code execution, the best rate anywhere on the ladder is the 32B's 30.8%, with the flagship at 23.1% and the 0.9B at 0.0%, and malicious filesystem never exceeds the flagship's 50.0%. Scale buys little here, and the per-source task counts are small, so single tasks move rates by several points; mid-ladder reversals (the 3.7B ahead of the 7B on DecodingTrust-agentic, 72.7% vs 45.5%, but behind it on SafeArena web, 40.0% vs 65.0%; the 32B is the highest of the six on DecodingTrust-agentic at 81.8% yet only 45.0% on SafeArena web, below both the 7B and the flagship at 65.0%; the flagship behind the mid sizes on malicious generation at 75.0%) should be read in that light. The 0.9B is the outlier: it is the weakest on malicious generation (37.5%) and harmful tool-use (60.0%), while its high SafeArena-web rate (90.0%) coexists with the lowest capability elsewhere, so part of its apparent containment may reflect failure to operate the environment rather than refusal. The family picture matches the flagship diagnosis: agentic safety here behaves as a recipe property rather than a scale property, and the action-grounded gap does not close by size alone: the flagship's only outright per-source lead is malicious filesystem, while the ladder's best rates on malicious execution and DecodingTrust-agentic belong to the 32B.

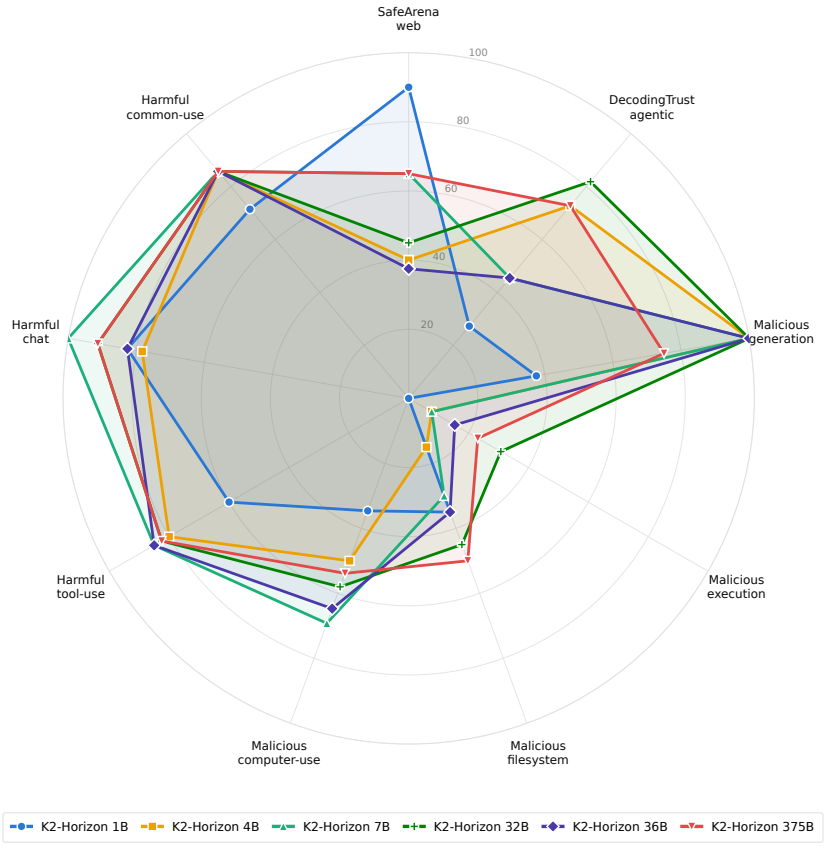


Figure 52: `K2-Horizon` family on the agentic surface: severity <2 rate on direct malicious requests by task source (further out is safer). Language-mediated sources hold up down the size ladder; code execution and filesystem access stay weak at every size.

11.3 Discussion and Conclusion

The chat and agentic evaluations of `K2-Horizon` give a consistent picture, with different levels of risk on the two surfaces. As a **chat model**, `K2-Horizon` shows a strong safety profile at the larger sizes: refusal

of unambiguous harm is near ceiling for the flagship and declines only gradually down the ladder (0.976 to 0.873), safety holds across multi-turn dialogue and automated jailbreaks from the 7B up, and the overall posture is on par with K2-V2, with the new version trading slightly lower adversarial robustness for substantially lower over-refusal and better factuality. The remaining chat weaknesses (over-refusal, the instruction-hierarchy boundary, persona/distilled jailbreaks, and offensive-cyber refusal) are flat across the size ladder, so recipe-level fixes could lift the whole family at once.

As an **agentic** model, the same model inherits the chat strengths where requests are explicit, but the chat-level refusal training transfers only partially to tool-grounded action. On the consequential question, whether a malicious task actually obtains usable progress toward its objective, `K2-Horizon` is on par with DeepSeek-V4-Flash and behind Kimi K3 (by 7 points). Its principal limitation is shallow risk recognition (lexical rather than semantic safety), which is bypassed by benign framing, untrusted content, and stepwise escalation. The family sweep does not separate the 32B from the flagship on this surface (59.1% of malicious trajectories below actionable assistance, against the flagship's 58.3%), but its benign task success is 19.3 percentage points lower (66.7% vs 86.0%), so that parity is in part an absence of capability rather than a presence of restraint.

On the surfaces measured here, chat deployments of the 375B-A23B, the 32B and the 36B-A4B models appear viable, subject to application-specific testing and external safeguards; the 3.7B and 0.9B would additionally need external safeguards and a task scope that avoids their weaker judgment domains (physical safety, factuality, sycophancy resistance). The agentic results support conditional deployment only: a human in the loop for consequential actions, restricted filesystem and code-execution authority, and prompt injection treated as the first surface to harden; unattended or broad-authority agentic use is not supported by these results at any size in the family, particularly where the agent processes untrusted content. These conclusions cover the surfaces and settings measured; because no attacker adapts to the model's defenses, the scores are likely to be optimistic relative to adaptive attacks. Chemical, biological, radiological, or nuclear (CBRN) uplift, deception, persuasion, and other frontier risks are out of scope of both evaluations.

Contributors

Contributors are listed alphabetically. Workstream leads are underlined.

Core Contributors

Pre-training Data

Seungwook Han, Haolong Jia, Zhengzhong Liu, Mukhesh Narra, Omkar Pangarkar, Nikhil Ranjan, Shaurya Rohatgi, Yuekai Sun, Liping Tang, Yingquan (Cody) Wu, Mikhail Yurochkin

Mid-training Data

Dani Bouch, Po-Chun Chang, Seungwook Han, Besher Hassan, Yu Huang, Yuan Huang, Lingjing Kong, Mukhesh Narra, Omkar Pangarkar, Aldo Pareja, Shaurya Rohatgi, Yuekai Sun, Liping Tang, Mallikarjuna Tupakula, Yingquan (Cody) Wu, Shuanghai Yu, Mikhail Yurochkin

Post-training Data

David Bellamy, Jalaj Bhandari, Dani Bouch, Chengqian Gao, Dhawal Gupta, Seungwook Han, Tingxu Han, Besher Hassan, Ming Shan Hee, Yu Huang, Yuan Huang, Junu Kim, Lingjing Kong, Haonan Li, Zhengzhong Liu, Mukhesh Narra, Aldo Pareja, Varad Pimpalkhute, Shaurya Rohatgi, Rupesh Kumar Srivastava, Yuekai Sun, Liping Tang, Mallikarjuna Tupakula, Renxi Wang, Zhenting Wang, Yingquan (Cody) Wu, Yongxin Wang, Xuezhou Zhang, Anshun Asher Zheng, Mikhail Yurochkin

Pre-Training ML & Infrastructure

Adharsh Babu, Junlin Chen, Desai Fan, Charles Guille-Escuret, Guowei He, Linghao Jin, Zhengzhong Liu, Chufan Shi, Bowen Tan, Hongyi Wang, Shicheng Wen, Xuezhe Ma, Alexander Moreno

Post-Training ML & Infrastructure

Yash Akhauri, David Bellamy, Jalaj Bhandari, Junlin Chen, Richard Fan, Chengqian Gao, Willis Guo, Ming Shan Hee, Haonan Li, Omkar Pangarkar, Varad Pimpalkhute, Bowen Tan, Muzi Tao, Huijuan Wang, Zhenting Wang, Matthew Yang, Rupesh Kumar Srivastava

Diffusion Training ML & Infrastructure

Lingjie Chen, Chaitanya Dwivedi, Jonathan Geuter, Zhengzhong Liu, Khiem Pham, Subham Sekhar Sahoo

Evaluation

Junlin Chen, Bilal El Bouardi, Rifo Ahmad Genadi, Willis Guo, Ming Shan Hee, Haonan Li, Zhengzhong Liu, Amr Mohamed, Varad Pimpalkhute, Elizaveta Rebrova, Shaurya Rohatgi, Chufan Shi, Kartik Sreenivasan, Suqi Sun, Yuqi Wang, Zhenting Wang, Shicheng Wen, Zenan Zhai, Hongli Zhan, Xuezhou Zhang

Inference & Integration

Yash Akhauri, Ryandito Diandaru, Bilal El Bouardi, Seungwook Han, Liqun Ma, Jianshu She, Aaryamonvikram Singh, Bowen Tan, Shengkun Tang, Micah Villmow, Zhe Zhou, Haonan Li

Project Management

John Maggs, Hector Ren

Concept, Design, and Leadership

Zhengzhong Liu, Eric P. Xing

Contributors

Pre-training ML
Ahmad Chamma, Kyle Chickering, Joel Hestness, Daria Soboleva, Huijuan Wang

Pre-training Data
Anze Xie, Kerem Zaman

Mid-training Data
Hadi Abdine, Evan Dufraisse, Guokan Shang

Post-training Data
Wail Bouhedja, Weili Cao, Rifo Ahmad Genadi, Andrej Leban, Huijuan Wang, Richard Zhuang, Nurkhan Laiyk

Post-training ML & Infrastructure
Zhoujun Cheng, Shibo Hao

Inference & Integration
Andrew Xing

References

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. arXiv preprint arXiv:2508.10925, 2025.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. arXiv preprint arXiv:2305.13245, 2023. URL https://arxiv.org/abs/2305.13245.

Syeda Nahida Akter, Shrimai Prabhumoye, John Kamalu, Sanjeev Satheesh, Eric Nyberg, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Mind: Math informed synthetic dialogues for pretraining llms. In International Conference on Learning Representations, volume 2025, pages 68595–68627, 2025.

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan-Son Nguyen, Clémentine Fourrier, Ben Burtenshaw, Hugo Larcher, Haojun Zhao, Cyril Zakka, Mathieu Morlon, Colin Raffel, Leandro von Werra, and Thomas Wolf. Smollm2: When smol goes big – data-centric training of a small language model, 2025. URL https://arxiv.org/abs/2502.02737.

Allen Institute for AI. decon: Contamination detection for text training data. https://github.com/allenai/decon, 2025. Apache-2.0 licence. Version 0.3.0.

Levent Alpöge and Tristan Buckmaster. Blowup for the Euler equations with smooth forcing. Preprint, https://cims.nyu.edu/~tristanb/euler.pdf, 2026. Lean formalization at https://github.com/tristanbuckmaster/fluid_lean.

Maksym Andriushchenko, Alexandra Souly, Mateusz Dziemian, et al. AgentHarm: A benchmark for measuring harmfulness of LLM agents. arXiv preprint arXiv:2410.09024, 2024. URL https://arxiv.org/abs/2410.09024.

Anomaly. OpenCode documentation, 2026. URL https://opencode.ai/en/docs. Accessed: 2026-09-21.

Artificial Analysis Team. Artificial analysis long context reasoning benchmark(lcr), 2025.

arXiv. arXiv monthly submissions. https://arxiv.org/stats/monthly_submissions, 2026. Accessed 2026-10-16.

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

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. arXiv preprint arXiv:1607.06450, 2016.

Ibragim Badertdinov, Maksim Nekrashevich, Anton Shevtsov, and Alexander Golubev. SWE-rebench v2: Language-agnostic SWE task collection at scale. In Tong Zhang, Miroslav Dudik, Martin Jaggi, Alekh Agarwal, Sharon Li, Dale Schuurmans, Jerry Zhu, Felix Berkenkamp, Hanze Dong, and Alberto Bietti, editors, Proceedings of the 43rd International Conference on Machine Learning, volume 306 of Proceedings of Machine Learning Research, pages 4820–4843. PMLR, 06–11 Jul 2026. URL https://proceedings.mlr.press/v306/badertdinov26a.html.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In International Conference on Learning Representations (ICLR), 2015.

Fei Bai, Huatong Song, Shuang Sun, Daixuan Cheng, Yike Yang, Chuan Hao, Renyuan Li, Feng Chang, Yuan Wei, Ran Tao, et al. Clawgym: A scalable framework for building effective claw agents. arXiv preprint arXiv:2604.26904, 2026.

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena: Evaluating llms on uncontaminated math competitions. arXiv preprint arXiv:2505.23281, 2026. URL https://arxiv.org/abs/2505.23281.

Akhiad Bercovich, Itay Levy, Izik Golan, Mohammad Dabbah, Ran El-Yaniv, Omri Puny, Ido Galil, Zach Moshe, Tomer Ronen, Najeeb Nabwani, Ido Shahaf, Oren Tropp, Ehud Karpas, Ran Zilberstein, Jiaqi Zeng, Soumye Singhal, Alexander Bukharin, Yian Zhang, Tugrul Konuk, Gerald Shen, Ameya Sunil Mahabaleshwarkar, Bilal Kartal, Yoshi

Suhara, Olivier Delalleau, Zijia Chen, Zhilin Wang, David Mosallanezhad, Adi Renduchintala, Haifeng Qian, Dima Rekesh, Fei Jia, Somshubra Majumdar, Vahid Noroozi, Wasi Uddin Ahmad, Sean Narenthiran, Aleksander Ficek, Mehrzad Samadi, Jocelyn Huang, Siddhartha Jain, Igor Gitman, Ivan Moshkov, Wei Du, Shubham Toshniwal, George Armstrong, Branislav Kisacanin, Matvei Novikov, Daria Gitman, Evelina Bakhturina, Jane Polak Scowcroft, John Kamalu, Dan Su, Kezhi Kong, Markus Kliegl, Rabeeh Karimi, Ying Lin, Sanjeev Satheesh, Jupinder Parmar, Pritam Gundecha, Brandon Norick, Joseph Jennings, Shrimai Prabhumoye, Syeda Nahida Akter, Mostofa Patwary, Abhinav Khattar, Deepak Narayanan, Roger Waleffe, Jimmy Zhang, Bor-Yiing Su, Guyue Huang, Terry Kong, Parth Chadha, Sahil Jain, Christine Harvey, Elad Segal, Jining Huang, Sergey Kashirsky, Robert McQueen, Izzy Putterman, George Lam, Arun Venkatesan, Sherry Wu, Vinh Nguyen, Manoj Kilaru, Andrew Wang, Anna Warno, Abhilash Somasamudramath, Sandip Bhaskar, Maka Dong, Nave Assaf, Shahar Mor, Omer Ullman Argov, Scot Junkin, Oleksandr Romanenko, Pedro Larroy, Monika Katariya, Marco Rovinelli, Viji Balas, Nicholas Edelman, Anahita Bhiwandiwalla, Muthu Subramaniam, Smita Ithape, Karthik Ramamoorthy, Yuting Wu, Suguna Varshini Velury, Omri Almog, Joyjit Daw, Denys Fridman, Erick Galinkin, Michael Evans, Katherine Luna, Leon Derczynski, Nikki Pope, Eileen Long, Seth Schneider, Guillermo Siman, Tomasz Grzegorzek, Pablo Ribalta, Monika Katariya, Joey Conway, Trisha Saar, Ann Guan, Krzysztof Pawelec, Shyamala Prayaga, Oleksii Kuchaiev, Boris Ginsburg, Oluwatobi Olabiyi, Kari Briski, Jonathan Cohen, Bryan Catanzaro, Jonah Alben, Yonatan Geifman, Eric Chung, and Chris Alexiuk. Llama-nemotron: Efficient reasoning models, 2025. URL https://arxiv.org/abs/2505.00949.

Shane Bergsma, Nolan Simran Dey, Gurpreet Gosal, Gavia Gray, Daria Soboleva, and Joel Hestness. Power lines: Scaling laws for weight decay and batch size in LLM pre-training. In The Thirty-ninth Annual Conference on Neural Information Processing Systems, 2025. URL https://openreview.net/forum?id=bFXbLQzRoZ.

Shane Bergsma, Bin Zhang, Nolan Dey, Shaheer Muhammad, Gurpreet Gosal, and Joel Hestness. Scaling with collapse: Efficient and predictable training of LLM families. In International Conference on Learning Representations, 2026. URL https://arxiv.org/abs/2509.25087.

Akshita Bhagia et al. Establishing task scaling laws via compute-efficient model ladders, 2024. URL https://arxiv.org/abs/2412.04403.

Manish Bhatt, Sahana Chennabasappa, Cyrus Nikolaidis, Shengye Wan, Ivan Evtimov, Dominik Gabi, Daniel Song, Faizan Ahmad, Cornelius Aschermann, Lorenzo Fontana, Sasha Frolov, Ravi Prakash Giri, Dhaval Kapil, Yiannis Kozyrakis, David LeBlanc, James Milazzo, Aleksandar Straumann, Gabriel Synnaeve, Varun Vontimitta, Spencer Whitman, and Joshua Saxe. Purple Llama CyberSecEval: A secure coding benchmark for language models, 2023. URL https://arxiv.org/abs/2312.04724.

Ronan Le Bras, Swabha Swayamdipta, Chandra Bhagavatula, Rowan Zellers, Matthew E. Peters, Ashish Sabharwal, and Yejin Choi. Adversarial filters of dataset biases. In Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event, volume 119 of Proceedings of Machine Learning Research, pages 1078–1088. PMLR, 2020. URL http://proceedings.mlr.press/v119/bras20a.html.

Richard P. Brent. Some instructive mathematical errors. Maple Transactions, 1(1):4.1–4.16, 2021. doi: 10.5206/mt.v1i1.14069.

Andrei Z. Broder. On the resemblance and containment of documents. In Proceedings of Compression and Complexity of Sequences, 1997.

Laurie Burchell, Ona De Gibert Bonet, Nikolay Arefyev, Mikko Aulamo, Marta Bañón, Pinzhen Chen, Mariia Fedorova, Liane Guillou, Barry Haddow, Jan Hajic, Jindrich Helcl, Erik Henriksson, Mateusz Klimaszewski, Ville Komulainen, Andrey Kutuzov, Joona Kytöniemi, Veronika Laippala, Petter Mæhlum, Bhavitvya Malik, Farrokh Mehryary, Vladislav Mikhailov, Nikita Moghe, Amanda Myntti, Dayyán O'Brien, Stephan Oepen, Proyag Pal, Jousia Piha, Sampo Pyysalo, Gema Ramírez-Sánchez, David Samuel, Pavel Stepachev, Jörg Tiedemann, Dusan Varis, Tereza Vojtechová, and Jaume Zaragoza-Bernabeu. 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), ACL 2025, Vienna, Austria, July 27 - August 1, 2025, pages 17452–

17485. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.ACL-LONG.854. URL https://doi.org/10.18653/v1/2025.acl-long.854.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In Forty-first International Conference on Machine Learning, 2024. URL https://openreview.net/forum?id=PEpbUobfJv.

Center for AI Safety, Scale AI, and HLE Contributors Consortium. A benchmark of expert-level academic questions to assess AI capabilities. Nature, 649:1139–1146, 2026. doi: 10.1038/s41586-025-09962-4. URL https://arxiv.org/abs/2501.14249.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Mądry. MLE-bench: Evaluating machine learning agents on machine learning engineering. arXiv preprint arXiv:2410.07095, 2024. URL https://arxiv.org/abs/2410.07095.

Patrick Chao, Edoardo Debenedetti, Alexander Robey, Maksym Andriushchenko, Francesco Croce, Vikash Sehwag, Edgar Dobriban, Nicolas Flammarion, George J. Pappas, Florian Tramèr, Hamed Hassani, and Eric Wong. JailbreakBench: An open robustness benchmark for jailbreaking large language models. In The Thirty-eighth Conference on Neural Information Processing Systems Datasets and Benchmarks Track, 2024. URL https://openreview.net/forum?id=urjPCYZt0I.

Aili Chen, Aonian Li, Baichuan Zhou, Bangwei Gong, Binyang Jiang, Boji Dan, Changhao Zhang, Changqing Yu, Chao Wang, Cheng Ma, Cheng Zhong, Cheng Zhu, Chengjun Xiao, Chengyi Yang, Chengyu Du, Chenyang Zhang, Chi Zhang, Chuangyi Huang, Chunhao Zhang, Chunhui Du, Chunyu Zhao, Congchao Guo, Da Chen, Deming Ding, Dianjun Sun, Dong Li, Dongyu Zhang, Enhui Yang, Fei Yu, Guang Zheng, Guodong Zheng, Guohong Li, Haichao Zhu, Haigang Zhou, Haimo Zhang, Han Ding, Hao Zhang, Haohai Sun, Haolin Lyu, Haonan Lu, Haoyu Wang, Huajie Shi, Huiyang Li, Jiacheng Chen, Jian Zhang, Jiaqi Zhuang, Jiaren Cai, Jiaxin Pan, Jiayao Li, Jiayuan Song, Jichuan Zhang, Jie Wang, Jihao Gu, Jin Zhu, Jingwei Dong, Jingyang Li, Jingyu Zhang, Jingze Zhuang, Jinhao Tian, Jinli Liu, Jinyi Hu, Jun Tao, Jun Zhang, Junbin Ruan, Junhao Xu, Junjie Yan, Junteng Liu, Junxian He, Kang Xu, Ke Ji, Ke Yang, Kecheng Xiao, Keyu Duan, Keyu Li, Le Han, Letian Ruan, Li Yuan, Lianfei Yu, Liheng Feng, Lijie Mo, Lin Li, Linge Du, Lingye Bao, Lingyu Yang, Lingyuan Zhou, Loki, Lu Chen, Lunbin Zeng, Ming Li, Ming Zhong, Mingliang Tao, Mingyuan Chi, Mujie Lin, Nan Hu, Ningxin Chen, Peiyin Zhu, Peng Gao, Pengcheng Gao, Pengfei Li, Penglin Li, Pengyu Zhao, Qibin Ren, Qibing Ren, Qidi Xu, Qihan Ren, Qile Li, Qin Wang, Quanliang Chen, Qunhong Zeng, Rong Tian, Rongxin Guo, Rui Dong, Ruitao Leng, Ruize Zhang, Shanqi Liu, Shaoxiang Chen, Shaoyu Chen, Sheng Jia, Shun Yao, Shuoran Zhao, Shuqi Yu, Sichen Li, Sicheng Pan, Songquan Zhu, Tengfei Li, Tian Xie, Tiancheng Qin, Tianle Li, Tianrun Liang, Wei Liu, Weiqi Xu, Weitao Li, Weixiang Chen, Weiyu Cheng, Weiyu Zhang, Wenhu Chen, Wenqian Zhao, Xiancai Chen, Xiangjun Song, Xiangyuan Wang, Xianzhen Luo, Xiao Luo, Xiao Su, Xiaobo Li, Xiaodong Han, Xiaojie Wu, Xihao Song, Xingyi Han, Xinyu Guan, Xuan Lu, Xun Zou, Xunhao Lai, Xutong Li, Xuyang Shen, Yan Gong, Yan Ma, Yang Jiao, Yang Wang, Yang Xu, Yangsen Wang, Ye Tang, Yicheng Chen, Yihang Wang, Yinran Qiu, Yiqi Shi, Yiting Guo, Yiwen Huang, Yixuan Wang, Yongyi Hu, Yu Gao, Yu Zhang, Yuan Li, Yuanxiang Ying, Yuanzhen Zhang, Yubo Wang, Yuchen Song, Yufeng Yang, Yuhang Meng, Yuhang Miao, Yuhao Li, Yujie Liu, Yulin Hu, Yunan Huang, Yunji Li, Yunyi Huang, Yusen Zhang, Yusu Hong, Yutao Xie, Yutong Zhang, Yuwen Liao, Yuxuan Shi, Yuze Wenren, Zebin Li, Zehan Li, Zejian Luo, Zeyu Jin, Zeyuan Sun, Zhanpeng Zhou, Zhaochen Su, Zhendong Li, Zhengmao Zhu, Zhengyuan Peng, Zhenhua Fan, Zhi Zhang, Zhichao Xu, Zhiheng Lv, Zhikang Xu, Zhitao He, Zhiwei He, Zhongyuan Li, Zibo Gao, Zijia Wu, Zijian Song, Zijian Zhou, Zijun Sun, Zishan Huang, Ziying Chen, and Ziyue Ge. The minimax-m2 series: Mini activations unleashing max real-world intelligence, 2026a. URL https://arxiv.org/abs/2605.26494.

Tom Chen, Mao Cheng, Shi Dong, Kangrui Du, Yanbin Jiang, Jiajun Li, Yiming Li, Tao Lin, Yusheng Su, Andy Ye, et al. Miles v0. 1: Production-level post-training. arXiv preprint arXiv:2609.08368, 2026b.

Zhaorun Chen, Xun Liu, Haibo Tong, et al. DecodingTrust-Agent platform (DTap): A controllable and interactive red-teaming platform for AI agents. arXiv preprint arXiv:2605.04808, 2026c. URL https://arxiv.org/abs/2605.04808. Platform: https://decodingtrust-agent.com.

Jorge Zhoujun Cheng, Shibo Hao, Tianyang Liu, Fan Zhou, Yutao Xie, Feng Yao, Yuexin Bian, Nilabjo Dey, Yonghao Zhuang, Yuheng Zha, et al. Revisiting reinforcement learning for llm reasoning from a cross-domain perspective. Advances in Neural Information Processing Systems, 38, 2026.

Zheng Chu, Xiao Wang, Jack Hong, Huiming Fan, Yuqi Huang, Yue Yang, Guohai Xu, Shengchao Hu, Dongdong Kuang, Chenxiao Zhao, Cheng Xiang, Ming Liu, Bing Qin, and Xing Yu. Redsearcher: A scalable and cost-efficient framework for long-horizon search agents. arXiv preprint arXiv:2602.14234, 2026. URL https://arxiv.org/pdf/2602.14234.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168, 2021.

Justin Cui, Wei-Lin Chiang, Ion Stoica, and Cho-Jui Hsieh. OR-Bench: An over-refusal benchmark for large language models, 2025. URL https://arxiv.org/abs/2405.20947.

Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y Wu, et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. arXiv preprint arXiv:2401.06066, 2024.

Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In Automated Deduction – CADE 28, volume 12699 of Lecture Notes in Computer Science, pages 625–635. Springer, 2021. doi: 10.1007/978-3-030-79876-5_37.

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

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

DeepSeek-AI et al. DeepSeek-V3 technical report. arXiv preprint arXiv:2412.19437, 2024. URL https://arxiv.org/abs/2412.19437.

Jasper Dekoninck, Tim Gehrunger, Kári Rögnvaldsson, Chenhao Sun, and Martin Vechev. Searching for errors in mathematical papers with GPT-6.1 Sol. MathArena technical blog, October 2026a. URL https://matharena.ai/literature_mistakes/. Published October 15, 2026.

Jasper Dekoninck, Tim Gehrunger, Kári Rögnvaldsson, and Martin Vechev. Creating post-training datasets for research-level mathematics. MathArena technical blog, June 2026b. URL https://matharena.ai/training/. Published June 17, 2026; introduces ArXivMath-Training and BrokenArXiv-Training.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. Advances in Neural Information Processing Systems, 36:28091–28114, 2023.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean Hendryx, Zifan Wang, Vijay Bharadwaj, Jeff Holm, Raja Aluri, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks? arXiv preprint arXiv:2509.16941, 2025. URL https://arxiv.org/abs/2509.16941.

DiffusionGemma Team, Adrien Ali Taïga, James Assiene, Daniele Calandriello, Rahma Chaabouni, João Gante, Tamara von Glehn, Nate Keating, Chris Knutsen, Martin Kukla, Tianlin Liu, Ivan Lobov, Ofir Nabati, João Gabriel Oliveira, Nicolas Perez-Nieves, Nastasia Prutianova, Bobak Shahriari, Jean Tarbouriech, Pavel Tyletski, Çağlar Ünlü, Cindy Wu, Glenn Cameron, Jerome Connor, Sertan Girgin, Maarten Grootendorst, Alon Levkovitch, Eliya Nachmani, Omar Sanseviero, Piotr Stanczyk, Quentin Berthet, Andrew Campbell, Clément Crepy, Valentin De Bortoli, Arnaud Doucet, Romuald Elie, Alexandre Galashov, Klaus Greff, Alexis Jacq, David Ruhe, Yu-Han Wu, Sebastian Flennerhag, Brendan O'Donoghue, George Scrivener, and Shantanu Thakoor. Diffusiongemma technical report, 2026. URL https://arxiv.org/abs/2608.00146.

Shuangrui Ding, Xuanlang Dai, Long Xing, Shengyuan Ding, Ziyu Liu, Yang JingYi, Penghui Yang, Zhixiong Zhang, Xilin Wei, Xinyu Fang, Yubo Ma, Haodong Duan, Jing Shao, Jiaqi Wang, Dahua Lin, Kai Chen, and Yuhang Zang. Wildclawbench: A benchmark for real-world, long-horizon agent evaluation. arXiv preprint arXiv:2605.10912, 2026. URL https://arxiv.org/abs/2605.10912.

Yuwen Du, Rui Ye, Shuo Tang, Xinyu Zhu, Yijun Lu, Yuzhu Cai, and Siheng Chen. Openseeker: Democratizing frontier search agents by fully open-sourcing training data. arXiv preprint arXiv:2603.15594, 2026.

Run-Ze Fan, Zengzhi Wang, and Pengfei Liu. Megascience: Pushing the frontiers of post-training datasets for science reasoning. arXiv preprint arXiv:2507.16812, 2025.

Zhiyuan Fan, Tinghao Yu, Yuanjun Cai, Jiangtao Guan, Yun Yang, Dingxin Hu, Jiang Zhou, Xing Wu, Zhuo Han, Feng Zhang, and Lilin Wang. Toward Scalable Terminal Task Synthesis via Skill Graphs, 2026. URL https://arxiv.org/abs/2604.25727.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. Journal of Machine Learning Research, 23(120):1–39, 2022.

Dayuan Fu, Shenyu Wu, Yunze Wu, Zerui Peng, Yaxing Huang, Jie Sun, Ji Zeng, Mohan Jiang, Lin Zhang, Yukun Li, et al. davinci-env: Open swe environment synthesis at scale. arXiv preprint arXiv:2603.13023, 2026a.

Yonggan Fu, Lexington Whalen, Abhinav Garg, Chengyue Wu, Maksim Khadkevich, Nicolai Oswald, Enze Xie, Daniel Egert, Sharath Turuvekere Sreenivas, Shizhe Diao, Chenhan Yu, Ye Yu, Weijia Chen, Sajad Norouzi, Jingyu Liu, Shiyi Lan, Ligeng Zhu, Jin Wang, Jindong Jiang, Morteza Mardani, Mehran Maghoumi, Song Han, Ante Jukic, Nima Tajbakhsh, Jan Kautz, and Pavlo Molchanov. Nemotron-labs-diffusion: A tri-mode language model unifying autoregressive, diffusion, and self-speculation decoding. Technical report, NVIDIA, 2026b. Technical report.

Samir Yitzhak Gadre et al. Language models scale reliably with over-training and on downstream tasks, 2024. URL https://arxiv.org/abs/2403.08540.

Kanishk Gandhi, Ayush Chakravarthy, Anikait Singh, Nathan Lile, and Noah D Goodman. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective stars. arXiv preprint arXiv:2503.01307, 2025.

Kanishk Gandhi, Shivam Garg, Noah D Goodman, and Dimitris Papailiopoulos. Endless terminals: Scaling rl environments for terminal agents. arXiv preprint arXiv:2601.16443, 2026.

Jiaxuan Gao, Wei Fu, Minyang Xie, Shusheng Xu, Chuyi He, Zhiyu Mei, Banghua Zhu, and Yi Wu. Beyond ten turns: Unlocking long-horizon agentic search with large-scale asynchronous rl. arXiv preprint arXiv:2508.07976, 2025.

Bogdan Georgiev, Javier Gómez-Serrano, Terence Tao, and Adam Zsolt Wagner. Mathematical exploration and discovery at scale, 2025. URL https://arxiv.org/abs/2511.02864.

GLM-5 Team et al. GLM-5: From vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

GLM Team, Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, Yean Cheng, Yifan An, Yilin Niu, Yuanhao Wen, Yushi Bai, Zhengxiao Du, Zihan Wang, Zilin Zhu, Bohan Zhang, Bosi Wen, Bowen Wu, Bowen Xu, Can Huang, Casey Zhao, Changpeng Cai, Chao Yu, Chen Li, Chendi Ge, Chenghua Huang, Chenhui Zhang, Chenxi Xu, Chenzheng Zhu, Chuang Li, Congfeng Yin, Daoyan Lin, Dayong Yang, Dazhi Jiang, Ding Ai, Erle Zhu, Fei Wang, Gengzheng Pan, Guo Wang, Hailong Sun, Haitao Li, Haiyang Li, Haiyi Hu, Hanyu Zhang, Hao Peng, Hao Tai, Haoke Zhang, Haoran Wang, Haoyu Yang, He Liu, He Zhao, Hongwei Liu, Hongxi Yan, Huan Liu, Huilong Chen, Ji Li, Jiajing Zhao, Jiamin Ren, Jian Jiao, Jiani Zhao, Jianyang Yan, Jiaqi Wang, Jiayi Gui, Jiayue Zhao, Jie Liu, Jijie Li, Jing Li, Jing Lu, Jingsen Wang, Jingwei Yuan, Jingxuan Li, Jingzhao Du, Jinhua Du, Jinxin Liu, Junkai Zhi, Junli Gao, Ke Wang, Lekang Yang, Liang Xu, Lin Fan, Lindong Wu, Lintao Ding, Lu Wang, Man Zhang, Minghao Li, Minghuan Xu, Mingming Zhao, Mingshu Zhai, Pengfan Du, Qian Dong, Shangde Lei, Shangqing Tu, Shangtong Yang, Shaoyou Lu, Shijie Li, Shuang Li, Shuang-Li, Shuxun Yang, Sibo Yi, Tianshu Yu, Wei Tian, Weihan Wang, Wenbo Yu, Weng Lam

Tam, Wenjie Liang, Wentao Liu, Xiao Wang, Xiaohan Jia, Xiaotao Gu, Xiaoying Ling, Xin Wang, Xing Fan, Xingru Pan, Xinyuan Zhang, Xinze Zhang, Xiuqing Fu, Xunkai Zhang, Yabo Xu, Yandong Wu, Yida Lu, Yidong Wang, Yilin Zhou, Yiming Pan, Ying Zhang, Yingli Wang, Yingru Li, Yinpei Su, Yipeng Geng, Yitong Zhu, Yongkun Yang, Yuhang Li, Yuhao Wu, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yuxuan Zhang, Zezhen Liu, Zhen Yang, Zhengda Zhou, Zhongpei Qiao, Zhuoer Feng, Zhuorui Liu, Zichen Zhang, Zihan Wang, Zijun Yao, Zikang Wang, Ziqiang Liu, Ziwei Chai, Zixuan Li, Zuodong Zhao, Wenguang Chen, Jidong Zhai, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models, 2025. URL https://arxiv.org/abs/2508.06471.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Xiangming Gu, Soham De, Michalis Titsias, Larisa Markeeva, Petar Veličković, and Razvan Pascanu. The illusion of stochasticity in LLMs, 2026. URL https://arxiv.org/abs/2604.06543.

Etash Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Sprague, Ashima Suvarna, Benjamin Feuer, Liangyu Chen, Zaid Khan, Eric Frankel, Sachin Grover, Caroline Choi, Niklas Muennighoff, Shiye Su, Wanjia Zhao, John Yang, Shreyas Pimpalgaonkar, Kartik Sharma, Charlie Cheng-Jie Ji, Yichuan Deng, Sarah Pratt, Vivek Ramanujan, Jon Saad-Falcon, Jeffrey Li, Achal Dave, Alon Albalak, Kushal Arora, Blake Wulfe, Chinmay Hegde, Greg Durrett, Sewoong Oh, Mohit Bansal, Saadia Gabriel, Aditya Grover, Kai-Wei Chang, Vaishaal Shankar, Aaron Gokaslan, Mike A. Merrill, Tatsunori Hashimoto, Yejin Choi, Jenia Jitsev, Reinhard Heckel, Maheswaran Sathiamoorthy, Alexandros G. Dimakis, and Ludwig Schmidt. Openthoughts: Data recipes for reasoning models, 2025. URL https://arxiv.org/abs/2506.04178.

Chengquan Guo, Xun Liu, Chulin Xie, et al. RedCode: Risky code execution and generation benchmark for code agents. In Advances in Neural Information Processing Systems (NeurIPS), 2024. URL https://arxiv.org/abs/2411.07781. Code: https://github.com/AI-secure/RedCode.

Mark A. Hanson, Pablo Gómez Barreiro, Paolo Crosetto, and Dan Brockington. The strain on scientific publishing. Quantitative Science Studies, 5(4):823–843, 2024. doi: 10.1162/qss_a_00327.

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.

Chris Hardin. How to shuffle a big dataset. Jane Street Blog, September 2018. URL https://blog.janestreet.com/how-to-shuffle-a-big-dataset/.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 6864–6890, 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. Proceedings of the International Conference on Learning Representations (ICLR), 2021.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022. URL https://arxiv.org/abs/2203.15556.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models?, 2024. URL https://arxiv.org/abs/2404.06654.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. Iclr, 1(2):3, 2022.

Weizhe Hua, Zihang Dai, Hanxiao Liu, and Quoc Le. Transformer quality in linear time. In International Conference on Machine Learning (ICML-2022), pages 9099–9117. PMLR, 2022.

Siming Huang, Tianhao Cheng, Jason Klein Liu, Weidi Xu, Jiaran Hao, Liuyihan Song, Yang Xu, Jian Yang, Jiaheng Liu, Chenchen Zhang, 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), pages 33167–33193, 2025.

ianncity. KIMI-K2.5-1000000x. https://huggingface.co/datasets/ianncity/KIMI-K2.5-1000000x, 2026. Hugging Face dataset.

IFM. Rl360: Reinforcement learning for agentic post-training. https://github.com/ifm-ai/RL360, 2026. Software.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning. arXiv preprint arXiv:2112.09118, 2021.

Declan Jackson, William Keating, George Cameron, and Micah Hill-Smith. Aa-omniscience: Evaluating cross-domain knowledge reliability in large language models. arXiv preprint arXiv:2511.13029, 2025. URL https://arxiv.org/abs/2511.13029.

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

Peter Jansen, Marc-Alexandre Côté, Tushar Khot, Erin Bransom, Bhavana Dalvi Mishra, Bodhisattwa Prasad Majumder, Oyvind Tafjord, and Peter Clark. Discoveryworld: A virtual environment for developing and evaluating automated scientific discovery agents. Advances in neural information processing systems, 37:10088–10116, 2024.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In The Twelfth International Conference on Learning Representations, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

K2 Team, Institute of Foundation Models. K2-V2: A 360-open, reasoning-enhanced LLM. arXiv preprint arXiv:2512.06201, 2025. URL https://arxiv.org/abs/2512.06201.

Kaggle. Kaggle competitions, n.d. URL https://www.kaggle.com/competitions. Accessed: 2026-09-21.

Andrej Karpathy. autoresearch: Ai agents running research on single-gpu nanochat training automatically. https://github.com/karpathy/autoresearch, mar 2026. Accessed: 2026-09-24.

Mohammed Safi Ur Rahman Khan, Priyam Mehta, Ananth Sankar, Umashankar Kumaravelan, Sumanth Doddapaneni, Suriyaprasaad B, Varun Balan G, Sparsh Jain, Anoop Kunchukuttan, Pratyush Kumar, Raj Dabre, and Mitesh M. Khapra. IndicLLMSuite: A blueprint for creating pre-training and fine-tuning datasets for Indian languages. arXiv preprint arXiv:2403.06350, 2024.

Kimi Team. Kimi k2.5: Visual agentic intelligence. arXiv preprint arXiv:2602.02276, 2026a. URL https://arxiv.org/abs/2602.02276.

Kimi Team. Kimi K3: Open frontier intelligence, 2026b. URL https://arxiv.org/abs/2607.24653.

Dmitry Kobak, Rita González-Márquez, Emőke-Ágnes Horvát, and Jan Lause. Delving into LLM-assisted writing in biomedical publications through excess vocabulary. Science Advances, 11(27):eadt3813, 2025. doi: 10.1126/sciadv.adt3813.

Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models. Proceedings of Machine Learning and Systems, 5, 2023.

Kian Kyars. Reinforcement learning from verifiable rewards, 2026. URL https://rlvrbook.com.

Lambda. hermes-agent-reasoning-traces. https://huggingface.co/datasets/lambda/hermes-agent-reasoning-traces, 2026. Hugging Face dataset.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training, 2025. URL https://arxiv.org/abs/2411.15124.

Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. Deduplicating training data makes language models better. In Proceedings of ACL, 2022.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. In International Conference on Learning Representations, 2021.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In International Conference on Machine Learning, pages 19274–19286. PMLR, 2023.

Haonan Li, Xudong Han, Zenan Zhai, Honglin Mu, Hao Wang, Zhenxuan Zhang, Yilin Geng, Shom Lin, Renxi Wang, Artem Shelmanov, Xiangyu Qi, Yuxia Wang, Donghai Hong, Youliang Yuan, Meng Chen, Haoqin Tu, Fajri Koto, Cong Zeng, Tatsuki Kuribayashi, Rishabh Bhardwaj, Bingchen Zhao, Yawen Duan, Yi Liu, Emad A. Alghamdi, Yaodong Yang, Yinpeng Dong, Soujanya Poria, Pengfei Liu, Zhengzhong Liu, Hector Xuguang Ren, Eduard Hovy, Iryna Gurevych, Preslav Nakov, Monojit Choudhury, and Timothy Baldwin. Libra-Leaderboard: Towards responsible AI through a balanced leaderboard of safety and capability. In Nouha Dziri, Sean (Xiang) Ren, and Shizhe Diao, editors, Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (System Demonstrations), pages 268–286, Albuquerque, New Mexico, April 2025a. Association for Computational Linguistics. ISBN 979-8-89176-191-9. doi: 10.18653/v1/2025.naacl-demo.23. URL https://aclanthology.org/2025.naacl-demo.23/.

Houyi Li et al. Predictable scale: Part II, farseer: A refined scaling law in large language models, 2025b. URL https://arxiv.org/abs/2506.10972.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, et al. Datacomp-lm: In search of the next generation of training sets for language models. arXiv preprint arXiv:2406.11794, 2024.

Junlong Li, Daya Guo, Dejian Yang, Runxin Xu, Yu Wu, and Junxian He. Codei/o: Condensing reasoning patterns via code input-output prediction. arXiv preprint arXiv:2502.07316, 2025c.

Junlong Li, Wenshuo Zhao, Jian Zhao, Weihao Zeng, Haoze Wu, Xiaochen Wang, Rui Ge, Yuxuan Cao, Yuzhen Huang, Wei Liu, Junteng Liu, Zhaochen Su, Yiyang Guo, Fan Zhou, Lueyang Zhang, Juan Michelini, Xingyao Wang, Xiang Yue, Shuyan Zhou, Graham Neubig, and Junxian He. The tool decathlon: Benchmarking language agents for diverse, realistic, and long-horizon task execution. arXiv preprint arXiv:2510.25726, 2026a. URL https://arxiv.org/abs/2510.25726.

Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: Experiences on accelerating data parallel training, 2020. URL https://arxiv.org/abs/2006.15704.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, Chujun Tao, Binxu Li, Xuandong Zhao, Hejia Geng, Xiaojun Wu, Junwei Zhou, Xiaokun Chen, Hanwen Xing, Yubo Li, Qunhong Zeng, Di Wang, Yuanli Wang, Roey Ben Chaim, Penghao Jiang, Haotian Shen, Luyang Kong, Xinyi Liu, Runhui Wang, Xuanqing Liu, Jiachen Li, Xin Lan, Yueqian Lin, Wengao Ye, Junwei He, Songlin Li, Yue Zhang, Yipeng Gao, Yijiang Li, Ze Ma, Liqiang Jing, Tianyu Wang, Kaixin Li, Yiqi Xue, Haoran Lyu, Yizhuo He, Yuchen Tian, Shutong Wu, Bowei Wang, Yixuan Gao, Bo Chen, Litong Liu, Sikai Cheng, Jiajun Bao, Shuaicheng Tong, Shuwen Xu, Terry Yue Zhuo, Tinghan Ye, Qi Qi, Miao Li, Longtai

Liao, Zelin Tan, Chang Shi, Xilin Tang, Srinath Tankasala, Boqin Yuan, Yaoyao Qian, Jianhong Tu, Chenguang Wang, Yizhou Sun, Wei Wang, Aaron Taylor, Ziyue Yang, Changkun Guan, Zhikang Dong, Xinyu Zhang, Steven Dillmann, Han chung Lee, and Dawn Song. SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks, 2026b. URL https://arxiv.org/abs/2602.12670.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test. Advances in Neural Information Processing Systems, 38:136737–136756, 2026c.

Zhongzhi Li, Yucheng Shi, Zongxia Li, Ruhan Wang, Anhao Li, Zixun Huang, Junyao Yang, Lei Ke, Ninghao Liu, Haitao Mi, et al. Recursive synthesis for long-horizon terminal tasks. arXiv preprint arXiv:2608.05466, 2026d.

Weixin Liang, Yaohui Zhang, Zhengxuan Wu, Haley Lepp, Wenlong Ji, Xuandong Zhao, Hancheng Cao, Sheng Liu, Siyu He, Zhi Huang, Diyi Yang, Christopher Potts, Christopher D. Manning, and James Zou. Quantifying large language model usage in scientific papers. Nature Human Behaviour, 9(12):2599–2609, 2025. doi: 10.1038/s41562-025-02273-8.

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 Thirty-seventh Conference on Neural Information Processing Systems, 2023. URL https://openreview.net/forum?id=1qvx610Cu7.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. Transactions of the association for computational linguistics, 12:157–173, 2024.

Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. Fineweb-edu: the finest collection of educational content, 2024. URL https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu.

Rui Lu, Zhenyu Hou, Zihan Wang, Hanchen Zhang, Xiao Liu, Yujiang Li, Shi Feng, Jie Tang, and Yuxiao Dong. Deepdive: Advancing deep search agents with knowledge graphs and multi-turn rl, 2025. URL https://arxiv.org/abs/2509.10446.

Jeffrey Jian Ma, Milad Hashemi, Amir Yazdanbakhsh, Kevin Swersky, Ofir Press, Enhui Li, Vijay Janapa Reddi, and Parthasarathy Ranganathan. Swe-fficiency: Can language models optimize real-world repositories on real workloads? arXiv preprint arXiv:2511.06090, 2025.

Xuezhe Ma, Chunting Zhou, Xiang Kong, Junxian He, Liangke Gui, Graham Neubig, Jonathan May, and Luke Zettlemoyer. Mega: Moving average equipped gated attention. In The Eleventh International Conference on Learning Representations (ICLR-2023), 2023.

Xuezhe Ma, Xiaomeng Yang, Wenhan Xiong, Beidi Chen, Lili Yu, Hao Zhang, Jonathan May, Luke Zettlemoyer, Omer Levy, and Chunting Zhou. Megalodon: Efficient llm pretraining and inference with unlimited context length. Advances in Neural Information Processing Systems, 37:71831–71854, 2024.

Xuezhe Ma, Shicheng Wen, Linghao Jin, Bilge Acun, Ruihang Lai, Bohan Hou, Will Lin, Hao Zhang, Songlin Yang, Ryan Lee, et al. Gecko: An efficient neural architecture inherently processing sequences with arbitrary lengths. In The Third Annual Conference on Language Modeling (COLM-2026), 2026.

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. arXiv preprint arXiv:2508.15096, 2025.

Mantas Mazeika, Long Phan, Xuwang Yin, Andy Zou, Zifan Wang, Norman Mu, Elham Sakhaee, Nathaniel Li, Steven Basart, Bo Li, David Forsyth, and Dan Hendrycks. HarmBench: A standardized evaluation framework for automated red teaming and robust refusal, 2024. URL https://arxiv.org/abs/2402.04249.

Mike A Merrill, Alexander Glenn Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In The Fourteenth International Conference on Learning Representations, 2026. URL https://openreview.net/forum?id=a7Qa4CcHak.

Bettina Messmer, Vinko Sabolčec, and Martin Jaggi. Enhancing multilingual LLM pretraining with model-based data selection. arXiv preprint arXiv:2502.10361, 2025.

Meta AI. Introducing meta llama 3: The most capable openly available llm to date, 2024. URL https://ai.meta.com/blog/meta-llama-3/.

Sewon Min, Eric Wallace, Sameer Singh, Matt Gardner, Hannaneh Hajishirzi, and Luke Zettlemoyer. Compositional questions do not necessitate multi-hop reasoning. In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers, pages 4249–4257. Association for Computational Linguistics, 2019. doi: 10.18653/V1/P19-1416. URL https://doi.org/10.18653/v1/p19-1416.

Igor Molybog, Peter Albert, Moya Chen, Zachary DeVito, David Esiobu, Naman Goyal, Punit Singh Koura, Sharan Narang, Andrew Poulton, Ruan Silva, et al. A theory on adam instability in large-scale machine learning. arXiv preprint arXiv:2304.09871, 2023.

Magnus Müller and Gregor Žunič. Browser use: Enable ai to control your browser, 2024. URL https://github.com/browser-use/browser-use.

Dhruv Nathawani, Igor Gitman, Somshubra Majumdar, Evelina Bakhturina, Ameya Sunil Mahabaleshwarkar, Jian Zhang, and Jane Polak Scowcroft. Nemotron-Post-Training-Dataset-v1, July 2025. URL https://huggingface.co/datasets/nvidia/Nemotron-Post-Training-Dataset-v1.

Elliot Nelson, Georgios Kollias, Payel Das, Subhajit Chaudhury, and Soham Dan. Needle in the haystack for memory based large language models, 2024. URL https://arxiv.org/abs/2407.01437.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. Ms marco: A human-generated machine reading comprehension dataset. 2016.

Pu Ning, Quan Chen, Kun Tao, Xinyu Tang, Tianshu Wang, Qianggang Cao, Xinyu Kong, Zujie Wen, Zhiqiang Zhang, and Jun Zhou. Searchswarm: Towards delegation intelligence in agentic llms for long-horizon deep research. arXiv preprint arXiv:2606.09730, 2026.

Alexander Novikov et al. AlphaEvolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

NVIDIA. Nemotron 3 Nano: Open, efficient mixture-of-experts hybrid Mamba-Transformer model for agentic reasoning, 2025a. URL https://research.nvidia.com/labs/nemotron/files/NVIDIA-Nemotron-3-Nano-Technical-Report.pdf. Technical report.

NVIDIA. NVIDIA Nemotron Nano 2: An accurate and efficient hybrid Mamba-Transformer reasoning model. arXiv preprint arXiv:2508.14444, 2025b.

NVIDIA. Nemotron-Math-Proofs-v2. https://huggingface.co/datasets/nvidia/Nemotron-Math-Proofs-v2, 2026a. Hugging Face dataset.

NVIDIA. Nemotron-SFT-Agentic-v2. https://huggingface.co/datasets/nvidia/Nemotron-SFT-Agentic-v2, 2026b. Hugging Face dataset.

NVIDIA. Nemotron-SFT-ARC-AGI-v1. https://huggingface.co/datasets/nvidia/Nemotron-SFT-ARC-AGI-v1, 2026c. Hugging Face dataset.

NVIDIA. Nemotron-SFT-Competitive-Programming-v2. https://huggingface.co/datasets/nvidia/Nemotron-SFT-Competitive-Programming-v2, 2026d. Hugging Face dataset.

NVIDIA. Nemotron-SFT-CUDA-v1. https://huggingface.co/datasets/nvidia/Nemotron-SFT-CUDA-v1, 2026e. Hugging Face dataset.

NVIDIA. Nemotron-SFT-Instruction-Following-Chat-v2. https://huggingface.co/datasets/nvidia/Nemotron-SFT-Instruction-Following-Chat-v2, 2026f. Hugging Face dataset.

NVIDIA. Nemotron-SFT-Instruction-Following-Chat-v3. https://huggingface.co/datasets/nvidia/Nemotron-SFT-Instruction-Following-Chat-v3, 2026g. Hugging Face dataset.

NVIDIA. Nemotron-SFT-Math-v3. https://huggingface.co/datasets/nvidia/Nemotron-SFT-Math-v3, 2026h. Hugging Face dataset.

NVIDIA. Nemotron-SFT-Math-v4. https://huggingface.co/datasets/nvidia/Nemotron-SFT-Math-v4, 2026i. Hugging Face dataset.

NVIDIA. Nemotron-SFT-Multilingual-v1. https://huggingface.co/datasets/nvidia/Nemotron-SFT-Multilingual-v1, 2026j. Hugging Face dataset.

NVIDIA. Nemotron-SFT-Multilingual-v2. https://huggingface.co/datasets/nvidia/Nemotron-SFT-Multilingual-v2, 2026k. Hugging Face dataset.

NVIDIA. Nemotron-SFT-OpenCode-v1. https://huggingface.co/datasets/nvidia/Nemotron-SFT-OpenCode-v1, 2026l. Hugging Face dataset.

NVIDIA. Nemotron-SFT-Safety-v1. https://huggingface.co/datasets/nvidia/Nemotron-SFT-Safety-v1, 2026m. Hugging Face dataset.

NVIDIA. Nemotron-SFT-Safety-v2. https://huggingface.co/datasets/nvidia/Nemotron-SFT-Safety-v2, 2026n. Hugging Face dataset.

NVIDIA. Nemotron-SFT-Science-v2. https://huggingface.co/datasets/nvidia/Nemotron-SFT-Science-v2, 2026o. Hugging Face dataset.

NVIDIA. Nemotron-SFT-SWE-v2. https://huggingface.co/datasets/nvidia/Nemotron-SFT-SWE-v2, 2026p. Hugging Face dataset.

NVIDIA. Nemotron-SpecializedDomains-Finance-v1. https://huggingface.co/datasets/nvidia/Nemotron-SpecializedDomains-Finance-v1, 2026q. Hugging Face dataset.

Arnold Overwijk, Chenyan Xiong, and Jamie Callan. Clueweb22: 10 billion web documents with rich information. In Enrique Amigó, Pablo Castells, Julio Gonzalo, Ben Carterette, J. Shane Culpepper, and Gabriella Kazai, editors, SIGIR '22: The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, July 11 - 15, 2022, pages 3360–3362. ACM, 2022. doi: 10.1145/3477495.3536321. URL https://doi.org/10.1145/3477495.3536321.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In Forty-second International Conference on Machine Learning, 2025. URL https://openreview.net/forum?id=2GmDdhBdDk.

Tejal Patwardhan, Rachel Dias, Elizabeth Proehl, Grace Kim, Michele Wang, Olivia Watkins, Simón Posada Fishman, Marwan Aljubeh, Phoebe Thacker, Laurance Fauconnet, Natalie S. Kim, Patrick Chao, Samuel Miserendino, Gildas Chabot, David Li, Michael Sharman, Alexandra Barr, Amelia Glaese, and Jerry Tworek. GDPval: Evaluating AI model performance on real-world economically valuable tasks. arXiv preprint arXiv:2510.04374, 2025. URL https://arxiv.org/abs/2510.04374.

Guilherme Penedo, Hynek Kydlíček, Vinko Sabolčec, Bettina Messmer, Negar Foroutan, Amir Hossein Kargaran, Colin Raffel, Martin Jaggi, Leandro Von Werra, and Thomas Wolf. FineWeb2: One pipeline to scale them all – adapting pre-training data processing to every language. arXiv preprint arXiv:2506.20920, 2025.

Antoine Peyronnet, Fabian Gloeckle, and Amaury Hayat. LemmaBench: A live, research-level benchmark to evaluate LLM capabilities in mathematics, 2026. URL https://arxiv.org/abs/2602.24173.

Niklas Pfister, Václav Volhejn, Manuel Knott, Santiago Arias, Julia Bazinska, Mykhailo Bichurin, Alan Y. Commike, Janet Darling, Peter Dienes, Matthew Fiedler, David Haber, Matthias Kraft, Marco Lancini, Max Mathys, Damian Pascual-Ortiz, Jakub Podolak, Adrià Romero-López, Kyriacos Shiarlis, Andreas Signer, Zsolt Terek, Athanasios Theocharis, Daniel Timbrell, Samuel Trautwein, Samuel Watts, Yun-Han Wu, and Mateo Rojas-Carulla. Gandalf

the Red: Adaptive security for LLMs. In Forty-second International Conference on Machine Learning, 2025. URL https://openreview.net/forum?id=Y861SAkeCQ.

Renjie Pi, Grace Lam, Mohammad Shoeybi, Pooya Jannaty, Bryan Catanzaro, and Wei Ping. On data engineering for scaling llm terminal capabilities, 2026. URL https://arxiv.org/abs/2602.21193.

Varad Pimpalkhute, Yuekai Sun, and Mikhail Yurochkin. Advisor: A privileged advisor for agentic coding rollouts. https://github.com/ifm-ai/advisor, 2026. Software.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. Generalizing verifiable instruction following, 2025.

Zihan Qiu, Zekun Wang, Bo Zheng, Zeyu Huang, Kaiyue Wen, Songlin Yang, Rui Men, Le Yu, Fei Huang, Suozhi Huang, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Gated attention for large language models: Non-linearity, sparsity, and attention-sink-free. In Advances in Neural Information Processing Systems, 2025.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

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

Mohit Raghavendra, Soham Dan, Miguel Romero Calvo, Yannis Yiming He, Johannes Baptist Mols, Gautam Anand, Cole McCollum, Edgar Arakelyan, Vijay Bharadwaj, Andrew Park, Jeff Da, MohammadHossein Rezaei, Bing Liu, Brad Kenstler, and Yunzhong He. Swe atlas: Benchmarking coding agents beyond issue resolution. arXiv preprint arXiv:2605.08366, 2026. URL https://arxiv.org/abs/2605.08366.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models, 2020. URL https://arxiv.org/abs/1910.02054.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In First Conference on Language Modeling, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Liliang Ren, Yang Liu, Yelong Shen, and Weizhu Chen. Rethinking language model scaling under transferable hypersphere optimization. arXiv preprint arXiv:2603.28743, 2026. URL https://arxiv.org/abs/2603.28743.

Subham Sekhar Sahoo, Justin Deschenaux, Aaron Gokaslan, Guanghan Wang, Justin T Chiu, and Volodymyr Kuleshov. The diffusion duality. In Forty-second International Conference on Machine Learning, 2025. URL https://openreview.net/forum?id=9P9Y8FOSOk.

Subham Sekhar Sahoo, Lingjie Chen, Khiem Pham, Jonathan Geuter, Chaitanya Dwivedi, Varad Pimpalkhute, Yash Akhauri, Alexander Moreno, Mikhail Yurochkin, Zhenting Wang, Mostafa Elhoushi, Nolan Dey, Shane Bergsma, Joel Hestness, John Thickstun, Eric Xing, and Zhengzhong Liu. Unlocking lossless speedups in llms via discrete diffusion, 2026. URL https://arxiv.org/abs/2609.04010.

Mohammad Samragh, Arnav Kundu, David Harrison, Kumari Nishu, Devang Naik, Minsik Cho, and Mehrdad Farajtabar. Your llm knows the future: Uncovering its multi-token prediction potential. arXiv preprint arXiv:2507.11851, 2025.

Neha Sengupta, Sunil Kumar Sahu, Bokang Jia, Satheesh Katipomu, Haonan Li, Fajri Koto, Osama Mohammed Afzal, Samta Kamboj, Onkar Pandit, Rahul Pal, et al. Jais and jais-chat: Arabic-centric foundation and instruction-tuned open generative large language models. arXiv preprint arXiv:2308.16149, 2023.

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

Noam Shazeer. GLU variants improve transformer. arXiv preprint arXiv:2002.05202, 2020. URL https://arxiv.org/abs/2002.05202.

Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. arXiv preprint arXiv:1804.04235, 2018.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In International Conference on Learning Representations, 2017.

Qijia Shen, Zhiqi Huang, Vamsidhar Kamanuru, Aznaur Aliev, Jay Rainton, Ahmed Awelkair, Zhichen Zeng, Jiajun Li, Shi Dong, Yueming Yuan, et al. Seta: Scaling environments for terminal agents. arXiv preprint arXiv:2607.10891, 2026.

Xinyue Shen, Zeyuan Chen, Michael Backes, Yun Shen, and Yang Zhang. "do anything now": Characterizing and evaluating in-the-wild jailbreak prompts on large language models, 2024. URL https://arxiv.org/abs/2308.03825.

Daniel Shepard and Robin Salimans. Automationbench. arXiv preprint arXiv:2604.18934, 2026. URL https://arxiv.org/abs/2604.18934.

Quan Shi, Alexandra Zytek, Pedram Razavi, Karthik R Narasimhan, and Victor Barres. τ-knowledge: Evaluating conversational agents over unstructured knowledge. In Forty-third International Conference on Machine Learning, 2026. URL https://openreview.net/forum?id=XHZK5abtw2.

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

KaShun Shum, Yuzhen Huang, Hongjian Zou, Qi Ding, Yixuan Liao, Xiaoxin Chen, Qian Liu, and Junxian He. Predictive data selection: The data that predicts is the data that teaches. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu, editors, Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025, volume 267 of Proceedings of Machine Learning Research. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/shum25a.html.

Seamus Somerstep, Aritra Guha, Divesh Srivastava, and Yuekai Sun. The open source economic index of ai adoption and capability, 2026. URL https://arxiv.org/abs/2606.26118.

Guijin Son, Jiwoo Hong, Honglu Fan, Heejeong Nam, Hyunwoo Ko, Seungwon Lim, Jinyeop Song, Jinha Choi, Gonçalo Paulo, Youngjae Yu, and Stella Biderman. When AI co-scientists fail: SPOT—a benchmark for automated verification of scientific research, 2025. URL https://arxiv.org/abs/2505.11855.

Dan Su, Kezhi Kong, Ying Lin, Joseph Jennings, Brandon Norick, Markus Kliegl, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-CC: Transforming Common Crawl into a refined long-horizon pretraining dataset. arXiv preprint arXiv:2412.02595, 2024.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. arXiv preprint arXiv:2104.09864, 2021.

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

Sho Takase, Shun Kiyono, Sosuke Kobayashi, and Jun Suzuki. Spike no more: Stabilizing the pre-training of large language models. In Second Conference on Language Modeling, 2025. URL https://openreview.net/forum?id=52YBEzcI0l.

Liping Tang, Nikhil Ranjan, Omkar Pangarkar, Xuezhi Liang, Zhen Wang, Li An, Bhaskar Rao, Linghao Jin, Huijuan Wang, Zhoujun Cheng, Suqi Sun, Cun Mu, Victor Miller, Xuezhe Ma, Yue Peng, Zhengzhong Liu, and Eric P. Xing. Txt360: A top-quality llm pre-training dataset requires the perfect blend, 2024.

Chaofan Tao, Jierun Chen, Yuxin Jiang, Kaiqi Kou, Shaowei Wang, Ruoyu Wang, Xiaohui Li, Sidi Yang, Yiming Du, Jianbo Dai, et al. Swe-lego: Pushing the limits of supervised fine-tuning for software issue resolving. arXiv preprint arXiv:2601.01426, 2026.

Aaron Xuxiang Tian, Ruofan Zhang, Janet Tang, et al. Measuring harmfulness of computer-using agents. arXiv preprint arXiv:2508.00935, 2025. URL https://arxiv.org/abs/2508.00935. Dataset: https://huggingface.co/datasets/CUAHarm/CUAHarm.

Changxin Tian, Jiapeng Wang, Qian Zhao, Kunlong Chen, Jia Liu, Ziqi Liu, Jiaxin Mao, Xin Zhao, Zhiqiang Zhang, and Jun Zhou. Wsm: decay-free learning rate schedule via checkpoint merging for llm pre-training. In International Conference on Learning Representations, volume 2026, pages 134767–134795, 2026.

Minyang Tian, Luyu Gao, Dylan Zhang, Xinan Chen, Cunwei Fan, Xuefei Guo, Roland Haas, Pan Ji, Kittithat Krongchon, Yao Li, Shengyan Liu, Di Luo, Yutao Ma, HAO TONG, Kha Trinh, Chenyu Tian, Zihan Wang, Bohao Wu, Shengzhu Yin, Minhui Zhu, Kilian Lieret, Yanxin Lu, Genglin Liu, Yufeng Du, Tianhua Tao, Ofir Press, Jamie Callan, Eliu A Huerta, and Hao Peng. Scicode: A research coding benchmark curated by scientists. In The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track, 2024. URL https://openreview.net/forum?id=ADLaALtdoG.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv:2307.09288, 2023.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Is multihop QA in dire condition? measuring and reducing disconnected reasoning. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020, pages 8846–8863. Association for Computational Linguistics, 2020. doi: 10.18653/V1/2020.EMNLP-MAIN.712. URL https://doi.org/10.18653/v1/2020.emnlp-main.712.

Ada Defne Tur, Nicholas Meade, Xing Han Lù, Alejandra Zambrano, Arkil Patel, Esin Durmus, Spandana Gella, Karolina Stańczak, and Siva Reddy. SafeArena: Evaluating the safety of autonomous web agents. arXiv preprint arXiv:2503.04957, 2025. URL https://arxiv.org/abs/2503.04957.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Advances in Neural Information Processing Systems, volume 30. Curran Associates, Inc., 2017.

Alexander Sasha Vezhnevets, John P. Agapiou, Avia Aharon, Ron Ziv, Jayd Matyas, Edgar A. Duéñez-Guzmán, William A. Cunningham, Simon Osindero, Danny Karmon, and Joel Z. Leibo. Generative agent-based modeling with actions grounded in physical, social, or digital space using Concordia, 2023. URL https://arxiv.org/abs/2312.03664.

Bertie Vidgen, Austin Mann, Abby Fennelly, John Wright Stanly, Lucas Rothman, Marco Burstein, Julien Benchek, David Ostrofsky, Anirudh Ravichandran, Debnil Sur, Neel Venugopal, Alannah Hsia, Isaac Robinson, Calix Huang, Olivia Varones, Daniyal Khan, Michael Haines, Austin Bridges, Jesse Boyle, Koby Twist, Zach Richards, Chirag Mahapatra, Brendan Foody, and Osvald Nitski. Apex-agents. arXiv preprint arXiv:2601.14242, 2026. URL https://arxiv.org/abs/2601.14242.

vLLM Project. Add MoE support. LLM Compressor documentation, 2026. URL https://docs.vllm.ai/projects/llm-compressor/en/0.11.0/developer-tutorials/add-moe-support/. Accessed November 4, 2026.

Renxi Wang, Mingshan Hee, Fajri Koto, Timothy Baldwin, and Haonan Li. SkillGym: Training skill-use agents with automatic verifiable environment generation. 2026.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader? In Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 11279–11298, 2022.

Xi Wang and Laurence Aitchison. How to set adamw's weight decay as you scale model and dataset size. In

Forty-second International Conference on Machine Learning, 2025. URL https://openreview.net/forum?id=IszVnczhfz.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. arXiv preprint arXiv:2406.01574, 2024a.

Yuxia Wang, Haonan Li, Xudong Han, Preslav Nakov, and Timothy Baldwin. Do-Not-Answer: Evaluating safeguards in LLMs. In Yvette Graham and Matthew Purver, editors, Findings of the Association for Computational Linguistics: EACL 2024, pages 896–911, St. Julian's, Malta, March 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-eacl.61. URL https://aclanthology.org/2024.findings-eacl.61/.

Zengzhi Wang, Fan Zhou, Xuefeng Li, and Pengfei Liu. Octothinker: Mid-training incentivizes reinforcement learning scaling. arXiv preprint arXiv:2506.20512, 2025.

Maurice Weber, Daniel Fu, Quentin Anthony, Yonatan Oren, Shane Adams, Anton Alexandrov, Xiaozhong Lyu, Huu Nguyen, Xiaozhe Yao, Virginia Adams, Ben Athiwaratkun, Rahul Chalamala, Kezhen Chen, Max Ryabinin, Tri Dao, Percy Liang, Christopher Ré, Irina Rish, and Ce Zhang. RedPajama: an open dataset for training large language models. arXiv preprint arXiv:2411.12372, 2024.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. Browsecomp: A simple yet challenging benchmark for browsing agents. arXiv preprint arXiv:2504.12516, 2025. URL https://arxiv.org/abs/2504.12516.

Alexander Wettig, Kyle Lo, Sewon Min, Hannaneh Hajishirzi, Danqi Chen, and Luca Soldaini. Organize the web: Constructing domains enhances pre-training data curation. arXiv preprint arXiv:2502.10341, 2025. doi: 10.48550/arXiv.2502.10341. URL https://arxiv.org/abs/2502.10341.

Mitchell Wortsman, Tim Dettmers, Luke Zettlemoyer, Ari Morcos, Ali Farhadi, and Ludwig Schmidt. Stable and low-precision training for large-scale vision-language models. arXiv preprint arXiv:2304.13013, 2023.

Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. Fast-dllm v2: Efficient block-diffusion llm, 2025. URL https://arxiv.org/abs/2509.26328.

Zijian Wu, Xiangyan Liu, xinyuan zhang, Lingjun Chen, Fanqing Meng, Lingxiao Du, Yiran Zhao, Fanshi Zhang, Yaoqi Ye, Jiawei Wang, Zirui Wang, Jinjie Ni, Yufan Yang, Arvin Xu, and Michael Qizhe Shieh. MCPMark: A benchmark for stress-testing realistic and comprehensive MCP use. In The Fourteenth International Conference on Learning Representations, 2026. URL https://openreview.net/forum?id=uobROwBsJm.

Sarina Xi, Vishisht Rao, Justin Payan, and Nihar B. Shah. FLAWS: A benchmark for error identification and localization in scientific papers, 2025. URL https://arxiv.org/abs/2511.21843.

Jian Xie, Tianhe Lin, Zilu Wang, Yuting Ning, Yuekun Yao, Tianci Xue, Zhehao Zhang, Zhongyang Li, Kai Zhang, Yufan Wu, Shijie Chen, Boyu Gou, Mingzhe Han, Yifei Wang, Vint Lee, Xinpeng Wei, Xiangjun Wang, Yu Su, and Huan Sun. Quest: Training frontier deep research agents with fully synthetic tasks, 2026.

Wenhan Xiong, Jingyu Liu, Igor Molybog, Hejia Zhang, Prajjwal Bhargava, Rui Hou, Louis Martin, Rashi Rungta, Karthik Abinav Sankararaman, Barlas Oguz, Madian Khabsa, Han Fang, Yashar Mehdad, Sharan Narang, Kshitiz Malik, Angela Fan, Shruti Bhosale, Sergey Edunov, Mike Lewis, Sinong Wang, and Hao Ma. Effective long-context scaling of foundation models, 2023. URL https://arxiv.org/abs/2309.16039.

XYZ Agentic Team. Ai4ai at scale: A full-pipeline system for enhancing llm agentic capabilities. Technical report, XYZ AI Lab, 2026. URL https://xyz-lab.ai/blogs/ai4ai-at-scale/assets/bounded-exploration-ai4ai-system-optimization.pdf.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei

Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Greg Yang, Edward J Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. arXiv preprint arXiv:2203.03466, 2022.

Sidi Yang, Chaofan Tao, Jierun Chen, Tiezheng Yu, Ruoyu Wang, Yuxin Jiang, Yiming Du, Wendong Xu, Jing Xiong, Taiqiang Wu, et al. What makes interaction trajectories effective for training terminal agents? arXiv preprint arXiv:2606.03461, 2026a.

Xiao-Wen Yang, Zihao Zhang, Jianuo Cao, Zhi Zhou, Zenan Li, Lan-Zhe Guo, Yuan Yao, Taolue Chen, Yu-Feng Li, and Xiaoxing Ma. FormalML: A benchmark for evaluating formal subgoal completion in machine learning theory. In International Conference on Learning Representations, 2026b. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/55d4e6e1a0557d3d9e3140966c1db533-Abstract-Conference.html.

Ziyi Yang, Zaibin Zhang, Zirui Zheng, Yuxian Jiang, Ziyue Gan, Zhiyu Wang, et al. OASIS: Open agent social interaction simulations with one million agents, 2024. URL https://arxiv.org/abs/2411.11581.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. Advances in Neural Information Processing Systems, 38:113222–113244, 2026.

Xiangchi Yuan, Dachuan Shi, Chunhui Zhang, Zheyuan Liu, Shenglong Yao, Soroush Vosoughi, and Wenke Lee. Behavior knowledge merge in reinforced agentic models. In Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 33007–33028, 2026.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? arXiv preprint arXiv:2504.13837, 2025.

Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. SWAG: A large-scale adversarial dataset for grounded commonsense inference. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018, pages 93–104. Association for Computational Linguistics, 2018. doi: 10.18653/V1/D18-1009. URL https://doi.org/10.18653/v1/d18-1009.

Barry Zhang, Keith Lazuka, and Mahesh Murag. Equipping agents for the real world with Agent Skills. Anthropic Engineering, October 2025a. URL https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills. Published October 16, 2025.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. Advances in Neural Information Processing Systems, 32, 2019.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin godel machine: Open-ended evolution of self-improving agents, 2026. URL https://arxiv.org/abs/2505.22954.

Jie Zhang, Cezara Petrui, Kristina Nikolić, and Florian Tramèr. RealMath: A continuous benchmark for evaluating language models on research-level mathematics, 2025b. URL https://arxiv.org/abs/2505.12575.

Jingyu Zhang, Ahmed Elgohary, Xiawei Wang, A S M Iftekhar, Ahmed Magooda, Benjamin Van Durme, Daniel Khashabi, and Kyle Jackson. Jailbreak distillation: Renewable safety benchmarking, 2025c. URL https://arxiv.org/abs/2505.22037.

Zhehao Zhang, Weijie Xu, Fanyou Wu, and Chandan K. Reddy. FalseReject: A resource for improving contextual safety and mitigating over-refusals in LLMs via structured reasoning, 2025d. URL https://arxiv.org/abs/2505.08054.

Jiale Zhao, Guoxin Chen, Fanzhe Meng, Minghao Li, Jie Chen, Hui Xu, Yongshuai Sun, Xin Zhao, Ruihua Song, Yuan Zhang, Peng Wang, Cheng Chen, Jirong Wen, and Kai Jia. Immersion in the github universe: Scaling coding agents to mastery, 2026a. URL https://arxiv.org/abs/2602.09892.

Jiale Zhao, Guoxin Chen, Fanzhe Meng, Wayne Xin Zhao, Ruihua Song, Ji-Rong Wen, and Kai Jia. Denovoswe: Scaling long-horizon environments for generating entire repositories from scratch, 2026b. URL https://arxiv.org/abs/2606.10728.

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

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, 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/91f18a1287b398d378ef22505bf41832-Abstract-Datasets_and_Benchmarks.html.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. Advances in neural information processing systems, 37:62557–62583, 2024.

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

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models, 2023b. URL https://arxiv.org/abs/2311.07911.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In International Conference on Learning Representations (ICLR), 2024. URL https://arxiv.org/abs/2307.13854. arXiv:2307.13854.

Hanqing Zhu, Wenyan Cong, Zhizhou Sha, Sagnik Mukherjee, Xinyuan Song, David González-Martínez, Xiaoxia Wu, Yuandong Tian, Shiwei Liu, David Z Pan, et al. Iso: An rlvr-native optimization stack. arXiv preprint arXiv:2607.19331, 2026a.

Minhui Zhu, Minyang Tian, Xiaocheng Yang, Tianci Zhou, Lifan Yuan, Penghao Zhu, Eli Chertkov, Shengyan Liu, Yufeng Du, Ziming Ji, et al. Probing the critical point (critpt) of ai reasoning: a frontier physics research benchmark. arXiv preprint arXiv:2509.26574, 2026b. URL https://arxiv.org/abs/2509.26574.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models, 2023. URL https://arxiv.org/abs/2307.15043.