Nemotron 3 Super: Open, Efficient Mixture-of-Experts Hybrid Mamba-Transformer Model for Agentic Reasoning

NVIDIA

Abstract.
We describe the pre-training, post-training, and quantization of Nemotron 3 Super, a 120 billion (active 12 billion) parameter hybrid Mamba-Attention Mixture-of-Experts model. Nemotron 3 Super is the first model in the Nemotron 3 family to 1) be pre-trained in NVFP4, 2) leverage LatentMoE, a new Mixture-of-Experts architecture that optimizes for both accuracy per FLOP and accuracy per parameter, and 3) include MTP layers for inference acceleration through native speculative decoding. We pre-trained Nemotron 3 Super on 25 trillion tokens followed by post-training using supervised fine tuning (SFT) and reinforcement learning (RL). The final model supports up to 1M context length and achieves comparable accuracy on common benchmarks, while also achieving up to 2.2× and 7.5× higher inference throughput compared to GPT-OSS-120B and Qwen3.5-122B, respectively. Nemotron 3 Super datasets, along with the base, post-trained, and quantized checkpoints, are open-sourced on HuggingFace.

1. Introduction

The last few years have seen a rise in the popularity of Mixture-of-Experts (MoE) based Large Language Models (LLMs) (DeepSeek-AI, 2025c; Yang et al., 2025; GLM-4.5-Team, 2025). MoEs help LLMs achieve higher accuracy at a lower active parameter count than regular dense models (Dai et al., 2024; Lepikhin et al., 2020). Orthogonal to MoEs, Hybrid Mamba-Attention models have shown promise in significantly improving inference throughput (NVIDIA, 2025c). We combine these two directions of improvement in Nemotron 3 (NVIDIA, 2025c). As part of our Nemotron 3 series of models, we present Nemotron 3 Super—a 12 billion active, 120 billion total parameter MoE hybrid Mamba-Attention model. Nemotron 3 Super achieves better or on-par benchmark accuracies than GPT-OSS-120B (OpenAI, 2025) and Qwen3.5-122B while achieving up to 2.2× and 7.5× higher inference throughput, respectively, on the 8k token input / 64k token output setting.

Nemotron 3 Super is our first model to use LatentMoE (Elango et al., 2026) - a novel MoE architecture that achieves better accuracy per parameter and per FLOP than regular MoEs. Nemotron 3 Super also incorporates Multi-Token-Prediction (MTP), which accelerates inference through speculative decoding while improving overall model quality. We pre-trained Nemotron 3 Super in NVFP4, demonstrating stable and accurate pre-training in low precision. Similar to Nemotron 3 Nano (NVIDIA, 2025a), we pre-trained Nemotron 3 Super on 25 trillion text tokens divided into 2 phases. The first phase accounted for 80% of pre-training (20 trillion tokens) and focused on diversity and broad coverage, while the second phase accounted for 20% of pre-training (5 trillion tokens) and focused on high-quality data and benchmark accuracy. Our base model achieves significantly better accuracy than similarly sized state-of-the-art base models, such as GLM-4.5-Air-Base (GLM-4.5-Team, 2025) and Ling-flash-Base-2.0 (Ling-Team, 2025).

We trained Nemotron 3 Super with a strong emphasis on agentic capabilities. To support this objective, we substantially scaled the breadth of our RL environments, the volume and quality

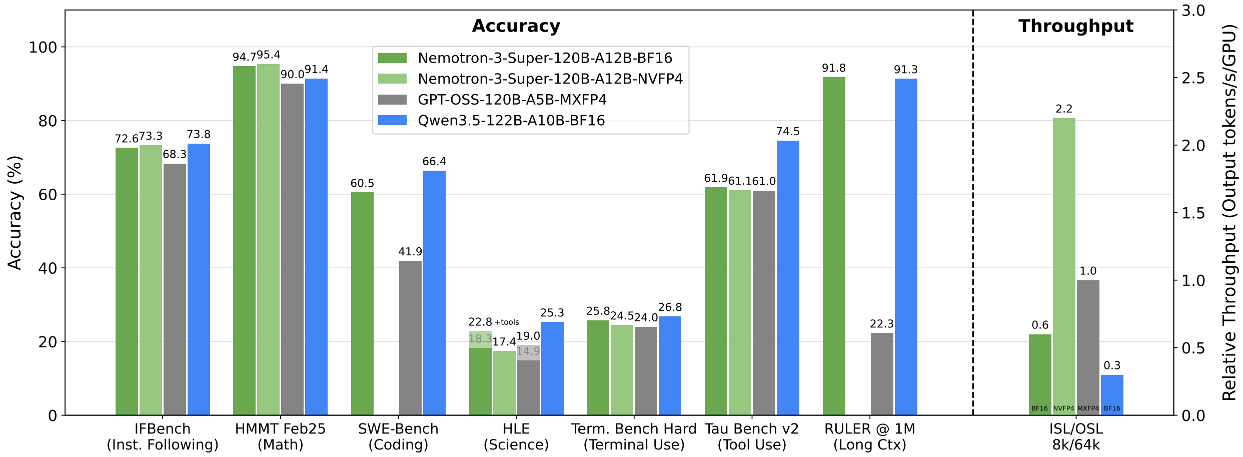


Figure 1 | Accuracy and throughput comparisons of Nemotron 3 Super with GPT-OSS-120B and Qwen3.5-122B. Nemotron 3 Super achieves comparable accuracies across popular benchmarks but provides the highest inference throughput; on 8k input sequence lengths and 64k output sequence lengths, Nemotron 3 Super provides up to 2.2× and 7.5× higher throughput than GPT-OSS-120B and Qwen3.5-122B, respectively. We measured throughput on B200 GPUs with vLLM and TRT-LLM and use the best out of the two frameworks for each model. For GPT-OSS-120B we use MXFP4 weights, MXFP8 activations, and FP8 KV-Cache; for Qwen3.5-122B we use BF16. We used the OpenHands harness to evaluate SWE-Bench.

of agentic training data, and the overall amount of post-training focused on multi-step tool-using behavior. To train effectively on this diverse set of long-horizon tasks, we made substantial improvements to the resiliency of our RL infrastructure, enabling large-scale asynchronous training. This expanded agentic training recipe yields substantial improvements over Nemotron 3 Nano across software engineering, terminal use, and general tool use benchmarks.

We are publicly sharing the training recipe for Nemotron 3 Super on the Nemotron Developer Repository[1]. We are also openly releasing the following:

Checkpoints

- Nemotron 3 Super 120B-A12B NVFP4 🤗 : post-trained and NVFP4 quantized model
- Nemotron 3 Super 120B-A12B FP8 🤗 : post-trained and FP8 quantized model
- Nemotron 3 Super 120B-A12B BF16 🤗 : post-trained model
- Nemotron 3 Super 120B-A12B Base BF16 🤗 : base model
- Qwen3-Nemotron-235B-A22B-GenRM-2603 🤗 : GenRM used for RLHF

Data

- Nemotron-Pretraining-Specialized-v1.1 🤗 : a collection of synthetic datasets aimed to improve LLM capabilities in code concepts and algorithms, formal logic, economics, and multiple choice questions.
- Nemotron-Super-Post-Training-Data 🤗 : a collection of RL environments and SFT datasets targeting a broad range of agentic capabilities.

[1] https://github.com/NVIDIA-NeMo/Nemotron

The report is organized into 3 broad sections: Pre-training (§2), Post-training (§3), and Quantization (§4), each describing in detail our approach.

2. Pretraining

In this section, we highlight the key features of Nemotron 3 Super 120B-A12B Base, detailing its hybrid Mamba-Attention Mixture-of-Experts (MoE) architecture, NVFP4 pre-training, hyperparameter configurations, long-context extension, and the 25-trillion-token corpus used for pretraining. We also demonstrate that Nemotron-3 Super 120B A12B Base achieves superior accuracy compared to other public state-of-the-art models—including Ling-flash-Base-2.0 and GLM-4.5-Air-Base—across a comprehensive suite of benchmarks.

2.1. Model Architecture

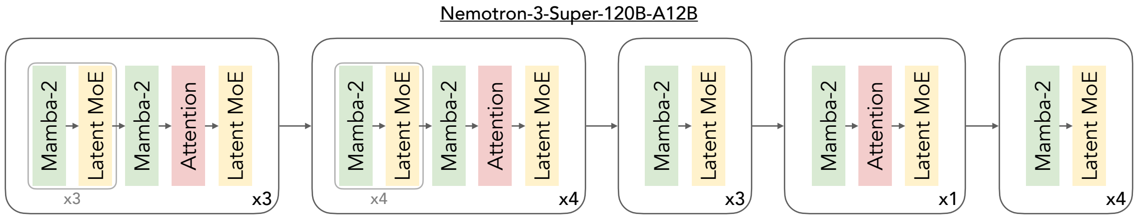


Figure 2 | Nemotron 3 Super layer pattern. Similar to Nemotron 3 Nano, we use a hybrid Mamba-Attention architecture, but Nemotron 3 Super is the first model to scale sparsely using LatentMoE layers rather than standard MoE layers.

Configuration	**Nemotron 3 Super 120B-A12B Base**
Total Layers	88
Model Dimension	4096
Q-Heads (n_q)	32
KV-Heads (n_{kv})	2
Head Dimension	128
Mamba State Dimension	128
Mamba Groups	8
Mamba Heads	128
Mamba Head Dimension	64
Expert Hidden Dimension	2688
Shared Expert Intermediate Size	5376
Total Experts per Layer	512
Top-k (Activated Experts)	22
MoE Latent Size	1024
MTP layers (shared weight)	2

Table 1 | Nemotron 3 Super Architectural Dimensions. The model employs a hybrid Mamba-2 and MoE design with strategic global attention layers to optimize the balance between sequence modeling performance and inference throughput.

Nemotron 3 Super 120B-A12B Base scales up the hybrid Mamba-Attention Mixture-of-Experts (MoE) architecture introduced in Nemotron-3 Nano (NVIDIA, 2025c). We extend this foundation to 120.6B total parameters, maintaining a constrained active budget of 12.7B parameters (12.1B excluding embeddings) per forward pass. The architecture comprises three core pillars: sparse LatentMoE scaling (§2.1.1), Multi-Token Prediction (MTP) for inference acceleration (§2.1.2), and a periodic hybrid interleaving pattern (§2.1.3).

2.1.1. LatentMoE: Hardware-Aware Expert Design for Improved Accuracy per Byte

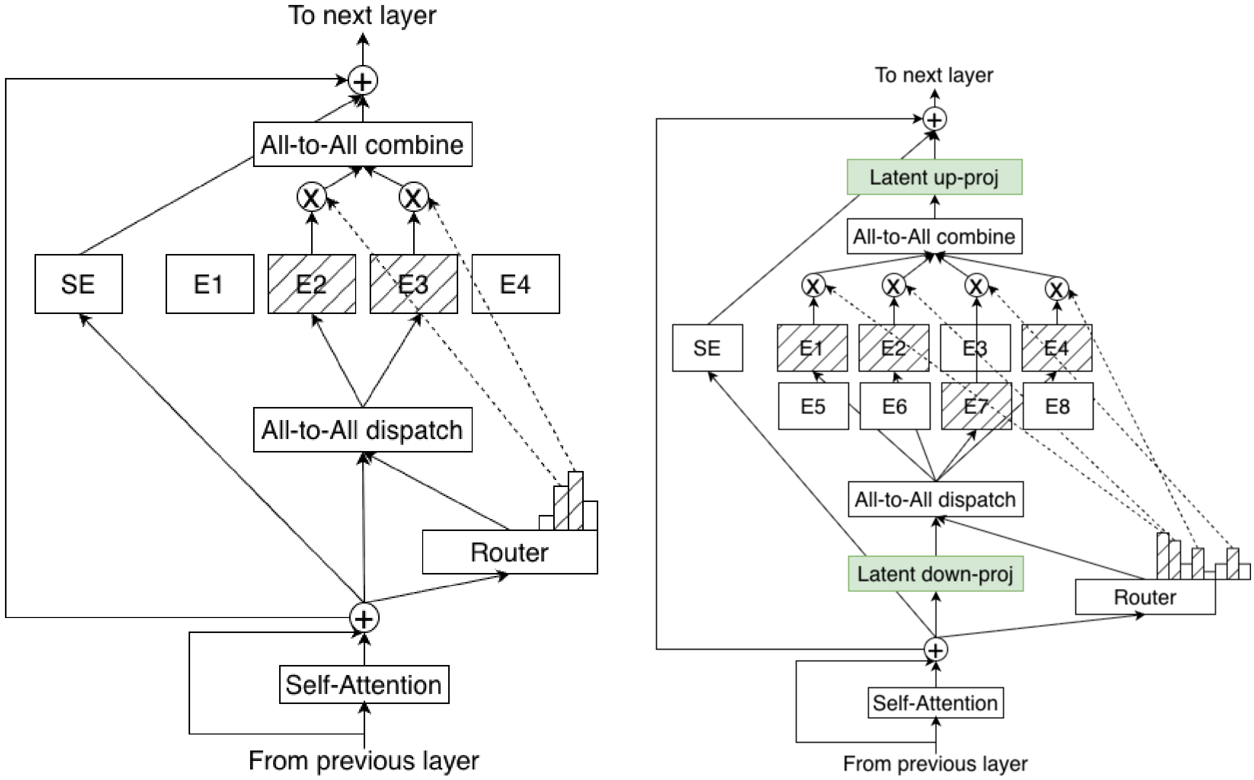
Mixture-of-Experts (MoE) architectures have emerged as a promising approach to maximize accuracy under fixed inference cost, allowing models to scale in parameter count while keeping floating-point operations (FLOPs) per token constant. Existing MoE designs are largely motivated by high-level sparsity arguments and optimized for offline, throughput-oriented settings, with little consideration for online deployments that impose strict latency, memory bandwidth, and communication constraints. Whereas accuracy per FLOP reflects computational efficiency, accuracy per parameter captures memory footprint, memory bandwidth, routing-induced communication, and sharding overhead. Neglecting these factors can yield architectures that appear efficient in aggregate compute yet incur substantial inefficiency in practice.

Motivated by these observations, we revisited MoE design from a hardware–software co-design perspective. Through systematic analysis of existing MoE systems across the throughput–latency Pareto frontier, together with accuracy measurements and theoretical analysis, we identified structural inefficiencies in prevailing MoE designs that limit accuracy per unit inference cost. From this analysis we distilled the following design principles for efficient MoE scaling:

1. In low-latency serving, MoE inference is often dominated by the memory bandwidth cost of reading expert weights. Each expert matrix has size $d \times m$, where d is the hidden dimension and m is the expert FFN intermediate dimension; reducing this cost therefore requires decreasing d or m.
2. In throughput-oriented serving, distributed MoE inference is dominated by all-to-all routing. Routing volume scales as $d \times K$, where K is the number of active experts; reducing communication overhead therefore requires decreasing d or K.
3. Preserving model quality requires preserving the effective nonlinear budget $K \cdot m$. To relieve memory and communication bottlenecks without sacrificing quality, K and m should thus be held fixed.
4. A task-specific effective feature rank r_{eff} imposes a lower limit on how much d can be reduced; reducing d below this limit causes model quality to collapse.
5. Scaling both the total number of experts N and the top-K experts per token improves quality by exponentially expanding the space of expert combinations.

Principles (1)–(3) imply that the hidden dimension d is the most promising axis for reduction, enabling gains in both throughput- and latency-oriented regimes without significant loss in accuracy. Principle (4) gives a lower bound on how far d can be reduced without collapse. Principle (5) indicates that increasing N and K improves quality; because memory bandwidth and communication scale linearly with K, we can increase K by a factor α and reduce d by the same factor α to obtain higher accuracy at similar inference cost. Guided by these insights, we developed LatentMoE (Elango et al., 2026), a MoE architecture designed to achieve higher accuracy than a standard MoE at similar inference cost.

The LatentMoE architecture is illustrated in Figure 3b. Each input token $x \in \mathbb{R}^d$ is first projected into a lower-dimensional latent space $\mathbb{R}^\ell$ via a learnable down-projection matrix $W_\downarrow \in \mathbb{R}^{\ell \times d}$. The



(a) Standard MoE architecture.

(b) LatentMoE architecture.

Figure 3 | Standard MoE vs. LatentMoE. In LatentMoE, tokens are projected from the hidden dimension d into a smaller latent dimension ℓ for routing and expert computation, reducing routed parameter loads and all-to-all traffic by a factor d/ℓ. These savings are used to increase both the total number of experts and the top-K active experts per token by the same factor, improving model accuracy at approximately constant inference cost.

compressed representation is then routed to an expanded set of experts that operate entirely in this latent space. After expert computation, the outputs are aggregated and projected back to dimension d via a learnable up-projection matrix $W_{\uparrow} \in \mathbb{R}^{d \times \ell}$. Shifting routed expert computation and all-to-all traffic into the latent space reduces both per-expert weight loads and communication payloads by a factor d/ℓ relative to a standard MoE. We use these savings to increase the total number of experts from N to $N' = N \cdot d/\ell$ and the top-K active experts per token from K to $K' = K \cdot d/\ell$. The reduction in dimension offsets the increase in expert count and in K, yielding higher model quality at a similar computational and communication budget. To preserve quality, all non-routed computations—including the routing gate (gating network), shared expert computation, and non-expert layers—remain in the full hidden dimension d, as they do not contribute significantly to the targeted bottlenecks. We refer the reader to the LatentMoE technical report (Elango et al., 2026) for further details.

2.1.2. Multi-Token Prediction

Nemotron-3 Super incorporates a Multi-Token Prediction (MTP) objective to improve both modeling quality and inference efficiency. Unlike conventional next-token training, MTP optimizes the model to predict multiple future tokens at each position (Gloeckle et al., 2024; DeepSeek-AI, 2025c). This encourages representations that capture multi-step dependencies and longer-range structure, leading to consistent improvements in validation loss and downstream benchmark accuracy.

Beyond quality gains, MTP enables native speculative decoding. The auxiliary prediction heads function as an internal draft model: during inference, they generate candidate continuations that are verified by the main model in a single forward pass. This substantially reduces decoding latency while introducing minimal additional FLOPs—significantly less than required by an external draft model. Although speculative decoding is particularly effective at small batch sizes, recent work shows that it can also improve throughput in larger-batch and sparse MoE settings (Huang et al., 2025).

Design for Robust Autoregressive Drafting. Standard MTP implementations use N independent heads, each trained to predict a fixed offset (e.g., $n+2, \ldots, n+N+1$). While effective during training, this limits speculative decoding to at most N draft tokens. Longer drafts require either increasing N or reusing a single offset-trained head autoregressively.

Reusing a fixed-offset head introduces a training–inference mismatch: the head is trained under ground-truth hidden states but, at inference, conditions on its own generated states. This distribution shift often reduces acceptance rates as draft length increases.

Nemotron-3 Super addresses this limitation by sharing parameters across multiple MTP heads during training, yielding a unified prediction head exposed to multiple offsets. This shared-weight formulation regularizes the head across prediction horizons and improves robustness to the self-generated hidden states encountered during autoregressive drafting. As a result, the same head can be applied recursively at inference to generate longer drafts with more stable acceptance behavior. While acceptance rates naturally decrease as draft length increases, the degradation is substantially milder than with independently trained offset heads. This enables more effective speculative decoding without introducing additional parameters or requiring a separate draft model.

Speculative Decoding Performance. We evaluate MTP quality using SPEED-Bench (Abramovich et al., 2026), a benchmark tailored for speculative decoding. Table 2 reports the average acceptance length (tokens accepted per verification step) with a fixed draft length of 7. Nemotron-3 Super achieves the highest overall average acceptance length (3.45), outperforming DeepSeek-R1 across all domains and remaining competitive with Qwen3-Next.

Figure 4 plots acceptance rate as a function of draft token index. As expected, acceptance decreases monotonically with draft depth for all models. However, Nemotron-3 Super consistently maintains higher acceptance than DeepSeek-R1 at every draft position and closely tracks or exceeds Qwen3-Next across most indices. Notably, the gap becomes more pronounced at larger draft indices (4–7), where recursive drafting is most challenging. This behavior indicates improved stability of the shared-head autoregressive design under longer speculative rollouts.

Overall, MTP in Nemotron-3 Super improves both representation learning and decoding efficiency, enabling higher acceptance at extended draft lengths without relying on an external draft model. These acceptance gains translate to superior serving efficiency on Blackwell hardware. As shown in Figure 5, increasing the draft depth ($D = 1$ to $D = 3$) via MTP significantly shifts the throughput–latency Pareto frontier, delivering higher aggregate output tokens per second (TPS) for any given median user latency compared to the baseline with MTP disabled.

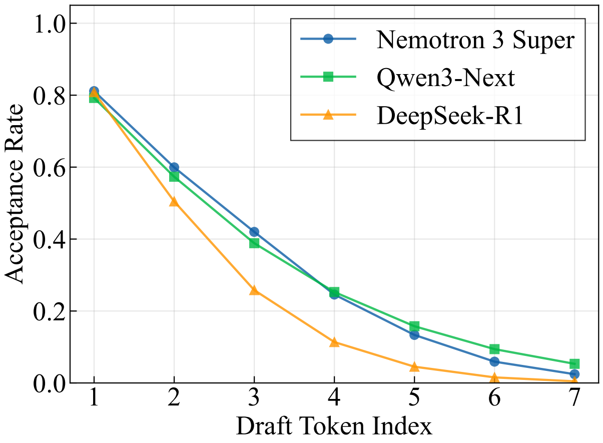


Figure 4 | MTP acceptance rate by draft index on SPEED-Bench using a draft length of 7.

Category	DSR1	Qwen3 Next	Nemotron3 Super
Coding	2.99	**4.32**	3.78
Humanities	2.67	3.07	**3.26**
Math	2.98	**3.89**	3.73
Multilingual	2.83	3.97	**4.05**
QA	2.63	3.09	**3.16**
RAG	2.79	3.53	**3.78**
Reasoning	2.80	3.47	**3.59**
Roleplay	2.19	2.17	**2.82**
STEM	2.79	**3.37**	3.30
Summarization	2.59	3.06	**3.48**
Writing	2.41	2.69	**2.99**
Average	2.70	3.33	**3.45**

Table 2 | MTP average acceptance lengths on SPEED-Bench using a draft length of 7.

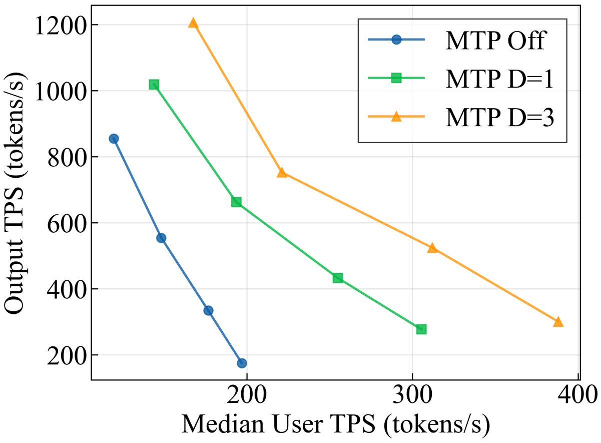


Figure 5 | Total vs. user throughput for an NVFP4 checkpoint (TRT-LLM, TP=1, B300 GPU). Comparing MTP off vs. MTP with draft lengths of 1,3. Measured on SPEED-Bench's Throughput-1k split, with 1k output tokens.

2.1.3. Hybrid Interleaved MoE Architecture and Global Anchors

Nemotron 3 Super adopts a hybrid Mixture-of-Experts (MoE) architecture designed to maximize inference throughput—particularly for long-context reasoning—while preserving the modeling capacity of large-scale dense Transformers. The primary systems bottleneck in modern sequence models is the quadratic growth of the KV cache in self-attention layers. To address this, we predominantly utilize Mamba-2 blocks (Dao & Gu, 2024), which operate with a constant-sized state during generation, substantially reducing memory overhead and latency.

The 88-layer stack follows a periodic interleaving pattern in which MoE layers are paired with Mamba-2 blocks. While Mamba provides efficient linear-time sequence modeling, a limited number of self-attention layers are strategically inserted as global "anchors" to enable full-token interaction and long-range information routing across the stack. This hybrid interleaving preserves global dependency modeling while offloading the majority of computation to the more efficient Mamba and sparse MoE components. Table 1 and Figure 2 provide a comprehensive summary of the structural parameters and the specific interleaving pattern of the hybrid stack.

The attention layers employ Grouped-Query Attention (GQA) with 32 query heads and 2 KV heads (head dimension 128). Consistent with prior Nemotron models, we omit positional embeddings, dropout, and bias terms in linear layers, use RMSNorm for normalization, and maintain un-tied embedding and output weights. This configuration supports context lengths of up to 1M tokens.

Sparse scaling further improves efficiency. Each MoE layer activates only a subset of experts per token (top-22 routing), enabling the model to scale to 120.6B total parameters while maintaining a 12.7B active parameter budget per forward pass.

Overall, the synergy between linear-time Mamba blocks, sparsely activated MoE capacity, and strategically placed attention anchors enables Nemotron 3 Super to deliver strong long-context performance while remaining optimized for real-world deployment on modern hardware.

2.2. NVFP4 Pretraining

Table 3 | Precision by Layer Type

Layer Type	Format	Rationale
All Linear Layers Unless Otherwise Noted	NVFP4	
Final 15% of Network	BF16	Promote training stability at scale
Latent Projections	BF16	Strategically kept in BF16 as step-time impact is negligible
MTP Layers	BF16	Preserves multi-token prediction capabilities
QKV & Attention Projections	BF16	Maintain fidelity of few attention layers
Mamba Output Projection	MXFP8	Mitigates high incidence of underflows observed when quantizing this layer to NVFP4 at smaller scales
Embedding Layers	BF16	

Nemotron 3 Super was trained with the NVFP4 pretraining recipe detailed in the Nemotron 3 white paper (NVIDIA, 2025c). All linear layers, unless otherwise noted in Table 3, are trained using the open-source NVFP4 GEMM kernels provided by Transformer Engine with the cuBLAS backend (NVIDIA Corporation, 2024) for fprop, dgrad, and wgrad GEMMs. This framework performs quantization of weights, activations, and gradients to NVFP4 according to the scheme first introduced in NVIDIA (2025d). Weights are quantized to NVFP4 using two-dimensional (2D) block scaling to maintain consistency between quantized weights in the forward and backward pass. Gradients and activations are quantized to NVFP4 using one-dimensional (1D) blocks along the GEMM reduction axis. Random Hadamard Transforms (RHTs) are performed on inputs to wgrad and stochastic rounding is applied to gradient tensors. The NVFP4 format utilizes an E2M1 element format with 16-element micro-blocks, E4M3 micro-block scaling factors, and a second-level FP32 global scale. Nemotron 3 Super showcases large-scale stable training in NVFP4 up to 25T tokens.

During the training of Nemotron 3 Super, we observed a growth in the number of zero-valued weight gradient elements and investigated the root cause to validate training health. Magnitude patterns emerged within some expert layers, characterized by the norms of FC1 output channels and corresponding FC2 input channels converging toward zero (Figure 6). By the end of pretraining, zero-valued weight gradient elements accounted for 7% of total parameters, appearing to correlate with the magnitude patterns. We believe that NVFP4 quantization increases the incidence of true zeros in the weight gradients that could have been more easily be represented by BF16 or MXFP8. Low-norm channels likely attenuate quicker when these layers are trained in NVFP4.

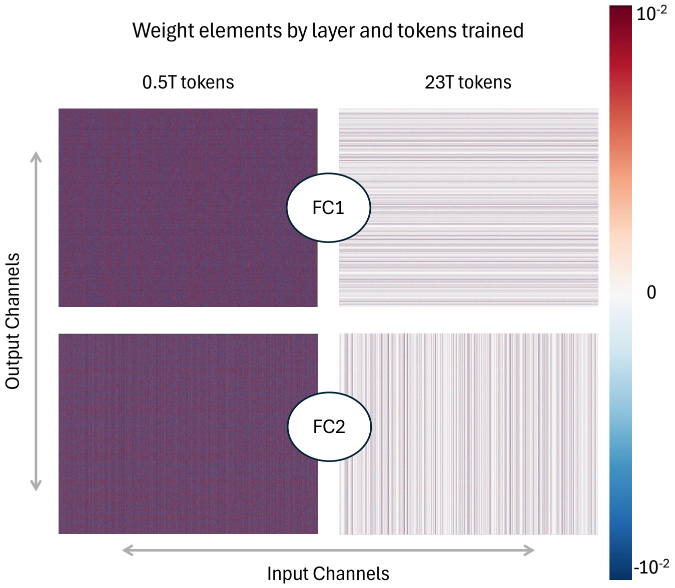


Figure 6 | Channel Magnitude Patterns in Weights of Expert Layers in Nemotron 3 Super. Patterns emerge as training progresses. Top: Early layer routed expert FC1 weight matrix at 0.5T tok and 23T tok. Bottom: Early layer routed expert FC2 weight matrix at 500B tok and 23T tok. Low-norm output channels of FC1 align with low-norm input channels of FC2.

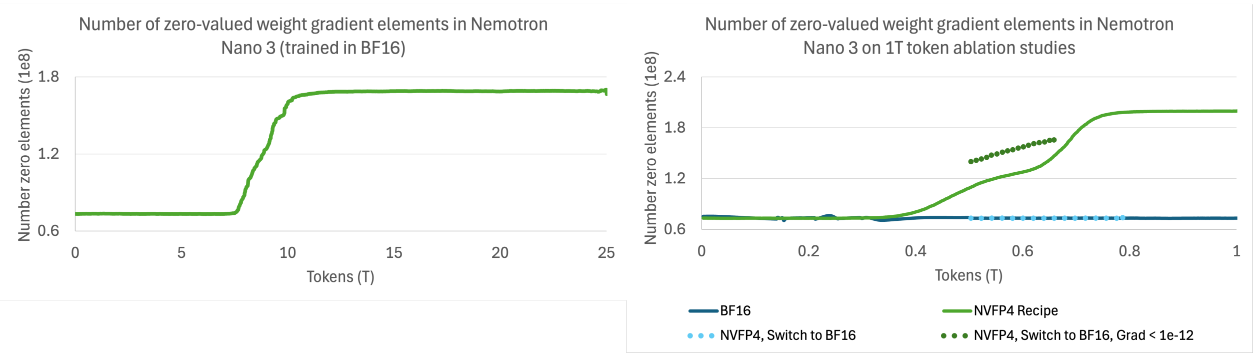


Figure 7 | Number of Zero-Valued Weight Gradient Elements on Nemotron 3 Nano. Left: Released Nemotron Nano 3 model (NVIDIA, 2025a) trained to 25T tokens in BF16. Right: Ablation study trained to 1T tokens in BF16 and with our NVFP4 recipe. The NVFP4 model at 1T tokens reaches a similar zero-valued weight gradient count as the BF16 model at 25T tokens. Switching from NVFP4 to BF16 at 0.5T tokens causes zero-valued weight gradients to return to baseline levels. The high prevalence of small-magnitude gradients (<1e-12) in BF16 suggests NVFP4 quantization underflows already-small values to zero.

We compare identical Nemotron 3 Nano models (NVIDIA, 2025a) trained for 1T tokens in BF16 and in NVFP4 and find that NVFP4 pretraining produces roughly 3x more zero-valued weight gradients at the same token horizon. When a partially trained NVFP4 model is switched back to BF16, the number of zero-valued weight gradients returns to baseline levels. The BF16 model still contains many small-magnitude gradients (<1e-12), but NVFP4 quantization underflows these values to zero (Figure 7). We sampled weight, activation, and gradient tensors from routed expert layers and observed high rates of underflow in dgrad of FC2 at 500B tokens, primarily because two-dimensional weight quantization blocks span high and low magnitude channels. Underflows in dgrad of FC2 create zeros in wgrad of FC1 through backpropagation of the gradient. At 750B tokens, we observe high rates of underflow in fprop of FC1, creating zeros in wgrad of FC2 (Figure 8). The 1T-token NVFP4 model behaves similarly to a much longer-trained BF16 model. After 10T tokens, the released Nemotron 3 Nano model, trained in BF16, NVIDIA (2025a) reaches a similar number of zero-valued weight gradient elements as the NVFP4 model trained to 1T tokens (Figure 7) and inspection of weight matrices in early expert layers revealed a similar channel magnitude pattern. These conclusions on the Nemotron 3 Nano architecture give insights into the channel magnitude patterns and growth in zero-valued weight gradient elements in Nemotron 3 Super.

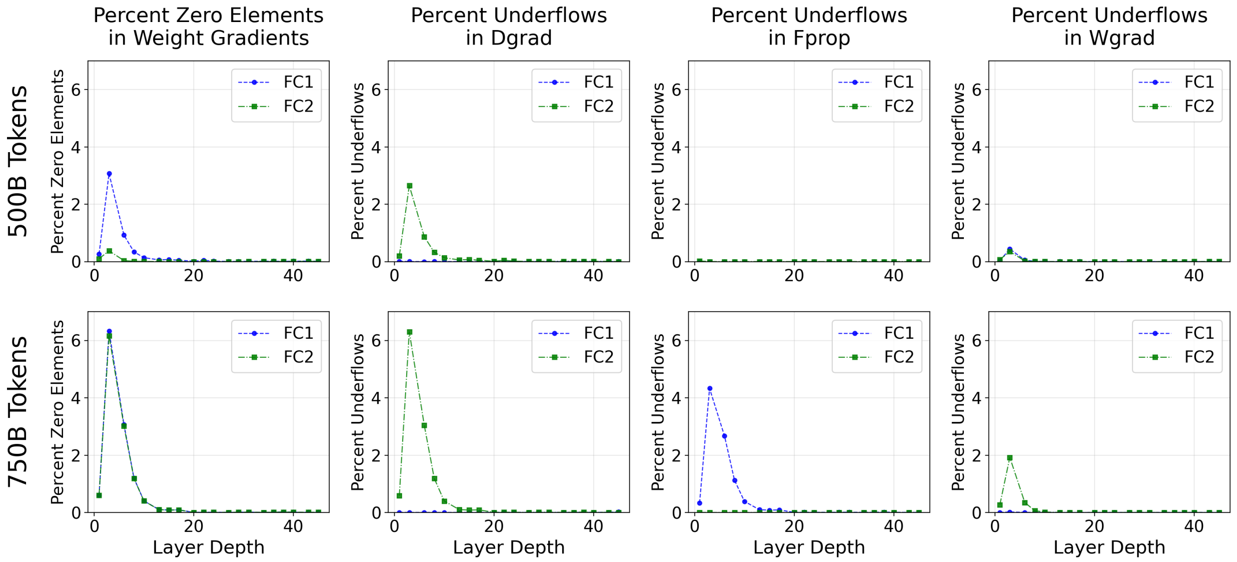


Figure 8 | Origin of Zero Element Weight Gradients in Nemotron 3 Nano, shown for routed expert layers increasing in layer depth. Averaged across all routed experts in a layer index. Top: Tensors sampled at 500B tokens. Percent of zero-valued weight gradients are higher for FC1 than FC2 and attributed almost entirely to underflows in dgrad of FC2. Bottom: Tensors sampled at 750B tokens. Percent of zero-valued weight gradients are equivalently high for FC1 and FC2. Zero-valued weight gradients in FC1 are attributed to underflows in dgrad of FC2. Zero element weight gradients in FC2 are attributed mainly to underflows in fprop of FC1, with a minor contribution of underflows in wgrad of FC2.

Following our previous work in NVFP4 pretraining (NVIDIA, 2025d), we evaluated whether switching all tensors to higher precision prior to learning rate annealing would benefit Nemotron 3 Super. We promoted all tensors to MXFP8 at 19T tokens (1T tokens before annealing) and continued training through 20.6T tokens. While this improved the loss trajectory, it yielded no gains in downstream task accuracy (Fig 9). The final Nemotron 3 Super model is therefore pretrained with our NVFP4 recipe for the entire token horizon.

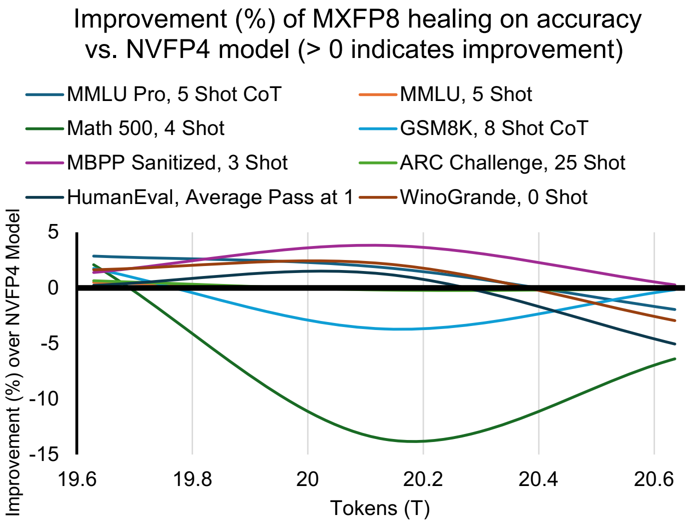


Figure 9 | Improvement in downstream task evaluation accuracy after switching the network precision to MXFP8. Values greater than zero indicate improvement in accuracy over the NVFP4 model. None of the downstream task evaluation metrics showed sustained improvement after training in MXFP8.

2.3. Pretraining Data

2.3.1. Data

We describe here several new datasets we added to pretraining since Nemotron 3 Nano (NVIDIA, 2025a). We are releasing these datasets on HuggingFace as `Nemotron-Pretraining-Specialized-v1.1`.

2.3.2. Synthetic Code Concepts

With the aim of improving Python problem-solving capabilities, we synthetically generated a dataset consisting Python problems and solutions. Using a taxonomy consisting of thousands of programming concepts curated from our `Nemotron-Pretraining-Code` datasets and GPT-OSS-120B, we extracted high-level programming concepts from the HumanEval benchmark dataset (Chen et al., 2021). In total, after deduplication of the extracted taxonomical representations, we collected a total of 91 concepts.

Using the extracted concepts, we performed open-ended generation using GPT-OSS 20B to generate Python programming problems that test these concepts and instructed it to generate the problem with a descriptive function name and problem description in the function docstring. To generate these problems at the pretraining scale, we combined up to four concepts per generation and generated up to five problems per set of concepts. This resulted in a total of approximately 14 million problems.

Following problem generation, we then used GPT-OSS 120B to generate five self-contained solutions for each generated problem. To avoid biasing models trained on these data to generate long-winded solutions, we instructed GPT-OSS 120B to restrict its solution to 60 lines maximum. For each problem, we generate five solutions, and we stopped generating after obtaining approximately 23 million problem-solution pairs.

As a final step in generating this dataset, we thoroughly cleaned the generated problem-solution pairs. Our cleaning consisted of the following steps: we check that GPT-OSS-120 B did not include additional imports that were not specified in the original problem generated from GPT-OSS-20 B. We discard all solutions that did not satisfy this condition. We form the final problem-solution pair by parsing only the solution provided by GPT-OSS-120B and appending it to the problem generated from GPT-OSS-20 B. We found that GPT-OSS-120B frequently modified the original problem and this ensured we maintained the desired format prescribed in our original problem-formation prompt. We check that the final problem-solution pair is valid Python code via generation of an abstract-syntax tree (AST). If the final function does not pass the final AST check, it is discarded. After the above cleaning procedure, we resulted in the 15 M problems that make up the dataset.

2.3.3. Synthetic Unconditional Algorithmic

To create this dataset, we generated algorithmic Python problems using Qwen3-235B-A22B (the base model) and gpt-oss-120b. We used minimalistic prompts—such as “Write a function,” “Write a Python function,” or “Write a coding problem and solution for a student to solve”—and optionally specified a difficulty level (easy, medium, or hard). To ensure diversity and quality, we prompted gpt-oss-120b to rewrite these samples to handle edge cases, add unit tests, and reformat the outputs in various ways.

In another variant, we instructed gpt-oss-120b to generate LeetCode-style questions and answers, again with a randomly selected difficulty level. We further used gpt-oss-120b to score the correctness of the solution and, if incorrect, to correct it. Overall, such nearly unconditional prompting did result in high rates of duplicates. To combat this, we found it effective to deduplicate based on short titles of around 5–8 words generated by gpt-oss-120b for each problem.

All samples were decontaminated against HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), CRUXEval (Gu et al., 2024), and LiveCodeBench (Jain et al., 2024) as follows: First, exact matches of the solution against those benchmarks were removed. Second, we used Qwen3-Embedding-0.6 to encode Problem and Solution and filtered any data with >0.8 similarity with any of the benchmarks.

Although this dataset is small by usual pretraining standards (0.2B tokens), we believe it helps to teach coding practices like edge case handling and reasoning about program execution, as evidenced by improvements of 1-2 points to HumanEval, MBPP, and CRUXEval-O over the Nemotron 3 Nano base checkpoint, when adding these datasets to a redo of the last 100B tokens of 25T token pretraining.

2.3.4. Synthetic Economics

We generated a diverse set of economics multiple-choice questions across various formats, including cloze, calculation, sentence completion, and multiple-response, covering key topics and terms in microeconomics, macroeconomics, and econometrics (e.g., “Statistical Inference and Hypothesis Testing - Type I error” and “Inflation and the Price Level - Inflation rate”) from a curated list. For each topic-term pair, we used Qwen3-235B-A22B-Thinking-2507 to generate multiple questions, each accompanied by a detailed, step-by-step, and well-formatted solution. To enhance the diversity, we further prompted the model to create new and original questions using the initial outputs as reference points. Each question-solution pair underwent model-based verification for clarity, ambiguity, solvability and accuracy.

2.3.5. Synthetic Formal Logic

We synthesized a set of formal logic problems and solutions spanning several tasks, such as translating between natural language and predicate or propositional logic, deriving the antecedents of conditional propositions, and solving logic problems using indirect or complete truth tables. We introduced variability into the generated scenarios, premises, and formulas by incorporating random personas,[2] letters, and/or logic connective (i.e., $\wedge$, $\vee$, $\supset$, $\equiv$, $\sim$) into the prompt. We generated and evaluated the problems and solutions using Qwen3-235B-A22B-Thinking-2507.

2.3.6. Synthetic Multiple Choice

We construct a multiple-choice question (MCQ) dataset by bootstrapping from the MMLU auxiliary training set (Hendrycks et al., 2021b), which aggregates auxiliary MCQ data from sources such as ARC (Clark et al., 2018b), MC_TEST (Richardson et al., 2013), OpenBookQA (Mihaylov et al., 2018a), and RACE (Lai et al., 2017), etc. Starting from each seed question, we generate multiple similar questions that follow the same task format and difficulty profile, along with corresponding answer options by prompting Qwen3-235B-A22B (Yang et al., 2025). In the second stage, we prompt the DeepSeek-V3 (DeepSeek-AI, 2025c) model to solve each generated question by selecting an answer and providing the supporting knowledge or contextual reasoning underlying its choice. To improve answer reliability, we sample multiple independent solution generations for each question using different random seeds. We then apply majority voting over the generated answers to identify the most consistent choice, retaining only those samples whose final answer agrees with the majority and discarding inconsistent or incorrect instances.

Using this pipeline, we generate approximately 3.5M MMLU-style MCQ samples (~1.6B tokens) augmented with explicit, relevant knowledge or reasoning traces. We evaluate the impact of this data via ablation experiments by continued training the Nemotron-Nano-V3 (NVIDIA, 2025a) 24.9T-token checkpoint with an additional 100B tokens, out of which 1B tokens are from the generated MMLU-aux-train-SDG data. The results show consistent gains on most of the benchmarks: MMLU improves from 77.22 to 77.51, "MATH Level 5" from 78.55 to 79.05, AIME-2024 improved from 53.3 to 56.7, and MBPP from from 74.8 to 75.2. Performance on other benchmarks remains largely stable with only minor variance, indicating that the synthesized MCQ data primarily strengthens mathematical and structured reasoning capabilities without introducing regressions elsewhere.

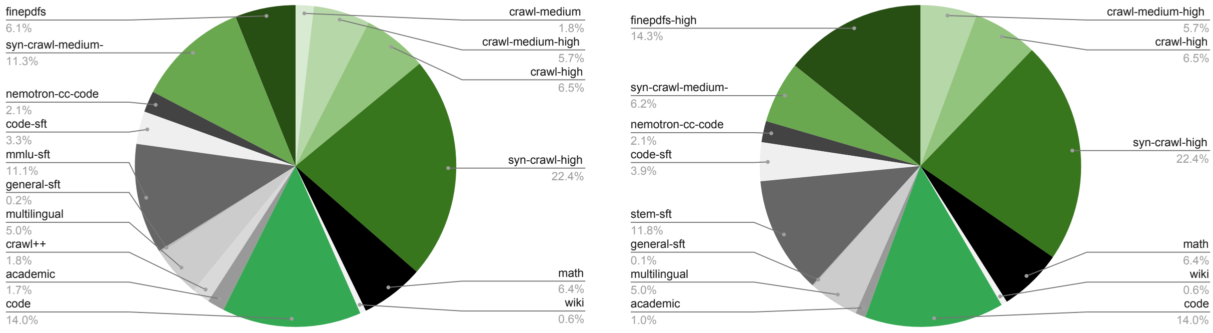
2.3.7. Data Mixture and Ordering

We adopt the Nemotron 3 Nano data mixture as described in (NVIDIA, 2025a). Our pretraining corpus spans 16 high-level categories. The largest component is web crawl data, which we partition into five quality-based groups following the Nemotron-CC taxonomy (Su et al., 2025): crawl-medium, crawl-medium-high, and crawl-high, representing progressively higher-quality crawl data, along with their synthetic counterparts, syn-crawl-medium-high and syn-crawl-high, generated from filtered web documents. Beyond web crawl, the mixture includes math (Mahabadi et al., 2025; Akter et al., 2024), Wikipedia, code, Nemotron-CC-Code, academic text, Crawl++, multilingual data, finepdfs (Kydlíček et al., 2025) and synthetic SFT-style datasets. The SFT-style data is further divided into general-sft, stem-sft, and code-sft. As part of the SFT-style component, we incorporate reasoning-focused datasets into pretraining, motivated by prior findings demonstrating their effectiveness (Akter et al., 2026). Crawl++ consists of OpenWebText, BigScience (Laurençon et al., 2023), and Reddit datasets.

Data blending is designed to balance diversity and quality: sources with comparable estimated quality are assigned similar weights, while higher-quality datasets receive proportionally greater weight in the mixture. Further details on dataset quality estimation and mixture construction are provided in (Feng et al., 2024). We adopt the two-phase curriculum proposed in (Feng et al., 2024) work. In Phase 1, the mixture emphasizes data diversity to promote broad coverage and generalization. In Phase 2, the blend shifts toward predominantly high-quality sources (e.g., Wikipedia) to refine

[2] https://huggingface.co/collections/nvidia/nemotron-personas

model performance. The transition to Phase 2 occurs at 80% of total training tokens. The specific mixtures used in each phase are illustrated in Figure 10.



(a) Data mixture of Phase 1.

(b) Data mixture of Phase 2.

Figure 10 | Data mixtures for each phase of pre-training.

2.4. Hyperparameters

The pretraining of Nemotron 3 Super 120B-A12B Base was conducted using a Warmup-Stable-Decay (WSD) (Hu et al., 2024) learning rate schedule over a total horizon of 25 trillion tokens. The learning rate (LR) was warmed up over the initial 200 billion tokens to a peak value of 4.5×10^{-4}. Following a sustained stable plateau phase, we implemented a `minus-sqrt` decay schedule for the final 5 trillion tokens, annealing the LR to a minimum of 4.5×10^{-6}.

We used AdamW (Loshchilov & Hutter, 2017) optimizer with a weight decay of 0.1 and momentum coefficients $\beta_1 = 0.9$ and $\beta_2 = 0.95$. The model was trained with a sequence length of 8,192 and a batch size of 3,072 sequences, resulting in approximately 25.17 million tokens per batch.

The architecture employs a hybrid Mamba-MoE design, featuring Mixture-of-Experts (MoE) layers with 512 total experts and a top-22 routing mechanism ($k = 22$). We utilized a sigmoid router score function complemented by expert biasing. To ensure equitable expert utilization across the 120.6B parameters, we adopted an auxiliary-loss-free load balancing strategy (Wang et al., 2024; DeepSeek-AI, 2025c) with an update rate of 10^{-3}, paired with a standard load balancing loss with coefficient of 10^{-4} (Lepikhin et al., 2020).

Furthermore, we used an MTP objective with loss scaling factor of 0.3. To maximize computational efficiency and training stability at scale, the execution utilized a hybrid precision scheme of BF16 and NVFP4.

2.5. Tracking Merge Evaluation

During the stable phase of the WSD learning rate schedule described in Section 2.4, the learning rate remains constant, and individual trained checkpoints exhibit noisy benchmark performance from step to step. Following recent work on weight-space merging (Wortsman et al., 2022; Tian et al., 2025; Ling Team, 2025), we apply checkpoint merging (weighted averaging over a sliding window of recent checkpoints) to produce stronger readouts of model quality without requiring dedicated learning rate decay runs. In a conventional pretraining workflow, evaluating model quality at intermediate checkpoints requires dedicated decay runs; checkpoint merging eliminates this cost. For a schedule comparable to ours, the savings could reach ∼4T tokens of compute (e.g., ∼2 avoided runs at 1.5T and ∼2 at 0.5T), or roughly 16% of the total pretraining FLOP budget.

Following Tian et al. (2025), we use a `minus-sqrt` decay emulation to compute merge coefficients, with checkpoints saved every 1,000 iterations (≈25B tokens at our global batch size of 3,072 × 8,192 tokens). We evaluated sliding merge windows of 125B, 250B, and 500B tokens over the course of pretraining. On average, across a suite of 12 benchmarks (MMLU-Pro, MMLU, HumanEval, HumanEval+, MBPP, MBPP+, GSM8K, MATH-500, RACE, ARC-Challenge, HellaSwag, WinoGrande), the best merge consistently outperforms the corresponding trained checkpoint by 2–4 points on the unweighted average. Since merging is computationally cheap relative to training, we can evaluate all three windows at each checkpoint and select the best. Figure 11 reports this best-of-three merge against the trained checkpoint over the full 25T-token training run.

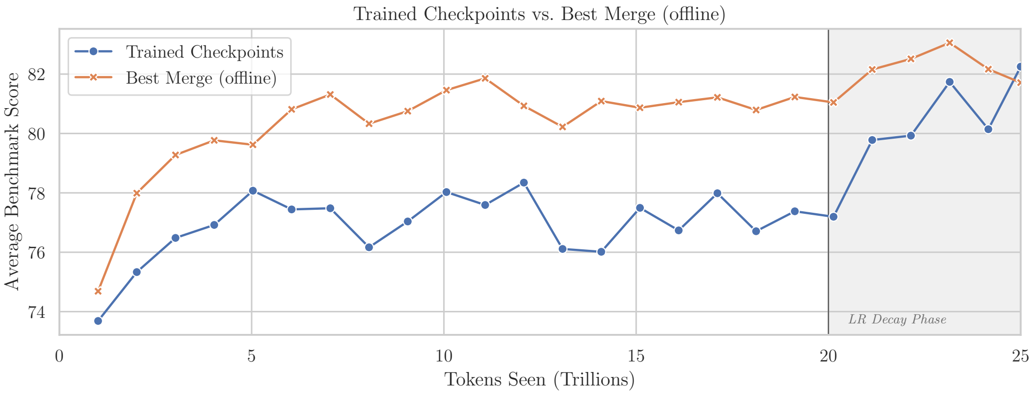


Figure 11 | Average accuracy across 12 benchmarks for trained checkpoints versus the best offline checkpoint merge during pretraining. During the stable LR phase, offline merging yields a consistent 2–4 point improvement. During the LR decay phase (shaded), the gap narrows as trained checkpoints benefit from actual learning rate annealing.

During the final 5T-token LR decay phase (from 20T to 25T tokens), the gap between merged and trained checkpoints narrows substantially, and the two evaluation trajectories largely coincide by the end of training. Tian et al. (2025) reported that combining merging with decay offers no gain over merging alone, so this convergence is expected. The original WSM results went further, showing merge-based readouts *surpassing* decay-trained checkpoints. In experiments on the Nemotron 3 Nano scale architecture (30B-A3B), we were able to reproduce such gains when emulating short (∼500B) decay windows. However, direct comparisons at 1T and 1.5T merge horizons showed no improvement over decay-trained checkpoints.

Our takeaway is that offline checkpoint merging appears most effective for shorter annealing horizons. This is consistent with Ling Team (2025), who employ a comparably short decay schedule and report merge-based improvements, in contrast with the much longer 5T decay used here, where trained decay is able to match or surpass merge-based readouts. The final base model checkpoint selected for downstream alignment was itself a 500B merge; short-horizon merging remains practically useful even alongside a full decay schedule. That said, our experiments explored only a single merge schedule (`minus-sqrt`) and a fixed checkpoint granularity; it remains plausible that alternative coefficient schemes, finer-grained checkpoint windows, or merging strategies tailored to longer decay horizons could recover the gains observed at shorter scales. Per-benchmark breakdowns are provided in Appendix Figure 17.

2.6. Long-Context Extension

Similar to Nemotron 3 Nano, we added a long-context phase (LC-Phase) at the end of pretraining. In the LC-Phase, we performed continuous pretraining (CPT) to equip the base model with long-context ability. We used a constant learning rate of $4.5 * 10^{-6}$ and global batch size of 16. We used 64-way context parallelism, 2-way tensor parallelism, and 64-way expert parallelism to train on GB200 GPUs. We reused the long-context document QA dataset from Nemotron 2 & 3 Nano. We allocated the document QA data to 20% in the Phase LC data blend, with the remaining 80% being downscaled Phase 2 data. We initially performed CPT on 1,048,576 (1m) context length. Such stage lasted for 34 billion tokens. Following that we added another stage to alternatingly train on both 1m and 4k sequences in order to mitigate the minor impact we observed on the math-related benchmarks. The second stage lasted for 17 billion tokens.

2.7. Base Model Evaluations

All evaluation results were collected via Nemo Evaluator SDK[3] and NVIDIA's open source container of LM Evaluation Harness[4], unless otherwise stated. For reproducibility purposes, more details on the evaluation settings can be found in the Nemo Evaluator SDK examples folder[5]. The open source container on LM Evaluation Harness packaged via NVIDIA's Nemo Evaluator SDK used for evaluations can be found here[6]. This container is is built on top of LM Evaluation Harness, with the following changes applied to all models for fairness:

1. For mathematical reasoning, we evaluate GSM8K and MATH (Cobbe et al., 2021; Hendrycks et al., 2021c) benchmarks using greedy-decoding. We also highlight the competition-level slice of the MATH benchmark as "MATH Level 5". Additionally, we report the pass@32 performance on AIME-2024. We use `Math-Verify`[7] to grade all generations.
2. For code tasks (HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021)) we evaluate the EvalPlus variants along with the sanitization of generations (Liu et al., 2023), in a 0-shot setup. We estimate avg@32, pass@1 from 32 generations per prompt.
3. General reasoning benchmarks (OpenBookQA (Mihaylov et al., 2018b), PIQA (Bisk et al., 2019), Hellaswag (Zellers et al., 2019), Winogrande (Sakaguchi et al., 2019)) are unchanged except for ARC-Challenge (Clark et al., 2018a), where we present all options at the same time, similar to MMLU (Hendrycks et al., 2021a).
4. For multilingual capability, we evaluate MGSM (Shi et al., 2022) (8-shot, native CoT) and Global MMLU-Lite (Singh et al., 2024).

Accuracy results for Nemotron 3 Super 120B-A12B Base with comparsions to Ling-flash-Base-2.0 and GLM-4.5-Air-Base are shown in Table 4.

3. Post-Training

We follow the same general recipe as Nemotron 3 Nano, with a stronger emphasis on agentic tasks. An overview of our pipeline is given in Figure 12. We start with Supervised-Finetuning (SFT) phase with details in §3.1 followed by the three stage Reinforcement Learning comprising of RLVR,

[3] https://github.com/NVIDIA-NeMo/Evaluator

[4] https://github.com/EleutherAI/lm-evaluation-harness

[5] https://github.com/NVIDIA-NeMo/Evaluator/tree/main/packages/nemo-evaluator-launcher/examples/nemotron/nemotron-3-super

[6] https://catalog.ngc.nvidia.com/orgs/nvidia/teams/eval-factory/containers/lm-evaluation-harness

[7] https://github.com/huggingface/math-verify.

Task	**Metric**	**N-3-Super 120B-A12B-Base**	**Ling-flash base-2.0**	**GLM-4.5 Air-Base**
General Knowledge				
MMLU	*5-shot, acc*	**86.01**	81.00	81.00
MMLU-Pro	*5-shot, CoT EM*	**75.65**	62.10	58.20
AGIEval-En	*3/5-shot, CoT EM*	**77.92**	61.70	62.40
GPQA-Diamond	*5-shot, CoT EM*	**60.00**	36.00	23.20
MATH				
GSM8K	*8-shot, EM*	90.67	**90.75**	82.60
MATH	*4-shot, EM*	**84.84**	63.80	50.36
MATH Level 5	*4-shot, EM*	**70.00**	39.80	26.30
AIME 2024	*pass@32*	**53.33**	30.00	20.00
Code				
HumanEval	*0-shot, pass@1 n=32*	**79.40**	70.10	76.30
MBPP-Sanitized	*3-shot, pass@1 n=32*	**78.38**	77.30	77.50
Commonsense Understanding				
ARC-Challenge	*25-shot, acc_norm*	**96.08**	94.80	93.90
HellaSwag	*10-shot, acc_norm*	**88.97**	84.69	87.70
OpenBookQA	*0-shot, acc_norm*	**50.20**	47.00	48.60
PIQA	*0-shot, acc_norm*	**85.47**	84.00	84.22
WinoGrande	*5-shot, acc*	78.93	78.37	**83.82**
Reading Comprehension				
RACE	*0-shot, acc*	**91.00**	90.10	89.50
Multilingual				
MMLU Global Lite	*5-shot, avg*	**85.72**	74.94	79.25
MGSM	*8-shot, avg*	**87.47**	82.73	80.33
Long Context				
RULER 64K	*0-shot*	**93.17**	-	81.58
RULER 128K	*0-shot*	**89.00**	57.56	63.62
RULER 256K	*0-shot*	**86.18**	-	-
RULER 512K	*0-shot*	**82.16**	-	-
RULER 1M	*0-shot*	**74.39**	-	-

Table 4 | Comparison of **Ling-flash-base-2.0**, **GLM-4.5-Air-Base**, and **Nemotron Super 120B-A12B Base**. Best results are marked in bold.

SWE-RL and RLHF in §3.2. We also do a final phase of MTP. In SFT, we expand the training blend to cover a wider range of agentic harnesses and interaction scenarios. We also significantly improved our RL infrastructure, enabling reliable large-scale asynchronous training on thousands of GPUs. This infrastructure allows us to (1) train across 21 diverse environments, improving robustness across tasks, and (2) train on long-horizon SWE tasks, strengthening multi-step reasoning and problem solving in realistic agentic settings.

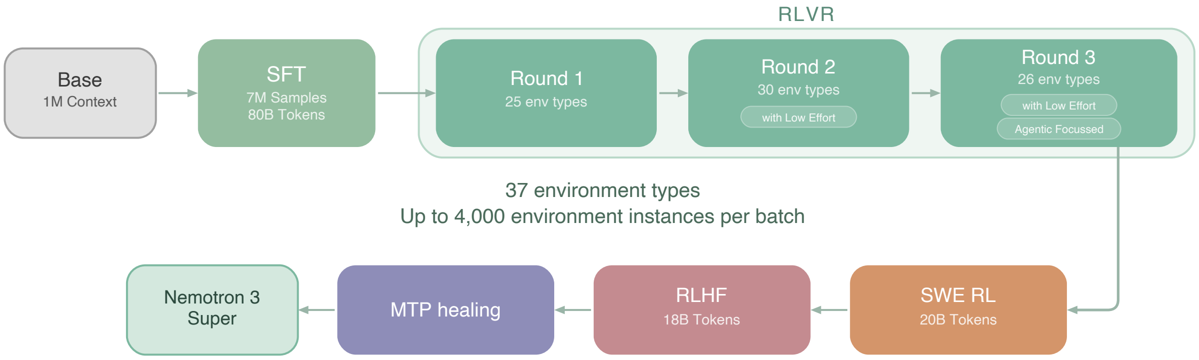


Figure 12 | Overview of the post-training pipeline for Nemotron 3 Super.

3.1. Supervised Fine Tuning

For Nemotron 3 Super SFT, we focused on improving dataset quality and diversity. In particular, we scaled up our agentic datasets and increased their share in the overall SFT blend. The chat template remains identical to Nemotron 3 Nano. In addition, we add low effort reasoning mode, giving users further control over reasoning length. We found that a single-stage SFT led to a marked degradation on long-input-short-output scenarios. We therefore adopt a two-stage SFT procedure: Stage 1 emphasizes learning from token-level supervision and induces strong reasoning behavior, while Stage 2 switches to per-conversation normalization to prevent long outputs from dominating the loss, which restores long-input-short-output performance while retaining reasoning. We describe it below:

SFT objective and two-stage loss

For a packed global batch $\mathcal{B}$ containing multiple conversations c, let $\mathcal{O}_c$ denote the set of output-token positions for conversation c and $|\mathcal{O}_c|$ its output-token count. With token-level negative log-likelihood $\ell_t = -\log p_\theta(y_t \mid x, y_{<t})$, we use:

Stage 1: token-level (global) average. We minimize the average loss over *all* output tokens in the packed global batch:

$$\mathcal{L}_{\text{tok}} = \frac{\sum\limits_{c\in\mathcal{B}} \sum\limits_{t\in\mathcal{O}_c} \ell_t}{\sum\limits_{c\in\mathcal{B}} |\mathcal{O}_c|}. \tag{1}$$

This corresponds to summing the output-token log probabilities across all conversations and normalizing by the total number of output tokens.

Stage 2: sample-level average. We then switch to a per-conversation normalized loss and average equally across conversations:

$$\mathcal{L}_{\text{samp}} = \frac{1}{|\mathcal{B}|} \sum_{c\in\mathcal{B}} \left(\frac{1}{|\mathcal{O}_c|} \sum_{t\in\mathcal{O}_c} \ell_t \right). \tag{2}$$

This stage reduces the dominance of long outputs by normalizing each conversation by its own output-token count before averaging across the batch.

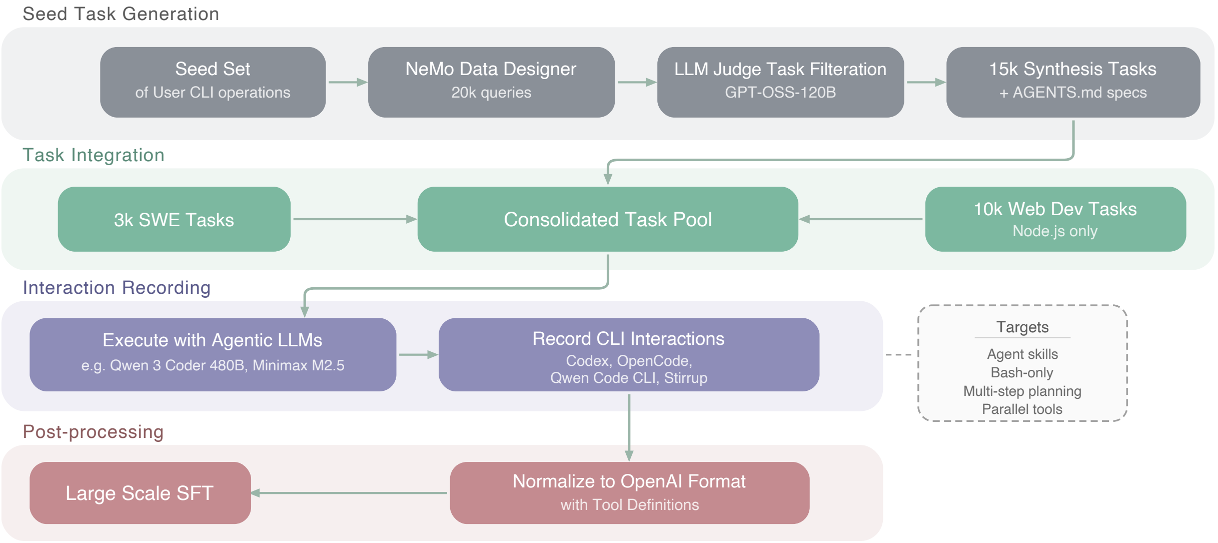


Figure 13 | Agentic Command Line Interface Dataset Construction & Training Pipeline

For Stage 1, we run SFT with 256k sequence length packing, global batch size 64, constant lr $1e-5$ with 30k warmup samples. For Stage 2, we use 512k sequence length packing and include long context data with length up to 512K, global batch size 32 and constant lr $1e-5$.

MTP during SFT We continue training Nemotron 3 Super with the same shared-weight MTP head used in pretraining to preserve both the accuracy benefits of multi-step prediction and the inference-time gains from speculative decoding. Concretely, we train two MTP layers with shared parameters and optimize the combined objective using a scaled auxiliary loss computed with per-token loss and 0.3 scaling factor.

3.1.1. Data

We reuse the following datasets from the Nemotron 3 Nano SFT datasets: Chat, Infinibyte, and Formal Proofs. We refresh the following datasets with new teacher models (`DeepSeek v3.2, Kimi K2`): Competition Math, Competition Code, Conversational Tool Use, Multilingual, Science. Below we describe new or heavily modified SFT datasets.

Software Engineering. We curate a dataset of coding tasks derived from real-world GitHub issues to train Nemotron 3 Super for autonomous software engineering capabilities including code exploration, task tracking, issue reproduction and bug fixing. We use the issues and containerized execution environments from the SWE-Gym (Pan et al., 2025), R2E-Gym (Jain et al., 2025) and SWE-rebench (Badertdinov et al., 2025) datasets. For R2E-Gym, we regenerate problem statements with `Qwen3-Coder-480B-A35B-Instruct`. We distill trajectories from the OpenHands agent harness using `Qwen3-Coder-480B-A35B-Instruct` as the teacher model.

Agentic Programming.

The landscape of software development has undergone a substantial shift with the emergence of Agentic Command Line Interface (CLI) tools, moving beyond the "autocomplete" era of 2021–2023 into a regime of autonomous execution. Alongside substantial improvements in harnesses such as Claude Code, OpenCode, and OpenAI's Codex, models are now capable of operating as active digital collaborators, capable of multi-step reasoning, long-horizon execution, and end-to-end task orchestration.

We established a foundational seed set of tasks designed to replicate common user-initiated operations within agentic CLIs. Refer to Figure 13 where we discuss the full pipeline. We utilized NeMo Data Designer (The NeMo Data Designer Team, 2025) to generate approximately 20k queries derived from a taxonomy of 24 distinct actions typically performed in these environments. Subsequently, we employed `GPT-OSS 120B` (OpenAI, 2025) in an LLM-as-a-Judge framework to filter out tasks referencing pre-existing codebases or modifications to extant files. This mitigation ensures that the models do not attempt modification operations within empty directories—a scenario that frequently results in redundant and exhausted tool invocations during failed execution cycles. The resulting dataset comprises roughly 15k tasks centered on direct solution synthesis. To further enhance the diversity of the generated outputs, we coupled each task with a supplementary markdown specification, equivalent to an AGENTS.md file. These documents impose additional constraints and architectural requirements, effectively narrowing the design space and necessitating more complex, varied solutions from the agent.

Since we eliminate all tasks that require a pre-existing codebases, we augment this task set with roughly 3000 questions from SWE tasks that are challenging and come with pre-existing repository and specific git hash commit. We apply a simple prompt prior to the existing issue statements, stating that the execution environment does not have the library installed and therefore execution of unit tests should be avoided. This is a conscientious decision made to substantially reduce the engineering effort required to support per sample container based execution of each SWE task, reduce the number of tool calls and avoid large tool outputs due to multiple rounds of unit test executions. This further relaxes the constraints imposed on the agent to solve the issue, as there is no SWE specific prompt to guide the agent in solving the task. Finally, we synthesize 10k web development tasks using a taxonomy of 100 fine-grained tasks that are commonly requested by users as the seed, on which we apply LLM-as-a-Judge to eliminate tasks that require a pre-existing repository. We impose no restrictions on these tasks, but provide only a Node.js environment and expect the agent to setup and install all dependencies and plugins on its own.

Applying these task sets, we distill from high-performance, open-source agentic LLMs such as `Qwen-3-Coder-480B` (Qwen, 2025) and `Minimax M2.5` (MiniMax AI, 2025) by recording their interactions with various CLI environments, such as Codex, OpenCode, Qwen Code CLI, and Stirrup. These interaction traces are subsequently filtered, normalized into the standard OpenAI message format with various tool definitions, and utilized for large-scale SFT to effectively embed agentic operational knowledge within the model. For each Agentic CLI, we study the individual capabilities that exist and are commonly utilized. We apply the same task sets to target different capabilities, such as Agent Skills, tool restriction (bash only execution), ask user clarifying questions, single and multi step planning, static and dynamic multi turn conversations and parallel tool calling, depending on whether the specific CLI can accommodate these capabilities.

Long Context. We extend the long-context SFT dataset from Nemotron 3 Nano with a more comprehensive synthetic data pipeline. To improve long-context multi-document reasoning, we construct a synthetic SFT dataset using long sequences from our pre-training blend, which contains books, papers, financial reports, code repositories, etc. We first cluster these documents by topic/domain and concatenate related documents to reach target sequence lengths, such as 128K, 256K, or 512K tokens. For each long-context sample, we use an LLM to generate one or more QA pairs. The prompt requires questions to involve cross-document or cross-section navigation, ensuring information is scattered rather than localized. It strictly enforces multi-hop reasoning, requiring at least 4 to 7 distinct retrieval or reasoning steps. These steps mandate computational or logical processing, preventing simple copy-pasting, and often include explicit formatting instructions. Next, we generate 8 independent reasoning traces for each context-question pair. We apply semantic majority voting to group the answers, either by exact match or via an LLM judge. From the resulting majority group, we select the answer containing the shortest reasoning trace. Additionally, we generate seven synthetic reasoning tasks to improve the model's ability to process context in a sequential, left-to-right manner. Specifically, synthetic snippets (generated using `Qwen3-235B-A22B-Thinking-2507`) are concatenated together to form long input context. The thinking traces are constructed by chaining rule-based reasoning steps, each of which includes relevant excerpts from the input context and tracking metadata, such as the frequency of query-related snippets. We also construct long-context samples by concatenating records from (Meyer & Corneil, 2025) to reach the required sequence length. Questions are designed to emphasize multi-hop reasoning and information aggregation across records. Context, question, and answer are formatted using pre-defined templates, with ground-truth answers derived by executing SQL queries over the underlying records.

Financial Reasoning. To construct a large-scale training corpus for financial reasoning, we employ a template-based synthetic data generation (SDG) pipeline that scales a curated seed set into hundreds of thousands of grounded question–answer pairs. The pipeline sources 565 expert-authored seed questions from the SecQue benchmark (BenYoash et al., 2025), a dataset of financial analysis questions anchored to SEC 10-K and 10-Q filings. These seeds are expanded combinatorially across S&P 500 companies[8] and fiscal years (2019–2024), where comparative questions are restricted to company pairs within the same GICS Sub-Industry to preserve semantic coherence. GPT-OSS-120B paraphrases each template instantiation, producing up to three diverse reformulations per combination. The resulting questions are mapped to relevant SEC filing sections using the original SecQue metadata, and the corresponding documents are converted to markdown with a configurable token limit. For answer generation, we adopt the GenSelect strategy (Toshniwal et al., 2025): five candidate answers are sampled per question using GPT-OSS-120B with distinct random seeds, and a larger judge model (Qwen3-235B-A22B) selects the best response based on numerical accuracy, financial methodology, and logical soundness. A smaller model (Qwen3-30B-A3B) then classifies each pair as `ANSWERABLE` or `UNANSWERABLE`, retaining only those containing a complete, substantive response. Prior to supervised fine-tuning, the SDG output undergoes percentile-based outlier removal and deduplication. The resulting dataset comprises 366,243 financial Q&A pairs with reasoning traces.

CUDA. A large-scale synthetic CUDA dataset comprising 100K samples for kernel generation, repair, and optimization was constructed using a synthetic data generation pipeline based on `DeepSeek-R1` and `GPT-OSS-120B`. Seed questions were sourced from popular open-source libraries, NVIDIA library API surfaces, and BackendBench (Saroufim et al., 2025). These seeds were used to generate tuples of the form (PyTorch reference, CUDA C++ kernel) and (natural language specification, CUDA C++ kernel), each accompanied by reasoning. For each seed item, multiple candidate kernels were generated and rigorously validated for correctness within an internal CUDA evaluation environment. The validated kernels were then ranked by performance, and the highest-performing kernel was retained. In addition, we collected traces from an internal CUDA agent, producing samples of the form (PyTorch reference, faulty CUDA C++ kernel, error message, corrected CUDA C++ kernel) and (PyTorch reference, slow CUDA C++ kernel, Nsight Compute log, optimized CUDA C++ kernel). Using publicly available documentation and official code samples, and following the same formulation as the CUDA-C data, we generated additional PyTorch references and corresponding CUDA-library implementations with reasoning chains, as well as aligned natural language specifications. These libraries include Thrust, CUB, cuBLAS, cuDNN, cuSPARSE, cuRAND, and cuSOLVER.

[8] `https://en.wikipedia.org/wiki/List_of_S%26P_500_companies`, accessed 2025.

Safety. We have significantly enhanced our safety framework compared to Nemotron 3 Nano by combining a robust prompt library with a two-stage synthetic response generation strategy. While retaining the core prompts from Nemotron Content Safety v2 (Ghosh et al., 2025), Gretel Safety Alignment v1 (Gretel, 2024), Harmful Tasks (Hasan et al., 2024) and Red-Team-2K (Luo et al., 2024) covering content safety and common jailbreak techniques, we added in new synthetic prompts targeting the elicitation of over-refusals, demographic biases, and copyright reproduction. We also expand coverage of jailbreak strategies to better capture emerging adversarial techniques including indirect prompt injection attacks.

Our primary advancement over Nemotron 3 Nano is an explicit response policy framework. For each prompt, a response policy is inferred from a combination of prompt metadata, its annotated safety category, and predictions from a set of lightweight auxiliary classifiers that detect attributes such as self-harm risk, demographic targeting, or the presence of embedded adversarial instructions. We cast this decision process as a multi-class classification problem, where each class corresponds to a distinct response mode aligned with safety guidelines. These response modes specify whether the model should provide supportive resources such as helpline information, issue a refusal with a brief explanation, or answer the benign portion of the request while ignoring malicious content. This ensures that the responses are safe, contextually appropriate, and consistent across a diverse set of safety-sensitive scenarios.

Following the deliberative alignment framework (Guan et al., 2025), we adopt a two-stage generation process in which the reasoning trace and the final response are curated separately but consistently with our response policy. In the first stage, we construct a concise reasoning trace that guides the model to reflect on the safety properties of the prompt, explicitly identifying why the request may be unsafe or policy-relevant and what constraints should govern the response. In the second stage, we generate the final response based on this reasoning trace, ensuring that it adheres to the predefined response policy and behavior guidelines. This structured separation encourages deliberate reflection on safety guidelines while producing responses that are consistent, policy-compliant, and contextually appropriate to ensure that the final response follows safety policies while minimizing unnecessary references to said policies. Finally, we apply a content-moderation classifier to filter any responses flagged as unsafe, providing an additional safeguard to ensure alignment with safety objectives.

Search. To improve search capabilities, we generated a synthetic search-agent SFT dataset using NeMo Data Designer (The NeMo Data Designer Team, 2025). The pipeline begins by constructing seed prompts grounded in the Wikidata knowledge graph (Vrandečić & Krötzsch, 2014): we query SPARQL (Harris & Seaborne, 2013) for well-connected hub entities across approximately 25 verified entity classes (cities, universities, films, chemical elements, etc.), then perform random walks of 4-8 hops through the graph, filtering out degenerate paths via stop-node lists, anti-meta-relation exclusions, and minimum path-length thresholds. Each valid walk yields a start entity, a chain of factual relations, and a final answer entity.

Data Designer then processes these seeds in three stages: (1) a draft stage converts the structured knowledge-graph walk into a natural-language multi-hop question, (2) an obfuscation stage rewrites the question to hide intermediate entities and eliminate breadcrumb-style chaining – producing search-riddle queries where the solver must decompose the problem independently – and (3) an agent stage in which MiniMax-M2 (MiniMax AI, 2025) solves the obfuscated question by issuing web searches via the Tavily (`https://tavily.com`) MCP search tool, producing a grounded search trajectory with supporting URLs. Each resulting SFT record is a multi-turn conversation where assistant turns interleave chain-of-thought reasoning with structured tool calls, and tool-response turns return search results as JSON, preserving a full Thought–Action–Observation loop across an average of 12 tool calls per trajectory. A final structured-output stage normalizes the agent's raw response into a validated JSON schema.

Terminal Use. The dataset for enhancing terminal capabilities follows the dual-stream *Terminal-Task-Gen* methodology described in Nemotron-Terminal (Pi et al., 2026), comprising a total of 84,864 samples. This pipeline combines the adaptation of existing high-quality datasets with synthetic task generation grounded in a comprehensive terminal skill taxonomy. The source distribution consists of 68,924 synthetic samples, 8,125 samples from Nemotron-Cascade-Math, and 7,815 samples from Nemotron-Cascade-Code (Wang et al., 2025a). For trajectory construction, we use `DeepSeek-V3.2` (DeepSeek-AI, 2025a) as the primary engine to generate step-by-step solution traces within isolated, Dockerized environments through an agentic execution-feedback loop. All samples are generated using the Terminus 2 agent framework (Merrill et al., 2026) as the underlying scaffolding, providing a unified set of terminal tools and a structured interaction protocol to maintain consistency and quality across long-horizon trajectories.

Multilingual. Our multilingual data combines synthetic translations of English SFT examples with a sentence-level parallel corpus to improve machine translation. We reuse the line-by-line translation pipeline from Nemotron 3 Nano, translating into six languages (German, Spanish, French, Italian, Japanese, and Chinese) using `Qwen2.5-Instruct-14b`. After translation, we apply filtering to remove samples in the wrong language and other common failure modes. We observed a recurring pattern where translation disrupts the alignment between prompt specifications and answer formats — a consistency usually maintained in English data. This mismatch led to instruction-following failures during preliminary testing. To mitigate this, we introduced a lightweight post-editing step with `Qwen3-4B-Thinking-2507` to automatically restore format compliance. We also expand the parallel corpus with additional Chinese ↔ English pairs and exclude very short samples that previously degraded post-training performance.

Structured Query Language (SQL). To improve Nemotron 3 Super on enterprise SQL workloads, we generate a synthetic text-to-SQL dataset with NeMo Data Designer (The NeMo Data Designer Team, 2025). The dataset contains 96.5k records spanning MySQL, PostgreSQL, and SQLite across 60 industry sectors, ∼700 domain topics, and 90 SQL concept buckets (from basic SELECT to recursive CTEs, window functions, and geospatial queries). Each sample pairs a natural-language prompt and a fully synthetic database schema context with a target SQL query. To improve robustness and to mimic the real-world messiness of production databases, the pipeline injects distractor tables and columns into the database context. Specifically, related but irrelevant tables and columns are added to the database context, forcing the model to learn to ignore irrelevant schema elements. Prompt diversity is controlled along three axes – instruction style (imperative, declarative, interrogative, contextual, abbreviated), linguistic register (formal, conversational, technical, academic, direct), and politeness level – yielding naturalistic and varied user requests. The final dataset of 96.5k records is validated and filtered down by Data Designer from a larger dataset using per-dialect syntax validators and five LLM-as-a-critic judges.

Conversational Tool Use. Large-scale specialized tool-use training data has been adopted by many models to boost agentic capabilities (Liu et al., 2024; DeepSeek-AI, 2025a; Team, 2025; GLM-4.5-Team, 2025). For Nemotron 3 Super, we scale conversational tool-use data via a fully synthetic, six-stage generation pipeline:

1. Domain Generation: Sample synthetic domains with a model, iteratively expanding initial generations into specialized subdomains.
2. Policy and Tool Generation: Sample customer-service policies and related tools, iteratively improving them via self-refinement; use few-shot prompting and an LM-as-a-Judge for quality filtering to maintain style and formatting.
3. Scenario Generation: Generate plausible user personas, background information, and inquiries

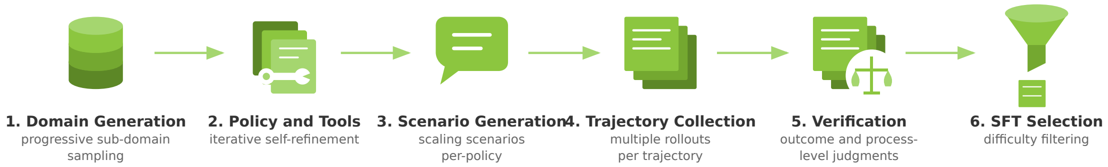

1. Domain Generation
progressive sub-domain sampling
2. Policy and Tools
iterative self-refinement
3. Scenario Generation
scaling scenarios per-policy
4. Trajectory Collection
multiple rollouts per trajectory
5. Verification
outcome and process-level judgments
6. SFT Selection
difficulty filtering

Figure 14 | Overview of the synthetic data generation pipeline for specialized conversational tool-use SFT data used for Nemotron 3 Super.

for each policy setting.

4. Trajectory Collection: For each policy–scenario pair, simulate 16 customer service interactions among a model-based agent, user, and environment.
5. Verification: Evaluate trajectories at both outcome and process levels using an LM-as-a-Judge.
6. SFT Data Selection: Select successful trajectories for SFT and filter for difficulty by dropping scenarios that yield all-success or all-failure outcomes.

A visualization of the pipeline is shown in Figure 14. We utilize `Qwen3-235B-A22B-Thinking-2507`, `Qwen3-32B`, `Qwen3-235B-A22B-Instruct-2507` (Yang et al., 2025), `deepseek-r1-0528` (DeepSeek-AI, 2025b), `DeepSeek-V3.2` (DeepSeek-AI, 2025a), and `gpt-oss-120b` (OpenAI, 2025) on various parts of the above pipeline, yielding 279,116 conversations across 838 domains. This represents a substantial scale-up over Nemotron 3 Nano, which used 15,588 conversations spanning 5 domains.

General-Purpose Tool Use. The broader, general purpose tool-calling synthetic data pipeline begins with the construction of diverse tool sets from ToolEyes (Ye et al., 2025), API-Bank (Li et al., 2023b), UltraTools (Huang et al., 2024), AutoTools (Shi et al., 2025), xLAM (Zhang et al., 2024), Glaive-Function-Calling-v2 (Glaive AI, 2025), Toucan-1.5M (Xu et al., 2025), as well as custom written tools that serve as the foundation for downstream synthetic task generation. A tool-calling trajectory is simulated by grounding in one or multiple of these tool sets. The trajectory simulation involves an LLM playing three roles - User (User-LLM), Assistant (Assistant-LLM), and Tool Environment (Tool-LLM). The User-LLM is seeded with the selected tool set, a persona sampled from Nemotron-Personas-USA (Meyer & Corneil, 2025), and a tool-calling scenario (single-turn, multi-turn, or multi-step). The User-LLM starts by designing a task guided by the tool-calling scenario which is relevant to the selected persona and can be solved by the selected tool set. The Assistant-LLM attempts to solve this task in one or more turns by producing tool-calls and responding to tool execution results. The Tool-LLM is responsible for producing a simulated tool execution result based on the tool-call generated by the Assistant-LLM and the tool being called. The Tool-LLM is prompted with a rubric that helps identify syntactic and semantic errors in tool-calling, as well as the original user query so that the tool results can be contextualized when tool-call is successful. To ensure accuracy, we employ a turn-level and trajectory-level judge similar to the specialized tool-calling data generation. The turn level judge is also paired with a rule-based verification for ensuring correctness of tool-calls. We scale this pipeline with DeepSeek-v3.2 (DeepSeek-AI et al., 2025) and GLM-4.7 (Z.ai / zai-org, 2025) to create a dataset of 1.5M diverse tool-calling trajectories.

The overall general-purpose synthetic tool-calling data pipeline is visualized in figure 15.

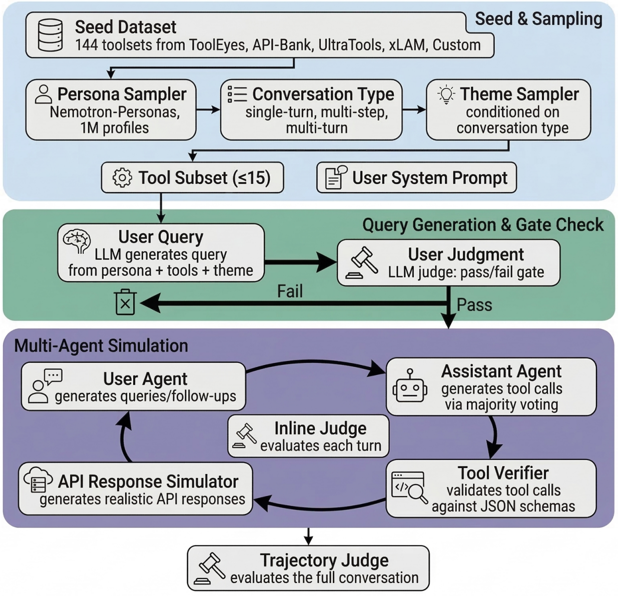

Seed & Sampling
Seed Dataset
144 toolsets from ToolEyes, API-Bank, UltraTools, xLAM, Custom
Persona Sampler
Nemotron-Personas, 1M profiles
Conversation Type
single-turn, multi-step, multi-turn
Theme Sampler
conditioned on conversation type
Tool Subset (≤15)
User System Prompt
Query Generation & Gate Check
User Query
LLM generates query from persona + tools + theme
User Judgment
LLM judge: pass/fail gate
Fail
Pass
Multi-Agent Simulation
User Agent
generates queries/follow-ups
Assistant Agent
generates tool calls via majority voting
Inline Judge
evaluates each turn
API Response Simulator
generates realistic API responses
Tool Verifier
validates tool calls against JSON schemas
Trajectory Judge
evaluates the full conversation

Figure 15 | Overview of the pipeline for general-purpose tool-calling data used in Nemotron 3 Super.

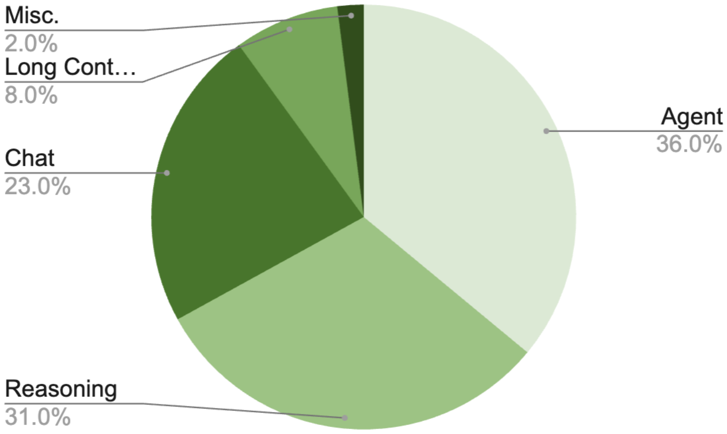


Figure 16 | SFT data blend for Nemotron 3 Super.

3.1.2. SFT Data Blend

Our general stage 1 SFT data blend can be found in Figure 16 (all datasets not listed make up less than 1% of the blend). We train on over 7M total samples. In stage 2, we use 85% of the stage 1 blend and augments it with 256K and 512K-token long-context data. Compared to Nemotron 3 Nano, we significantly increased the volume and diversity of agentic tasks, and allocate it a much larger proportion of our blend.

3.1.3. Reasoning Control

Nemotron 3 Super is trained for three reasoning modes: reasoning-off, regular and low-effort. The low-effort reasoning mode is a new addition. The regular and low-effort reasoning modes have the option to be used in conjunction with inference-time budget control (NVIDIA, 2025b). These combinations of controls provide flexibilities that cover the entire spectrum of accuracy-efficiency trade-off to meet customers' needs in various application scenarios.

The low-effort reasoning mode is introduced during the SFT stage by adding training samples generated by GPT-OSS-120B in its low-effort mode (Du et al., 2025). These low-effort training samples cover the tasks of math reasoning, STEM question answering and instruction following, and represent 2% of the overall SFT data by sample count. The low-effort mode is later optimized during the RL stages to be discussed in the next section.

The SFT recipe for reasoning-off mode and for inference-time budget control is similar to NVIDIA (2025a) with a few differences. We strip the reasoning traces from a random 3% samples for reasoning-off mode. After the main SFT stage, we add a short semi-on-policy SFT stage of 350 steps for inference-time budget control, where we collect roll-outs from the model and truncate 12% of reasoning traces to random reasoning budgets.

3.2. Reinforcement Learning

The RL phase of Nemotron 3 Super post-training consists of three stages followed by an MTP healing stage as shown in Figure 12:

- **Stage 1: Multi-environment RL from Verifiable Rewards** (§3.2.1). This is the primary training stage, where we optimize Nemotron 3 Super jointly across the full set of environments. Training in a unified mixture keeps each RL update informed by the complete environment distribution and helps prevent regressions on individual tasks over the course of training.
- **Stage 2: SWE-RL for end-to-end software engineering tasks** (§3.2.2). We run SWE-RL as a separate stage because SWE rollouts are substantially slower to generate and typically require longer context lengths, creating a throughput bottleneck when co-trained with shorter-horizon environments. Isolating this stage allows us to tune rollout and batching settings for long-horizon, long-context trajectories.
- **Stage 3: RLHF** (§3.2.3). We finally apply RLHF as a distinct stage to improve instruction-following behavior, robustness, and overall interaction quality.
- **Stage 4: MTP Healing**. In this stage we train the MTP heads and keep rest of the weights frozen. We re-use the prompts from RLVR, and train the MTP head using the negative log likelihood loss similar to SFT on the generated responses. We find this stage significantly improves MTP accuracy.

We describe these stages below including the training algorithm, data and systems setup.

3.2.1. Stage 1: Multi-environment RL from Verifiable Rewards

We employ a unified RLVR strategy similar to Nemotron 3 Nano, but significantly scale the number of environments. We find that training on all environments simultaneously yields stable gains, whereas single-environment training leads to severe regressions on other benchmarks.

Our RLVR setting contains 21 environments covering diverse domains, including math, code, STEM, safety, chat, instruction following, long context capabilities, puzzles, and various agentic tasks. For data mixture and curriculum, we adopt an approach similar to Nemotron 3 Nano: we filter out prompts where the SFT model consistently provides correct answers, then sort the remaining samples via a difficulty-based curriculum. Further details of this methodology are available in Nemotron 3 Nano.

Low-effort Reasoning During the multi-environmental RL stages, we convert a subset of the prompts to be in the low-effort mode. For each low-effort prompt, the reward for a roll-out is adjusted as a function of both correctness and the number of generated tokens. The low-effort prompt mix starts with subsets of Math, STEM QA and competitive coding prompts, in total representing 2% of all RL prompts being in low-effort mode, and is later reduced to subsets of Math and STEM QA, representing just 1% of RL prompts. For Math and STEM QA, we randomly sample a subset. For competitive coding, we have a set of coding problems that have been withheld from SFT data and we only sample from this withheld set for low-effort coding prompts. Empirical results suggest that this data strategy provides sufficient generalization and the low-effort mode is improved across a wide set of benchmarks during the course of multi-environmental RL.

RLVR Data

We scale up our RL data significantly compared to Nemotron 3 Nano. We describe the RL datasets we used in our multi environment RL below. The majority of the RL training environments are open sourced in Nemo Gym (NVIDIA, 2025a). In total, we train on 21 environments and 37 different RL datasets.

— **Math.** We use the same competitive math problems as Nemotron 3 Nano. For this dataset, we train both with and without a Python execution tool. We also introduce a new environment for formal proof verification.

— **Code.** We train on competition-style code data from Nemotron 3 Nano and Wang et al. (2025a).

— **STEM.** We include the STEM datasets from Nemotron 3 Nano and add newly curated, more challenging scientific problems.

— **Instruction Following.** We augment the instruction-following data from Nemotron 3 Nano with a new multi-challenge dataset. In this setting, the agent must follow complex user instructions, with rewards computed from a predetermined rubric.

— **Safety.** We add two environments: one targeting reduced over-refusals on safety-related prompts, and one improving robustness to jailbreaks. For jailbreaks, seed prompts come from our SFT data; to surface harder attacks during RL, we apply an iterative attack pipeline following PAIR (Chao et al., 2024), attacking an early SFT-only checkpoint and collecting modified prompts with high attack success rates.

— **Long Context.** We use the same long-context environment introduced in Nemotron 3 Nano.

— **Agentic Tool Use.** Beyond the tool-use environments from Nemotron 3 Nano, we add new environments focused on conversational tool use and terminal use.

— **Reasoning Gym.** We train with Reasoning Gym (Stojanovski et al., 2025), enabling learning over a diverse suite of reasoning tasks.

3.2.2. Stage 2: End-to-end RL for Software Engineering

In the SWE-RL stage, we improve the model's ability to autonomously solve GitHub issues under diverse harnesses. Each rollout launches an Apptainer container with the target repository, runs an OpenHands agent loop to produce a code patch, and evaluates it against ground-truth tests for a binary reward. For tool diversity, we implemented OpenCode and Codex agent classes within OpenHands that match the tool formats of Claude Code and Codex CLI, reusing a single harness while varying tools and prompts at training time. This multi-harness training improves the model's generalization and performance across all target harnesses at inference time.

3.2.3. Stage 3: Reinforcement Learning from Human Feedback

We follow a similar approach to Nemotron 3 Nano for RLHF, training a large GenRM model to provide supervision during RL. Rather than using a vanilla GenRM, we train a principle following GenRM as in Wang et al. (2025b). These principles allow us to guide Nemotron 3 Super's behavior on important domains like identity and safety related topics. Similar to Nemotron 3 Nano, we use `Qwen3-235B-A22B-Thinking-2507` as the initialization for training the GenRM.

To train the GenRM we use the Helpsteer 3 dataset (Wang et al., 2025c), commercially friendly subsets of the lmarena-140k dataset (Chiang et al., 2024), and some more recently collected human preference data. Unlike Nemotron 3 Nano, we train using our GenRM throughout our multi environment RL stage and also perform a separate RLHF-only stage at the end of post training.

3.2.4. Algorithm

We use an asynchronous GRPO setup in which training and inference are decoupled across separate GPU devices. Inference workers continuously generate trajectories, which are stored in a rollout buffer. Once enough trajectories are collected to form a batch, the batch is sent to the training engine for a model update. We push the updated weights to the inference workers as soon as a new model version is available. Because weight updates can happen mid-rollout, a single trajectory may contain tokens produced by different model versions. We do not recompute the KV cache after updating the model weights on the inference workers. To avoid excessive policy lag, which may result in accuracy degradation, we restrict the inference workers to be at most one step behind the latest model version.

To stabilize the training and minimize off-policy effects caused by the training-inference mismatch and policy lag, we mask the importance sampling ratio computed from the training and inference logprobs (Shao et al., 2024; Team et al., 2025; Yao et al., 2025).

In multi-environment RLVR, we sample 256 prompts per step and generate 16 responses per prompt. We train with a batch size of 4096, which corresponds to a single gradient update per rollout. We begin training with a maximum generation length of 49K tokens and later increase it to 64K.

Agentic RL - PivotRL: Post-training for long-horizon agentic capabilities has a tension between efficiency and accuracy. By long-horizon, we mean tasks that require many turns of interaction with an environment, such as conversational tool use, code editing, terminal interaction, and web search. SFT is cheap and simple for this task, but it often degrades performance outside of the target domain (OOD). End-to-end RL avoids that outcome to a large part, but it is costly because every update requires online interactive rollouts in complex environments. To address this, during the post-training of Super, we adopt PivotRL.

PivotRL is a assistant-turn-level RL method that addresses this tradeoff by reusing offline SFT expert trajectories during RL. It focuses training on informative turns (called "pivots") within those SFT traces, where the policy has uncertainty over the next action, and it uses a domain-appropriate reward to match the policy's action to the expert action, so the model gets credit for similar actions rather than the exact expert action. We notice that this method greatly improves the efficiency of our agentic RL, without facing the OOD degradation issues of SFT.

We apply PivotRL for all agentic domains: including for Agentic Programming, Search, Terminal Use, and Conversational Tool Use. We will have a manuscript with more details soon.

3.2.5. Infrastructure

RL at the frontier of model post-training is currently defined by scaling up to an increasing diversity of tasks or environments designed for the model to learn increasingly general capabilities. Scaling RL to many environments requires a high-performance, extensible, and standardized interface for coordinating between rollouts and training. To address the scaling performance and extensibility challenges using one standard framework, we adopt NeMo Gym (NVIDIA, 2025a) and NeMo RL (NVIDIA, 2025b) for enabling large-scale RL on many different environments/verifiers.

NeMo Gym is based on the abstraction of servers. There are three core varieties of servers in Gym: (1) agents, (2) models, and (3) resources. An agent server implements the rollout kernel of a RL environment. A model server wraps an inference engine such as vLLM (Kwon et al., 2023) to provide a prompt-response API, and also carefully preserves token and inference log-prob data and metadata required for RL. A resource server provides a verification API for computing rewards from a given rollout.

Our Nemotron Super 3 RLVR experiments were all based on an integrated infrastructure of NeMo RL and NeMo Gym: NeMo RL acts as the RL training loop controller, using Megatron-Core (Shoeybi et al., 2020) for model training at scale, and routing all rollouts through NeMo Gym and vLLM.

NeMo RL and NeMo Gym use ray for orchestration and resource management and deploy the ray cluster on SLURM. Megatron training workers, vLLM generation workers, Gym environments and judge models are all scheduled onto a single ray cluster.

Async RL Infrastructure: All RL stages used asynchronous RL, where generation can happen independently of training which improves training efficiency by trading off how on-policy the rollouts are. The trainings used one-step off policy where each training step was earmarked for a future training step so there were no wasted rollouts. Training and generation were not collocated which simplifies deployment and avoids having to orchestrate complex memory management between the async training and generation workers.

All asynchronous RL runs also used in-flight weight updates (Piché et al., 2025) where training can update generation worker weights without waiting for the remaining ongoing rollouts to finish. The result is a single rollout can have tokens and log probabilities from differently aged policies. Enabling in-flight weight updates is critical for speeding up asynchronous RL training. We did not recompute the KV cache after in-flight weight updates.

Resiliency: As we scaled up to 1k GPUs, we encountered several issues that caused intermittent failures that were not observed in smaller job shapes. These issues fell into two categories (1) hardware related and (2) software related.

We observed several hardware issues that required full restart of the job, so several optimizations were also made to improve startup time by (1) parallelizing all initialization (2) prefetching all virtual environments and binaries (3) utilize caching in upstream repos like vLLM and flashinfer.

Parallel initialization exacerbated latent race conditions with port bindings in the post-training software stack, which became a significant failure point. Several components of the post-training software required ports: (1) Ray control plane (2) vLLM workers and OpenAI servers (3) TCP rendezvous (4) NeMo Gym servers. Due to the high number of processes needing ports on a node, we hit port conflict frequently at 1K GPU scale that were all time-of-check to time-of-use (TOCTOU) race conditions. The pattern we observed was that a component would check if a port was available without claiming it exclusively, and by the time it (or another process it notified) attempted to bind, another process had already claimed the port.

SWE-RL Infrastructure: Training a model on software engineering tasks requires a gym environment that can execute hundreds of concurrent agent-codebase interactions, each within an isolated sandbox, and return a reward signal derived from real test execution. In the SWE-RL environment in Nemo-Gym, each rollout launches an Apptainer container with the target repository, runs the OpenHands agent loop to produce a code patch, and runs the ground-truth tests to compute a binary reward. Rollouts are distributed across nodes using Ray with a `SPREAD` scheduling strategy. Below we describe the key components of this environment.

- **Container Execution with Apptainer.** The absence of root access on our cluster restricts the use of Docker for container isolation. Instead, we use Apptainer (formerly Singularity) to run each SWE task instance in a pre-built container image (`.sif` files), providing filesystem isolation via a writable tmpfs overlay while sharing the host kernel.
- **OpenHands Agent Loop.** The agent loop is executed by a modified version of OpenHands, managing the full lifecycle of each interaction: initializing the runtime, presenting the problem statement, running the agent's step loop upto a configurable turn limit, extracting the git patch, and cleaning up. The agent interacts with the repository workspace through bash commands and file operations via a tmux-based session.
- **Memory Management.** Since Apptainer containers share host memory and processes (unlike Docker's cgroup isolation), runaway agent processes can cause OOM conditions affecting the entire node. We implemented a memory watchdog daemon that monitors the aggregate RSS of the tmux process tree and proactively kills processes inside panes when a configurable limit is exceeded, while keeping the tmux server alive for graceful recovery.
- **Command Blocklist.** The shared-kernel nature of Apptainer means an agent issuing `killall` or `pkill` could terminate training processes or vLLM servers on the same node. A regex-based command block list intercepts and blocks dangerous commands before execution, returning informative error messages with safer alternatives.
- **Harness Diversity.** To increase tool diversity during training without building separate harness integrations, we implemented OpenCode and Codex agent classes within OpenHands that match the tool input/output formats of their respective external harnesses (Claude Code and Codex CLI). Both agents plug into OpenHands' existing runtime and conversation memory, inheriting container management and observation handling.
- **Serialization.** We replaced Python's standard `json` with `orjson` for serialization of HTTP payloads between the gym and the model server, as each trajectory turn carries prompt token IDs, generated token IDs, and log probabilities, resulting in large payloads that benefit from orjson's Rust-based implementation.

3.3. Post-trained Model Evaluations

We evaluate Nemotron 3 Super on the same broad benchmark suite and evaluation stack as Nemotron 3 Nano (NVIDIA, 2025a), covering general knowledge, reasoning, agentic, instruction following, long-context, and multilingual capability. All evaluation results were collected via Nemo Evaluator SDK[9] and for most benchmarks, the Nemo Skills Harness[10]. For reproducibility purposes, the open source container on Nemo Skills packaged via NVIDIA's Nemo Evaluator SDK used for evaluations can be found here[11]. In addition to Nemo Skills, the evaluations also used dedicated open-source packaged containers for Tau-2 Bench (default prompt), Terminal Bench Hard (48 tasks), ScaleAI Multi Challenge Multi-turn Instruction Following, Ruler. More details on the evaluation settings

[9] https://github.com/NVIDIA-NeMo/Evaluator

[10] https://github.com/NVIDIA-NeMo/Skills

[11] https://catalog.ngc.nvidia.com/orgs/nvidia/teams/eval-factory/containers/nemo_skills

Benchmark	N-3-Super	Qwen3.5-122B-A10B	GPT-OSS-120B
General Knowledge			
MMLU-Pro	83.73	86.70	81.00
Reasoning			
AIME25 (no tools)	90.21	90.36	92.50
HMMT Feb25 (no tools)	93.67	91.40	90.00
HMMT Feb25 (with tools)	94.73	89.55	-
GPQA (no tools)	79.23	86.60	80.10
GPQA (with tools)	82.70	-	80.09
LiveCodeBench (v5 2024-07↔2024-12)	81.19	78.93	88.00
SciCode (subtask)	42.05	42.00	39.00
HLE (no tools)	18.26	25.30	14.90
HLE (with tools)	22.82	-	19.0
Agentic			
Terminal Bench (hard subset)	25.78	26.80	24.00
Terminal Bench Core 2.0	31.00	37.50	18.70
SWE-Bench (OpenHands)	60.47	66.40	41.9
SWE-Bench (OpenCode)	59.20	67.40	-
SWE-Bench (Codex)	53.73	61.20	-
SWE-Bench Multilingual (OpenHands)	45.78	-	30.80
TauBench V2			
Airline	56.25	66.0	49.2
Retail	62.83	62.6	67.80
Telecom	64.36	95.00	66.00
Average	61.15	74.53	61.0
BrowseComp with Search	31.28	-	33.89
BIRD Bench	41.80	-	38.25
Chat & Instruction Following			
IFBench (prompt)	72.56	73.77	68.32
Scale AI Multi-Challenge	55.23	61.50	58.29
Arena-Hard-V2	73.88	75.15	90.26
Long Context			
AA-LCR	58.31	66.90	51.00
RULER 256k	96.30	96.74	52.30
RULER 512k	95.67	95.95	46.70
RULER 1M	91.75	91.33	22.30
Multilingual			
MMLU-ProX (avg over langs)	79.36	85.06	76.59
WMT24++ (en→xx)	86.67	87.84	88.89

Table 5 | Evaluation suite for Nemotron 3 Super. We compare against Qwen-3.5-122B-A10B and GPT-OSS-120B. (Values omitted in this template; to be filled with final scores.)

can be found in the Nemo Evaluator SDK configs folder[12]. The following benchmarks are not onboarded yet in our open source tools and for these we used either their official open source implementation or otherwise an internal scaffolding that we plan to open source in the future: SWE Bench Verified (OpenHands), SWE Bench Multilingual (OpenHands), BrowseComp with Search (internal implementation, with Serp API), Terminal Bench Core 2.0 (Harbor).

Reasoning Capabilities. We report results on AIME 25, HMMT Feb 25, GPQA (Rein et al., 2023), LiveCodeBench v5 (Jain et al., 2024), SciCode (Tian et al., 2024), and HLE (Phan et al., 2025). Across all benchmarks Nemotron 3 Super is competitive with `GPT-OSS-120B`, while lagging behind `Qwen-3.5-122B` slightly.

Agentic capabilities. We report results on TerminalBench (the hard subset and the v2 set), SWE-Bench (OpenHands, OpenCode, Codex, and the Multilingual set) (Jimenez et al., 2023), TauBench V2 (Airline, Retail, Telecom; and their average) (Barres et al., 2025), and BrowseComp (Wei et al., 2025). For Browsecomp, our harness takes strong inspiration from the browser tool released with GPT OSS. (OpenAI, 2025), and we do not evaluate with any context management strategies. For SQL we evaluate on the BIRD benchmark Li et al. (2023a) dev set (1,534 samples, SQLite, execution accuracy). Across all agentic benchmarks, Nemotron 3 Super outperforms or is competitve with GPT-OSS 120B and is competitive to Qwen 3.5 122B on some harnesses.

Chat and Instruction Following Capabilities. We report results on IFBench, Multi-Challenge, Arena-Hard V2 (Li et al., 2024). Across all benchmarks Nemotron 3 Super is competitive with the baseline models.

Long Context Capabilities. We report results on Ruler (Hsieh et al., 2024) using 100 samples per task and AALCR.

Multilingual Capabilities. We measure multilingual capability on MMLU-ProX (Xuan et al., 2025) and WMT24++ en→xx (Deutsch et al., 2025). Nemotron 3 Super matches or outperforms the baseline models on both bencharks.

For comparison with GPT-OSS-120B and Qwen-3.5-122B-A10B, we use officially reported numbers whenever available; when not available, we follow the Nemotron 3 Nano report (NVIDIA, 2025a) procedure of either sourcing values from reputable public aggregators (when consistent with the official protocol) or computing scores ourselves using the official evaluation settings.

4. Quantization For Inference

We apply post-training quantization (PTQ) using Model-Optimizer[13] to quantize weights and activations to generate two efficient deployment checkpoints: FP8 (W8A8) for Hopper and NVFP4 (W4A4) for Blackwell.

[12] `https://github.com/NVIDIA-NeMo/Evaluator/tree/main/packages/nemo-evaluator-launcher/examples/nemotron/nemotron-3-super`

[13] `https://github.com/NVIDIA/Model-Optimizer/`

4.1. Nemotron 3 Super FP8 Checkpoint

For FP8 PTQ calibration, we used a small subset containing 256 samples with 65536 context length from the post-training SFT dataset. For FP8 quantization, we quantized MoE GEMMs, both routed and shared, and Mamba Linear layers. We also kept the KV Cache in FP8, whereas Mamba state cache has been quantized to FP16 for speedup. The precision assignments for the different operators in this checkpoint are summarized in Table 6.

Table 6 | Precision settings for the FP8 checkpoint compared with the BF16 baseline.

Configuration	FP8 Checkpoint	BF16 Baseline
Embedding	BF16	BF16
Attention GEMM (QKV and Out Projection)	BF16	BF16
KV Cache + Attention BMM1	FP8	FP8
Attention BMM2	BF16	BF16
MoE GEMM (Sparse Experts and Shared Experts)	FP8	BF16
MoE Latent Projection GEMM	BF16	BF16
Router	FP32	FP32
Mamba GEMM	FP8	BF16
Mamba SSM Cache	FP16	FP32
Mamba 1D Conv	BF16	BF16
Output Layers	BF16	BF16

4.2. Nemotron 3 Super FP4 Checkpoint

FP4 is a more aggressive quantization format than FP8 and is particularly attractive for prefill-heavy inference workloads, such as coding-agent deployments, where linear and MoE GEMMs are major performance bottlenecks. NVFP4 is natively accelerated on Blackwell GPUs and delivers better accuracy than alternative FP4 formats such as MXFP4 (NVIDIA, 2025). NVFP4 for inference (NVIDIA, 2025) uses signed E2M1 values with per-block scaling over 1D blocks of size 16 along the last dimension. These per-block scales are further quantized into FP8 E4M3 using a per-tensor statically calibrated FP32 scaling factor.

In the baseline NVFP4 PTQ recipe (NVIDIA, 2025), each per-block scale is determined by the maximum absolute value in the block. We evaluated a range of alternative PTQ methods. The results of these experiments can be found in Appendix B.1. The best overall results were obtained with a hybrid FP4 recipe: weight per-block scales were selected by minimizing weight MSE, while activation per-block scales continued to use max-based scaling. This choice is both effective and practical. Weight quantization is calibrated offline so an expensive scale search can be performed without impacting runtime performance. Activation quantization must be computed efficiently at runtime, making scale search algorithms impractical. Max-based scaling of activations provide a good trade-off between runtime performance and quantization accuracy.

In addition, we selectively promoted some layers from FP4 (W4A4) to FP8 (W8A8) or BF16 (W16A16) to further improve accuracy. We used Model-Optimizer AutoQuantize [14], a neural architecture search (NAS) inspired method for deriving optimal mixed-precision assignments. AutoQuantize estimates per-operation sensitivity, models the performance cost of available quantization choices, and solves

[14] https://github.com/NVIDIA/Model-Optimizer/tree/main/examples/llm_ptq

for the optimal layer-wise allocation using a knapsack-style optimization procedure. Its sensitivity metric follows a second-order Taylor approximation inspired by Optimal Brain Surgeon (Hassibi et al., 1993), as introduced in LLM-MQ (Li et al., 2023c). AutoQuantize generalizes LLM-MQ beyond weight-only quantization. It supports operator-level quantization, including joint weight-and-activation quantization for GEMMs, and accounts for inference deployment constraints such as operator fusion. The detailed AutoQuantize algorithm is given in B.2.

Overall, our final NVFP4 PTQ recipe combines:

- calibrated per-block weight scaling that minimizes MSE,
- dynamic per-block max-based activation scaling, and
- selective promotion of sensitive layers through Model-Optimizer AutoQuantize.

This combination addresses the accuracy loss of naive NVFP4 PTQ while preserving the runtime efficiency needed for deployment. The resulting per-operator precision assignments in the final NVFP4 checkpoint are summarized in Table 7.

Table 7 | Precision settings for the backbone NVFP4 checkpoint compared with the BF16 baseline.

Configuration	**AutoQuantize Searched?**	**NVFP4 Checkpoint**	**BF16 Baseline**
Embedding	No	BF16	BF16
Attention QKV Projection GEMM	Yes	BF16	BF16
Attention Output Projection GEMM	Yes	FP8 / BF16	BF16
KV Cache + Attention BMM1	No	FP8	FP8
Attention BMM2	No	BF16	BF16
Sparse Expert (Routed) GEMM	Yes	NVFP4	BF16
Shared Expert GEMM	Yes	NVFP4 / FP8 / BF16	BF16
MoE Latent Projection GEMM	Yes	FP8 / BF16	BF16
Router	No	FP32	FP32
Mamba Projection GEMM	Yes	FP8 / BF16	BF16
Mamba 1D Conv	No	BF16	BF16
Mamba SSM Cache	No	FP16	FP32
Output Layers	No	BF16	BF16

For all searched backbone GEMMs, AutoQuantize considered candidate precisions from { NVFP4, FP8, BF16 } and selected the per-operator assignment under a quantization-sensitivity objective with an effective-precision budget of 4.75 bits. In the searched model, sparse-expert GEMMs are assigned NVFP4 throughout, attention and Mamba projection GEMMs are assigned FP8 or BF16, and shared-expert GEMMs use a mix of NVFP4, FP8, and BF16.

Combining AutoQuantize with the improved NVFP4 PTQ recipe produced a mostly-FP4 model, with only a small subset of layers retained in FP8 or BF16 for accuracy preservation. The full mixed-precision PTQ process completed in less than 2 hours on a single B200 node with 8 GPUs, using 512 samples from the Nemotron 3 Super SFT dataset at sequence length 4096. The resulting model achieved 99.8% median accuracy relative to the BF16 baseline while retaining near-FP4 performance. Final evaluation results are reported in Table 9.

4.3. Mamba State Quantization

In memory-bound settings, DRAM reads of the Mamba state cache (SSM cache) become a major bottleneck to decoding speed. The SSM cache is stored in FP32 by default. One option is to store the SSM cache in FP16 while arithmetic executes in FP32. In this case, the cache is fetched from memory in FP16, upcast to FP32 for the recurrent update, and then cast back to FP16 for storage. Table 8 shows experiments on an early checkpoint of Nemotron 3 Super. These results show that directly casting the SSM cache to FP16 can result in up to 40% increase in verbosity when combined with W8A8 quantization. Even with maintaining weights and activations in BF16, casting the SSM cache to FP16 leads to up to 37% increase in verbosity.

Table 8 | Impact of SSM cache recipe on code benchmarks and verbosity.

		Accuracy		Completion Tokens		Verbosity Increase	
Weight and Activation Precision	**SSM cache recipe**	**livecodebench (pass@1 avg-of-8)**	**scicode (pass@1 avg-of-8)**	**livecodebench (pass@1 avg-of-8)**	**scicode (pass@1 avg-of-8)**	**livecodebench (pass@1 avg-of-8)**	**scicode (pass@1 avg-of-8)**
W16A16	FP32	72.91	40.90	21769	3680	0.00%	0.00%
W16A16	FP16	73.24	42.01	29812	3760	36.95%	2.19%
W16A16	FP16+RS	72.00	41.94	21392	3580	−1.73%	−2.72%
W8A8	FP16	73.13	40.98	30536	3780	40.27%	2.70%
W8A8	INT16+block128	72.22	41.46	22406	3521	2.90%	−4.30%
W8A8	FP16+SR (Philox 10)	72.22	40.38	22120	3672	1.71%	−0.74%
W8A8	FP16+SR (Philox 5)	72.63	41.86	22159	3720	1.79%	1.08%
W8A8	FP16+SR (Philox 4)	72.85	40.46	21631	3785	−0.63%	2.84%
W8A8	FP16+SR (Philox 3)	70.07	39.94	24098	3827	10.70%	3.98%

A key challenge in quantizing the Mamba cache is that quantization error does not remain local to a single step. Because Mamba decoding is recurrent, quantization error from previous steps propagates into future steps and accumulates over time. This accumulation of quantization error can be seen by unrolling the recurrent update. Let the recurrent state update be $h_t = A_t h_{t-1} + B_t x_t$, and let cache quantization at step t introduce an additive error e_t, so that $h_{q,t} = A_t h_{q,t-1} + B_t x_t + e_t$. Unrolling this recursion gives

$$\begin{aligned} h_{q,t} &= h_t + e_t + A_t e_{t-1} + A_t A_{t-1} e_{t-2} + \cdots \\ &\quad + A_t A_{t-1} \cdots A_2 e_1 + A_t A_{t-1} \cdots A_1 e_0 \\ &= h_t + \sum_{i=0}^{t} \left(\prod_{j=i+1}^{t} A_j \right) e_i, \end{aligned} \tag{3}$$

showing that quantization error from earlier steps is propagated through subsequent recurrent transitions and can accumulate over decoding time.

Addressing these quantization errors through changes in training (e.g., QAT or QAD) is non-trivial. Mamba training uses the chunked State Space Duality algorithm which does not explicitly materialize the recurrence relationship and the inference-time cache. Accurately modeling the recurrent decode-time cache behavior during training introduces substantial overhead. We therefore focused on training-free methods to recover the accuracy lost from cache quantization.

One way to decrease the accumulation of quantization error during PTQ is to increase the mantissa precision. We explored this by using INT16 instead of FP16 for the SSM cache. Naive INT16 quantization did not improve verbosity as tensor-level analysis showed that the SSM cache has a wide dynamic range. We then introduced FP32 per-block scaling over blocks of size 128 along the state dimension to increase effective dynamic range. This eliminated the verbosity issue (Table 8).

We also explored a hypothesis that the error accumulation was tied to rounding during the cast from FP32 to FP16. The key issue is that round to nearest, ties on even (RTNE) introduces bias in the quantization process. Because RTNE maps a given input to the same rounded value, its quantization error has zero variance but non-zero bias relative to the original value. In contrast, stochastic rounding (SR) is unbiased in expectation. In a recurrent setting, the bias from RTNE accumulates coherently over time, whereas stochastic rounding replaces this systematic drift with zero-mean noise. Based on this observation, we applied stochastic rounding before casting the cache to FP16 which. This fixed the verbosity issue for both the BF16 baseline and the FP8 checkpoint (Table 8).

Benchmark	**N-3-Super**	**N-3-Super FP8**	**N-3-Super NVFP4**
General Knowledge			
MMLU-Pro	83.73	83.63	83.33
Reasoning			
HMMT Feb25 (with tools)	94.73	94.38	95.36
GPQA (no tools)	79.23	79.36	79.42
LiveCodeBench (v6 2024-08↔2025-05)	78.69	78.44	78.44
LiveCodeBench (v5 2024-07↔2024-12)	81.19	80.99	80.56
SciCode (subtask)	42.05	41.38	40.83
HLE (no tools)	18.26	17.42	17.42
Agentic			
Terminal Bench (hard subset)	25.78	26.04	24.48
TauBench V2			
Airline	56.25	56.25	54.75
Retail	62.83	63.05	63.38
Telecom	64.36	63.93	63.27
Average	61.15	61.07	60.46
Chat & Instruction Following			
IFBench (prompt)	72.58	72.32	73.30
Scale AI Multi-Challenge	55.23	54.35	52.8
Arena-Hard-V2 (Hard Prompt)	73.88	76.06	76.00
Long Context			
AA-LCR	58.31	57.69	58.06
RULER 128k (500 samples per task)	96.79	96.85	95.99
RULER 256k (500 samples per task)	96.60	96.33	96.52
RULER 512k (500 samples per task)	96.09	95.66	96.23
Multilingual			
MMLU-ProX (avg over languages)	79.35	79.21	79.37

Table 9 | Evaluation suite for Nemotron 3 Super. We compare our FP8 and NVFP4 optimized models with BF16 model.

Table 8 shows accuracy and verbosity of the different SSM cache recipes for livecodebench and scicode. Both INT16 with per-block scales and FP16 with stochastic rounding were able to maintain accuracy and verbosity similar to the FP32 baseline. We selected FP16 with stochastic rounding (SR) using Philox<5> pseudorandom number generation as the SSM cache recipe for Nemotron 3 Super for three reasons:

- It does not require calculating, storing, and loading block scale factors.
- Blackwell provides a dedicated PTX instruction for stochastic rounding during type conversion.
- Blackwell supports Philox-based pseudorandom number generation through cuRAND.

To further improve efficiency, Table 8 also varies the number of Philox rounds. Increasing the number of rounds improves the statistical quality of the generated values, while reducing the number of rounds lowers pseudorandom number generation overhead. Philox<5> was chosen to maintain accuracy and verbosity while minimizing pseudorandom number generation overhead.

5. Conclusion

We introduce Nemotron 3 Super, a 12B active and 120B total parameter MoE hybrid Mamba-Attention model with strong agentic capabilities. Nemotron 3 Super employs LatentMoE to improve accuracy and incorporates MTP layers to accelerate inference via speculative decoding. We pretrained Nemotron 3 Super on 25 trillion text tokens with low-precision NVFP4, followed by post-training on a diverse set of RL environments. Finally, we quantized the model to FP8 and NVFP4, achieving significantly higher inference throughput without sacrificing model accuracy. Nemotron 3 Super achieves up to 2.2× higher throughput than GPT-OSS-120B while maintaining higher accuracy across a wide range of tasks. We release the pre-trained, post-trained, and quantized checkpoints for Nemotron 3 Super on HuggingFace.

Contributors

We thank the following people for their invaluable contributions to NVIDIA Nemotron 3 Super.

Aaron Blakeman, Aakshita Chandiramani, Abdullahi Olaoye, Abhilash Somasamudramath, Abhibha Gupta, Abhinav Khattar, Adeola Adesoba, Adi Renduchintala, Adil Asif, Aditya Agrawal Lilach Ilan, Aditya Vavre, Ahmad Kiswani, Aishwarya Padmakumar, Ajay Hotchandani, Akanksha Shukla, Akhiad Bercovich, Aleksander Ficek, Aleksandr Shaposhnikov, Alex Gronskiy, Alex Kondratenko, Alex Neefus, Alex Steiner, Alex Yang, Alexander Bukharin, Alexander Young, Ali Hatamizadeh, Ali Taghibakhshi, Alina Galiautdinova, Alisa Liu, Alok Kumar, Ameya Sunil Mahabaleshwarkar, Amir Klein, Amit Zuker, Amnon Geifman, Anahita Bhiwandiwalla, Ananth Subramaniam, Andrew Tao, Anjaney Shrivastava, Anjulie Agrusa, Ankur Srivastava, Ankur Verma, Ann Guan, Anna Shors, Annamalai Chockalingam, Anubhav Mandarwal, Aparnaa Ramani, Arham Mehta, Arti Jain, Arun Venkatesan, Asha Anoosheh, Ashwath Aithal, Ashwin Poojary, Asif Ahamed, Asit Mishra, Asli Sabanci Demiroz, Asma Kuriparambil Thekkumpate, Atefeh Sorabizadeh, Avinash Kaur, Ayush Dattagupta, Barath Subramaniam Anandan, Bardiya Sadeghi, Barnaby Simkin, Ben Lanir, Benedikt Schifferer, Benjamin Chislett, Besmira Nushi, Bilal Kartal, Bill Thiede, Bita Darvish Rouhani, Bobby Chen, Boris Ginsburg, Brandon Norick, Branislav Kisacanin, Brian Yu, Bryan Catanzaro, Buvaneswari Mani, Carlo del Mundo, Chanran Kim, Chantal Hwang, Chankyu Lee, Chao Ni, Charles Wang, Charlie Truong, Chenhan Yu, Chenjie Luo, Cherie Wang, Cheng-Ping Hsieh, Chetan Mungekar, Chintan Patel, Chris Alexiuk, Chris Holguin, Chris Wing, Christian Munley, Christopher Parisien, Chunyang Sheng, Chuck Desai, Collin Neale, Cyril Meurillon, Dakshi Kumar, Dan Gil, Dan Su, Dane Corneil, Daniel Afrimi, Daniel Burkhardt Eliuth Triana, Daniel Egert, Daniel Fatade Douglas O'Flaherty, Daniel Lo, Daniel Rohrer, Daniel Serebrenik, Daniil Sorokin, Daria Gitman, Daria Levy, Darko Stosic, David Edelsohn, David Messina, David Mosallanezhad, David Tamok, Deena Donia, Deepak Narayanan, Devin O'Kelly, Dheeraj Peri, Dhruv Nathawani, Di Wu, Dima Rekesh, Dina Yared, Divyanshu Kakwani, Dmitry Konyagin Brandon Tuttle, Dong Ahn, Dongfu Jiang, Dorrin Poorkay, Duncan Riach, Dusan Stosic, Dustin Van Stee, Edgar Minasyan,

Edward Lin, Eileen Peters Long, Elad Segal, Elena Lewis, Elena Lantz, Ellie Evans, Elliott Ning, Eric Chung, Eric Harper, Eric Pham-Hung, Eric W. Tramel, Erick Galinkin, Erik Pounds, Esti Etrog, Evan Briones, Evan Wu, Evelina Bakhturina, Evgeny Tsykunov, Ewa Dobrowolska, Farshad Saberi Movahed, Farzan Memarian, Fay Wang, Fei Jia, Felipe Soares, Felipe Vieira Frujeri, Feng Chen, Fengguang Lin, Ferenc Galko, Fortuna Zhang, Frankie Siino, Frida Hou, Gantavya Bhatt, Gargi Prasad, Geethapriya Venkataramani, Geetika Gupta, George Armstrong, Gerald Shen, Giulio Borghesi, Gordana Neskovic, Gorkem Batmaz, Grace Lam, Grace Wu, Greg Pauloski, Greyson Davis, Grigor Nalbandyan, Guoming Zhang, Guy Farber, Guyue Huang, Haifeng Qian, Haran Kumar Shiv Kumar, Harry Kim, Harsh Sharma, Hayate Iso, Hayley Ross, Herbert Hum, Herman Sahota, Hexin Wang, Himanshu Soni, Hiren Upadhyay, Huy Nguyen, Iain Cunningham, Ido Galil, Ido Shahaf, Igor Gitman, Igor Shovkun, Igino Padovani, Ikroop Dhillon, Ilya Loshchilov, Ingrid Kelly, Itamar Schen, Itay Levy, Ivan Moshkov, Izik Golan, Izzy Putterman, Jain Tu, Jan Baczek, Jan Kautz, Janica Rosenberg, Jane Polak Scowcroft, Jared Casper, Jarrod Pflum, Jason Grant, Jason Sewall, Jatin Mitra, Jeffrey Glick, Jian Zhang, Jenny Chen, Jesse Oliver, Jiafan Zhu, Jialin Song, Jiaqi Zeng, Jie Lou, Jill Milton, Jimmy Zhang, Jim Chow, Jinhang Choi, Jining Huang, Jocelyn Huang, Joel Caruso, Joey Conway, Joey Guman, Johan Jatko, John Kamalu, Johnny Greco, Jonathan Cohen, Jonathan Raiman, Joseph Jennings, Joyjit Daw, Juan Yu, Julio Tapia, Junkeun Yi, Jupinder Parmar, Jyothi Achar, Kari Briski, Kartik Mattoo, Katherine Cheung, Katherine Luna, Keith Wyss, Kevin Shih, Kezhi Kong, Khanh Nguyen, Khushi Bhardwaj, Kirill Buryak, Kirthi Shankar Sivamani, Konstantinos Krommydas, Kris Murphy, Krishna C. Puvvada, Krzysztof Pawelec, Kumar Anik, Laikh Tewari, Laya Sleiman, Leo Du, Leon Derczynski, Li Ding, Lingjie Wu, Lizzie Wei, Luis Vega, Maarten Van Segbroeck, Maer Rodrigues de Melo, Magaret Zhang, Mahan Fathi, Makesh Narsimhan Sreedhar, Makesh Sreedhar, Makesh Tarun Chandran, Manuel Reyes Gomez, Maor Ashkenazi, Marc Cuevas, Marc Romeijn, Margaret Zhang, Mark Cai, Mark Gabel, Markus Kliegl, Martyna Patelka, Maryam Moosaei, Matvei Novikov, Matthew Varacalli, Mauricio Ferrato, Mehrzad Samadi, Melissa Corpuz, Meng Xin, Mengdi Wang, Mengru Wang, Meredith Price, Michael Andersch, Michael Boone, Michael Evans, Michael Z Wang, Micah Schaffer, Miguel Martinez, Mikail Khona, Mike Chrzanowski, Mike Hollinger, Mingyuan Ma, Minseok Lee, Mohammad Dabbah, Mohammad Shoeybi, Mostofa Patwary, Nancy Agarwal, Nader Khalil, Nanthini Balasubramaniam, Narsi Kodukula, Nabin Mulepati, Najeeb Nabwani, Narimane Hennouni, Natalie Hereth, Nathaniel Pinckney, Nave Assaf, Negar Habibi, Nestor Qin, Netanel Haber, Neta Zmora, Nick Reamaroon, Nickson Quack, Nidhi Bhatia, Nikhil Jukar, Nikolai Ludwig, Nikki Pope, Nima Tajbakhsh, Nir Ailon, Nirmal Juluru, Nirmalya De, Nowel Pitt, Olivier Delalleau, Oleg Rybakov, Oleksii Hrinchuk, Oleksii Kuchaiev, Oluwatobi Olabiyi, Omer Ullman Argov, Omri Almog, Omri Puny, Oren Tropp, Otavio Padovani, Ouye Xie, Parth Chadha, Pasha Shamis, Paul Gibbons, Pavlo Molchanov, Peter Belcak, Peter Jin, Pinky Xu, Piotr Januszewski, Pooya Jannaty, Prachi Shevate, Pradeep Thalasta, Pranav Prashant Thombre, Prasoon Varshney, Prerana Gambhir, Pritam Gundecha, Przemek Tredak, QA Lun Su, Qing Miao, Qiyu Wan, Quan Tran Minh, Rabeeh Karimi Mahabadi, Rachel Oberman, Rachit Garg, Rahul Kandu, Raina Zhong, Ran El-Yaniv, Ran Zilberstein, Rasoul Shafipour, Renjie Pi, Renee Yao, Richard Mazzarese, Richard Wang, Rick Izzo, Ridhima Singla, Rima Shahbazyan, Rishabh Garg, Ritika Borkar, Ritu Gala, Riyad Islam, Robert Clark, Robert Hesse, Roger Waleffe, Rohit Varma Kalidindi, Rohit Watve, Roi Koren, Ron Fan, Ruchika Kharwar, Ruisi Cai, Ruoxi Zhang, Russell J. Hewett, Ryan Prenger, Ryan Timbrook, Ryota Egashira, Sadegh Mahdavi, Sagar Singh Ashutosh Joshi, Sahil Modi, Samuel Kriman, Sandeep Pombra, Sanjeev Satheesh, Sanjay Kariyappa, Santiago Pombo, Saori Kaji, Satish Pasumarthi, Saurav Mishra, Saurav Muralidharan, Scott Hara, Sean Narenthiran, Sebastian Rogawski, Seonjin Na, Seonmyeong Bak, Sepehr Sameni, Seth Poulos, Shahar Mor, Shaona Ghosh Adam Lord, Sharath Turuvekere Sreenivas, Shaun Kotek, Shaya Gharghabi, Shelby Thomas, Sheng-Chieh Lin, Shibani Likhite, Shiqing Fan, Shiyang Chen, Shreya Gopal, Shrimai Prabhumoye, Shubham Pachori, Shubham Toshniwal, Shuo Zhang, Shuoyang

Ding, Shyam Renjith, Shyamala Prayaga, Siddhartha Jain, Simeng Sun, Sirisha Rella, Sirshak Das, Smita Ithape, Sneha Harishchandra S, Somshubra Majumdar, Soumye Singhal, Sri Harsha Singudasu, Sriharsha Niverty, Stas Sergienko, Stefana Gloginic, Stefania Alborghetti, Stephen Ge, Stephen McCullough, Sugam Dipak Devare, Suguna Varshini Velury, Sukrit Rao, Sumeet Kumar Barua, Sunny Gai, Suseella Panguluri, Sushil Koundinyan, Swathi Patnam, Sweta Priyadarshi, Swetha Bhendigeri, Syeda Nahida Akter, Sylendran Arunagiri, Tailling Yuan, Talor Abramovich, Tan Bui, Tan Yu, Terry Kong, Thanh Do, Thomas Gburek, Thorgane Marques, Tiffany Moore, Tim Moon, Timothy Ma, Tiyasa Mitra, Tomasz Grzegorzek, Tomer Asida, Tomer Bar Natan, Tomer Keren, Tomer Ronen, Traian Rebedea, Trenton Starkey, Tugrul Konuk, Twinkle Vashishth, Tyler Condensa, Udi Karpas, Ushnish De, Vahid Noorozi, Vahid Noroozi, Vanshil Atul Shah, Veena Vaidyanathan, Venkat Srinivasan, Venmugil Elango, Victor Cui, Vijay Korthikanti, Vikas Mehta, Virginia Adams, Virginia Wu, Vitaly Kurin, Vitaly Lavrukhin, Vladimir Anisimov, Wan Seo, Wanli Jiang, Wasi Uddin Ahmad, Wei Du, Wei Ping, Wei-Ming Chen, Wendy Quan, Wenliang Dai, Wenwen Gao, Will Jennings, William Zhang, Xiaowei Ren, Xiaowen Xin, Xin Li, Yang Yu, Yangyi Chen, Yaniv Galron, Yashaswi Karnati, Yejin Choi, Yev Meyer, Yi-Fu Wu, Yian Zhang, Ying Lin, Yonatan Geifman, Yonggan Fu, Yoshi Suhara, Youngeun Kwon, Yuan Zhang, Yuki Huang, Zach Moshe, Zhiyu Cheng, Zhilin Wang, Zihan Liu, Zijia Chen, Zijie Yan, Zhongbo Zhu, Zhuolin Yang, Zuhair Ahmed.

References

Talor Abramovich, Maor Ashkenazi, Carl (Izzy) Putterman, Benjamin Chislett, Tiyasa Mitra, Bita Darvish Rouhani, Ran Zilberstein, and Yonatan Geifman. SPEED-Bench: A Unified and Diverse Benchmark for Speculative Decoding, 2026. URL https://research.nvidia.com/publication/2026-02_speed-bench-unified-and-diverse-benchmark-speculative-decoding.

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, 2024. URL https://arxiv.org/abs/2410.12881.

Syeda Nahida Akter, Shrimai Prabhumoye, Eric Nyberg, Mostofa Patwary, Mohammad Shoeybi, Yejin Choi, and Bryan Catanzaro. Front-loading reasoning: The synergy between pretraining and post-training data. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=VmEkhV2yCX.

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.

Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. Swe-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents, 2025. URL https://arxiv.org/abs/2505.20411.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating Conversational Agents in a Dual-Control Environment. *arXiv preprint arXiv:2506.07982*, 2025.

Noga BenYoash, Menachem Brief, Oded Ovadia, Gil Shenderovitz, Moshik Mishaeli, Rachel Lemberg, and Eitam Sheetrit. Secque: A benchmark for evaluating real-world financial analysis capabilities. In *Proceedings of the Fourth Workshop on Generation, Evaluation and Metrics (GEM^2)*, pp. 212–230, 2025.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about Physical Commonsense in Natural Language, 2019. URL https://arxiv.org/abs/1911.11641.

Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J. Pappas, and Eric Wong. Jailbreaking black box large language models in twenty queries, 2024. URL https://arxiv.org/abs/2310.08419.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, et al. Evaluating Large Language Models Trained on Code, 2021. URL https://arxiv.org/abs/2107.03374.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference, 2024.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *ArXiv*, abs/1803.05457, 2018a.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*, 2018b.

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, 2021. URL https://arxiv.org/abs/2110.14168.

Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu Wu, et al. DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models. *arXiv preprint arXiv:2401.06066*, 2024.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality, 2024. URL https://arxiv.org/abs/2405.21060.

DeepSeek-AI. Deepseek-v3.2-exp: Boosting long-context efficiency with deepseek sparse attention, 2025a.

DeepSeek-AI. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning, 2025b. URL https://arxiv.org/abs/2501.12948.

DeepSeek-AI. DeepSeek-V3 Technical Report, 2025c. URL https://arxiv.org/abs/2412.19437.

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, et al. Deepseek-v3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025. URL https://arxiv.org/abs/2512.02556. State-of-the-art large language model with sparse attention and agentic reasoning capabilities.

Daniel Deutsch, Eleftheria Briakou, Isaac Rayburn Caswell, Mara Finkelstein, Rebecca Galor, Juraj Juraska, Geza Kovacs, Alison Lui, Ricardo Rei, Jason Riesa, et al. WMT24++: Expanding the Language Coverage of WMT24 to 55 Languages & Dialects. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 12257–12284, 2025.

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

Venmugil Elango, Nidhi Bhatia, Roger Waleffe, Rasoul Shafipour, Tomer Asida, Abhinav Khattar, Nave Assaf, Maximilian Golub, Joey Guman, Tiyasa Mitra, Ritchie Zhao, Ritika Borkar, Ran Zilberstein, Mostofa Patwary, Mohammad Shoeybi, and Bita Rouhani. LatentMoE: Toward Optimal Accuracy per FLOP and Parameter in Mixture of Experts, 2026. URL https://arxiv.org/abs/2601.18089.

Steven Feng, Shrimai Prabhumoye, Kezhi Kong, Dan Su, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Maximize Your Data's Potential: Enhancing LLM Accuracy with Two-Phase Pretraining, 2024. URL https://arxiv.org/abs/2412.15285.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. In *International Conference on Learning Representations (ICLR)*, 2023. Originally released as arXiv:2210.17323.

Shaona Ghosh, Prasoon Varshney, Makesh Narsimhan Sreedhar, Aishwarya Padmakumar, Traian Rebedea, Jibin Rajan Varghese, and Christopher Parisien. AEGIS2.0: A Diverse AI Safety Dataset and Risks Taxonomy for Alignment of LLM Guardrails. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 5992–6026, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.306. URL https://aclanthology.org/2025.naacl-long.306/.

Glaive AI. glaiveai/glaive-function-calling-v2. Hugging Face Dataset, 2025. URL https://huggingface.co/datasets/glaiveai/glaive-function-calling-v2. Function calling dataset with multi-turn and diverse domain scenarios for training and evaluating function-calling capabilities.

GLM-4.5-Team. GLM-4.5: Agentic, Reasoning, and Coding (ARC) Foundation Models, 2025. URL https://arxiv.org/abs/2508.06471.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Roziere, David Lopez-Paz, and Gabriel Synnaeve. Better & Faster Large Language Models via Multi-token Prediction. In *International Conference on Machine Learning*, pp. 15706–15734. PMLR, 2024.

Gretel. Gretel Synthetic Safety Alignment Dataset, 12 2024. URL https://huggingface.co/datasets/gretelai/gretel-safety-alignment-en-v1.

Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I. Wang. Cruxeval: A benchmark for code reasoning, understanding and execution, 2024. URL https://arxiv.org/abs/2401.03065.

Melody Y. Guan, Manas Joglekar, Eric Wallace, Saachi Jain, Boaz Barak, Alec Helyar, Rachel Dias, Andrea Vallone, Hongyu Ren, Jason Wei, Hyung Won Chung, Sam Toyer, Johannes Heidecke, Alex Beutel, and Amelia Glaese. Deliberative alignment: Reasoning enables safer language models, 2025. URL https://arxiv.org/abs/2412.16339.

Steve Harris and Andy Seaborne. SPARQL 1.1 Query Language. https://www.w3.org/TR/sparql11-query/, 2013. W3C Recommendation.

Adib Hasan, Ileana Rugina, and Alex Wang. Pruning for Protection: Increasing Jailbreak Resistance in Aligned LLMs Without Fine-Tuning. *arXiv preprint arXiv:2401.10862*, 2024.

B. Hassibi, D.G. Stork, and G.J. Wolff. Optimal brain surgeon and general network pruning. In *IEEE International Conference on Neural Networks*, pp. 293–299 vol.1, 1993. doi: 10.1109/ICNN.1993.298572.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring Massive Multitask Language Understanding, 2021a. URL https://arxiv.org/abs/2009.03300.

Dan Hendrycks, Collin Burns, 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)*, 2021b.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring Mathematical Problem Solving With the MATH Dataset, 2021c. URL https://arxiv.org/abs/2103.03874.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the Real Context Size of Your Long-Context Language Models? *arXiv preprint arXiv:2404.06654*, 2024.

Shengding Hu, Yuge Tu, Xu Han, Ganqu Cui, Chaoqun He, Weilin Zhao, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Xinrong Zhang, Zhen Leng Thai, Chongyi Wang, Yuan Yao, Chenyang Zhao, Jie Zhou, Jie Cai, Zhongwu Zhai, Ning Ding, Chao Jia, Guoyang Zeng, dahai li, Zhiyuan Liu, and Maosong Sun. MiniCPM: Unveiling the potential of small language models with scalable training strategies. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=3X2L2TFr0f.

Shijue Huang, Wanjun Zhong, Jianqiao Lu, Qi Zhu, et al. Planning, creation, usage: Benchmarking llms for comprehensive tool utilization in real-world complex scenarios. *arXiv preprint arXiv:2401.17167*, 2024. URL https://arxiv.org/abs/2401.17167.

Zongle Huang, Lei Zhu, Zongyuan Zhan, Ting Hu, Weikai Mao, Xianzhi Yu, Yongpan Liu, and Tianyu Zhang. Moesd: Unveil speculative decoding's potential for accelerating sparse moe. *arXiv e-prints*, pp. arXiv–2505, 2025.

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.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. R2E-Gym: Procedural Environments and Hybrid Verifiers for Scaling Open-Weights SWE Agents, 2025. URL https://arxiv.org/abs/2504.07164.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? *arXiv preprint arXiv:2310.06770*, 2023.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Hynek Kydlíček, Guilherme Penedo, and Leandro von Werra. Finepdfs. https://huggingface.co/datasets/HuggingFaceFW/finepdfs, 2025.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. RACE: Large-scale ReAding comprehension dataset from examinations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 785–794, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1082. URL https://aclanthology.org/D17-1082.

Hugo Laurençon, Lucile Saulnier, Thomas Wang, Christopher Akiki, Albert Villanova del Moral, Teven Le Scao, Leandro Von Werra, Chenghao Mou, Eduardo González Ponferrada, Huu Nguyen, Jörg Frohberg, Mario Šaško, Quentin Lhoest, Angelina McMillan-Major, Gerard Dupont, Stella Biderman, Anna Rogers, Loubna Ben allal, Francesco De Toni, Giada Pistilli, Olivier Nguyen, Somaieh Nikpoor, Maraim Masoud, Pierre Colombo, Javier de la Rosa, Paulo Villegas, Tristan Thrush, Shayne Longpre, Sebastian Nagel, Leon Weber, Manuel Muñoz, Jian Zhu, Daniel Van Strien, Zaid Alyafeai, Khalid Almubarak, Minh Chien Vu, Itziar Gonzalez-Dios, Aitor Soroa, Kyle Lo, Manan Dey, Pedro Ortiz Suarez, Aaron Gokaslan, Shamik Bose, David Adelani, Long Phan, Hieu Tran, Ian Yu, Suhas Pai, Jenny Chim, Violette Lepercq, Suzana Ilic, Margaret Mitchell, Sasha Alexandra Luccioni, and Yacine Jernite. The bigscience roots corpus: A 1.6tb composite multilingual dataset, 2023. URL https://arxiv.org/abs/2303.03915.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding. *arXiv preprint arXiv:2006.16668*, 2020.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQL. In *Advances in Neural Information Processing Systems*, 2023a.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, et al. Api-bank: A comprehensive benchmark for tool-augmented llms. In *Proceedings of EMNLP 2023*, 2023b. URL https://arxiv.org/abs/2304.08244.

Shiyao Li, Xuefei Ning, Ke Hong, Tengxuan Liu, Luning Wang, Xiuhong Li, Kai Zhong, Guohao Dai, Huazhong Yang, and Yu Wang. Llm-mq: Mixed-precision quantization for efficient llm deployment. In *The Efficient Natural Language and Speech Processing Workshop at NeurIPS*, 2023c.

Xuehai Li, Zi Ye, Xiaoxin Zhang, Xinshi Lu, Yingqiang Xia, Bairu Wu, Shihan Dong, Qipeng Jin, Jialu Wang, Heng Ji, et al. WildChat: 1M ChatGPT Interaction Logs in the Wild. *arXiv preprint arXiv:2405.01470*, 2024.

Ling Team. Every activation boosted: Scaling general reasoner to 1 trillion open language foundation. *arXiv preprint arXiv:2510.22115*, 2025.

Ling-Team. Every activation boosted: Scaling general reasoner to 1 trillion open language foundation. *arXiv preprint arXiv:2510.22115*, 2025.

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. *arXiv preprint arXiv:2305.01210*, 2023. doi: https://doi.org/10.48550/arXiv.2305.01210. URL https://arxiv.org/abs/2305.01210.

Zuxin Liu et al. Toolace: Winning the points of llm function calling. *arXiv preprint arXiv:2409.00920*, 2024.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Weidi Luo, Siyuan Ma, Xiaogeng Liu, Xiaoyu Guo, and Chaowei Xiao. JailBreakV: A Benchmark for Assessing the Robustness of MultiModal Large Language Models against Jailbreak Attacks, 2024. URL https://arxiv.org/abs/2404.03027.

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, 2025. URL https://arxiv.org/abs/2508.15096.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag

Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces, 2026. URL https://arxiv.org/abs/2601.11868.

Yev Meyer and Dane Corneil. Nemotron-Personas-USA: Synthetic personas aligned to real-world distributions, June 2025. URL https://huggingface.co/datasets/nvidia/Nemotron-Personas-USA.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*, 2018a.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a Suit of Armor Conduct Electricity? A New Dataset for Open Book Question Answering, 2018b. URL https://arxiv.org/abs/1809.02789.

MiniMax AI. MiniMax-M2. https://huggingface.co/MiniMaxAI/MiniMax-M2, 2025.

NVIDIA. NeMo Gym: An Open Source Framework for Scaling Reinforcement Learning Environments for LLM. https://github.com/NVIDIA-NeMo/Gym, 2025a. GitHub repository.

NVIDIA. NeMo RL: A Scalable and Efficient Post-Training Library. https://github.com/NVIDIA-NeMo/RL, 2025b. GitHub repository.

NVIDIA. NVIDIA Nemotron Nano 2: An Accurate and Efficient Hybrid Mamba-Transformer Reasoning Model. *arXiv preprint arXiv:2508.14444*, 2025c.

NVIDIA. Fine-Tuning gpt-oss for Accuracy and Performance with Quantization Aware Training. https://developer.nvidia.com/blog/fine-tuning-gpt-oss-for-accuracy-and-performance-with-quantization-aware-training/, 2025. NVIDIA Technical Blog.

NVIDIA. Nemotron 3 nano: Open, efficient mixture-of-experts hybrid mamba-transformer model for agentic reasoning, 2025a. URL https://arxiv.org/abs/2512.20848.

NVIDIA. Nemotron-H: A Family of Accurate and Efficient Hybrid Mamba-Transformer Models, 2025b. URL https://arxiv.org/abs/2504.03624.

NVIDIA. NVIDIA Nemotron 3: Efficient and Open Intelligence, 2025c. URL https://arxiv.org/abs/2512.20856.

NVIDIA. Pretraining large language models with nvfp4, 2025d. URL https://arxiv.org/abs/2509.25149.

NVIDIA. Introducing nvfp4 for efficient and accurate low-precision inference. https://developer.nvidia.com/blog/introducing-nvfp4-for-efficient-and-accurate-low-precision-inference/, June 2025. NVIDIA Developer Blog.

NVIDIA Corporation. Transformerengine pull request #2177: [common] add support for nvfp4 quantization and casting. https://github.com/NVIDIA/TransformerEngine/pull/2177, 2024. Accessed: 2026-02-18.

OpenAI. gpt-oss-120b gpt-oss-20b model card, 2025. URL https://arxiv.org/abs/2508.10925.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training Software Engineering Agents and Verifiers with SWE-Gym, 2025. URL https://arxiv.org/abs/2412.21139.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity's last exam, 2025. URL https://arxiv.org/abs/2501.14249.

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.

Alexandre Piché, Ehsan Kamalloo, Rafael Pardinas, Xiaoyin Chen, and Dzmitry Bahdanau. Pipelinerl: Faster on-policy reinforcement learning for long sequence generation. *arXiv preprint arXiv:2509.19128*, 2025.

Qwen. Qwen2.5 Technical Report, 2025. URL https://arxiv.org/abs/2412.15115.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A Graduate-Level Google-Proof Q&A Benchmark, 2023.

Matthew Richardson, Christopher J.C. Burges, and Erin Renshaw. MCTest: A challenge dataset for the open-domain machine comprehension of text. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 193–203, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D13-1020.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An Adversarial Winograd Schema Challenge at Scale, 2019. URL https://arxiv.org/abs/1907.10641.

Mark Saroufim, Jiannan Wang, Bert Maher, Sahan Paliskara, Laura Wang, Shahin Sefati, and Manuel Candales. Backendbench: An evaluation suite for testing how well llms and humans can write pytorch backends, 2025. URL https://github.com/meta-pytorch/BackendBench.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, et al. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *arXiv preprint arXiv:2402.03300*, 2024.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, Dipanjan Das, and Jason Wei. Language models are multilingual chain-of-thought reasoners, 2022. URL https://arxiv.org/abs/2210.03057.

Zhengliang Shi, Shen Gao, Lingyong Yan, et al. Tool learning in the wild: Empowering language models as automatic tool agents. In *Proceedings of the ACM Web Conference 2025 (WWW '25)*, 2025. URL https://openreview.net/pdf?id=T4wMdeFEjX.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism, 2020. URL https://arxiv.org/abs/1909.08053.

Shivalika Singh, Angelika Romanou, Clémentine Fourrier, David I. Adelani, Jian Gang Ngui, Daniel Vila-Suero, Peerat Limkonchotiwat, Kelly Marchisio, Wei Qi Leong, Yosephine Susanto, Raymond Ng, Shayne Longpre, Wei-Yin Ko, Madeline Smith, Antoine Bosselut, Alice Oh, Andre F. T. Martins, Leshem Choshen, Daphne Ippolito, Enzo Ferrante, Marzieh Fadaee, Beyza Ermis, and Sara Hooker. Global mmlu: Understanding and addressing cultural and linguistic biases in multilingual evaluation, 2024. URL https://arxiv.org/abs/2412.03304.

Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. Reasoning gym: Reasoning environments for reinforcement learning with verifiable rewards. *arXiv preprint arXiv:2505.24760*, 2025.

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. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2459–2475, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.123. URL https://aclanthology.org/2025.acl-long.123/.

Kimi Team. Kimi K2: Open Agentic Intelligence, 2025. URL https://arxiv.org/abs/2507.20534.

Ling Team, Anqi Shen, Baihui Li, Bin Hu, Bin Jing, Cai Chen, Chao Huang, Chao Zhang, Chaokun Yang, Cheng Lin, et al. Every step evolves: Scaling reinforcement learning for trillion-scale thinking model. *arXiv preprint arXiv:2510.18855*, 2025.

NVIDIA The NeMo Data Designer Team. Nemo data designer: A framework for generating synthetic data from scratch or based on your own seed data. https://github.com/NVIDIA-NeMo/DataDesigner, 2025. GitHub Repository.

Changxin Tian, Jiapeng Wang, Qian Zhao, Kunlong Chen, Jia Liu, Ziqi Liu, Jiaxin Mao, Wayne Xin Zhao, Zhiqiang Zhang, and Jun Zhou. WSM: Decay-free learning rate schedule via checkpoint merging for LLM pre-training, 2025. URL https://arxiv.org/abs/2507.17634.

Minyang Tian, Luyu Gao, Shizhuo 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, Yanyu Xiong, Shengzhu Yin, Minhui Zhu, Kilian Lieret, Yanxin Lu, Genglin Liu, Yufeng Du, Tianhua Tao, Ofir Press, Jamie Callan, Eliu Huerta, and Hao Peng. SciCode: A Research Coding Benchmark Curated by Scientists, 2024. URL https://arxiv.org/abs/2407.13168.

Shubham Toshniwal, Ivan Sorokin, Aleksander Ficek, Ivan Moshkov, and Igor Gitman. GenSelect: A Generative Approach to Best-of-N, 2025. URL https://arxiv.org/abs/2507.17797.

Denny Vrandečić and Markus Krötzsch. Wikidata: A Free Collaborative Knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014. doi: 10.1145/2629489.

Boxin Wang, Chankyu Lee, Nayeon Lee, Sheng-Chieh Lin, Wenliang Dai, Yang Chen, Yangyi Chen, Zhuolin Yang, Zihan Liu, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Nemotron-cascade: Scaling cascaded reinforcement learning for general-purpose reasoning models, 2025a. URL https://arxiv.org/abs/2512.13607.

Lean Wang, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. Auxiliary-Loss-Free Load Balancing Strategy for Mixture-of-Experts. *arXiv preprint arXiv:2408.15664*, 2024.

Zhilin Wang, Jiaqi Zeng, Olivier Delalleau, Ellie Evans, Daniel Egert, Hoo-Chang Shin, Felipe Soares, Yi Dong, and Oleksii Kuchaiev. Rlbff: Binary flexible feedback to bridge between human feedback & verifiable rewards. *arXiv preprint arXiv:2509.21319*, 2025b.

Zhilin Wang, Jiaqi Zeng, Olivier Delalleau, Hoo-Chang Shin, Felipe Soares, Alexander Bukharin, Ellie Evans, Yi Dong, and Oleksii Kuchaiev. Helpsteer3-preference: Open human-annotated preference data across diverse tasks and languages. *arXiv preprint arXiv:2505.11475*, 2025c.

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.

Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time, 2022. URL https://arxiv.org/abs/2203.05482.

Zhangchen Xu, Adriana Meza Soria, Shawn Tan, et al. Toucan: Synthesizing 1.5m tool-agentic data from real-world mcp environments, 2025. URL https://arxiv.org/abs/2510.01179.

Weihao Xuan, Rui Yang, Heli Qi, Qingcheng Zeng, Yunze Xiao, Aosong Feng, Dairui Liu, Yun Xing, Junjue Wang, Fan Gao, et al. MMLU-ProX: A Multilingual Benchmark for Advanced Large Language Model Evaluation. *arXiv preprint arXiv:2503.10497*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 Technical Report, 2025. URL https://arxiv.org/abs/2505.09388.

Feng Yao, Liyuan Liu, Dinghuai Zhang, Chengyu Dong, Jingbo Shang, and Jianfeng Gao. Your Efficient RL Framework Secretly Brings You Off-Policy RL Training, August 2025. URL https://fengyao.notion.site/off-policy-rl.

Junjie Ye, Guanyu Li, Songyang Gao, Caishuang Huang, et al. Tooleyes: Fine-grained evaluation for tool learning capabilities of large language models in real-world scenarios. In *Proceedings of the 31st International Conference on Computational Linguistics (COLING 2025)*, 2025. URL https://aclanthology.org/2025.coling-main.12/.

Z.ai / zai-org. Glm-4.7. Hugging Face Model Card, 2025. URL https://huggingface.co/zai-org/GLM-4.7. Open-source large language model with advanced reasoning, coding, and tool calling capabilities.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a Machine Really Finish Your Sentence?, 2019. URL https://arxiv.org/abs/1905.07830.

Jianguo Zhang, Tian Lan, Ming Zhu, et al. xlam: A family of large action models to empower ai agent systems. *arXiv preprint arXiv:2409.03215*, 2024. URL https://arxiv.org/abs/2409.03215.

A. Per-Benchmark Merge Evaluation

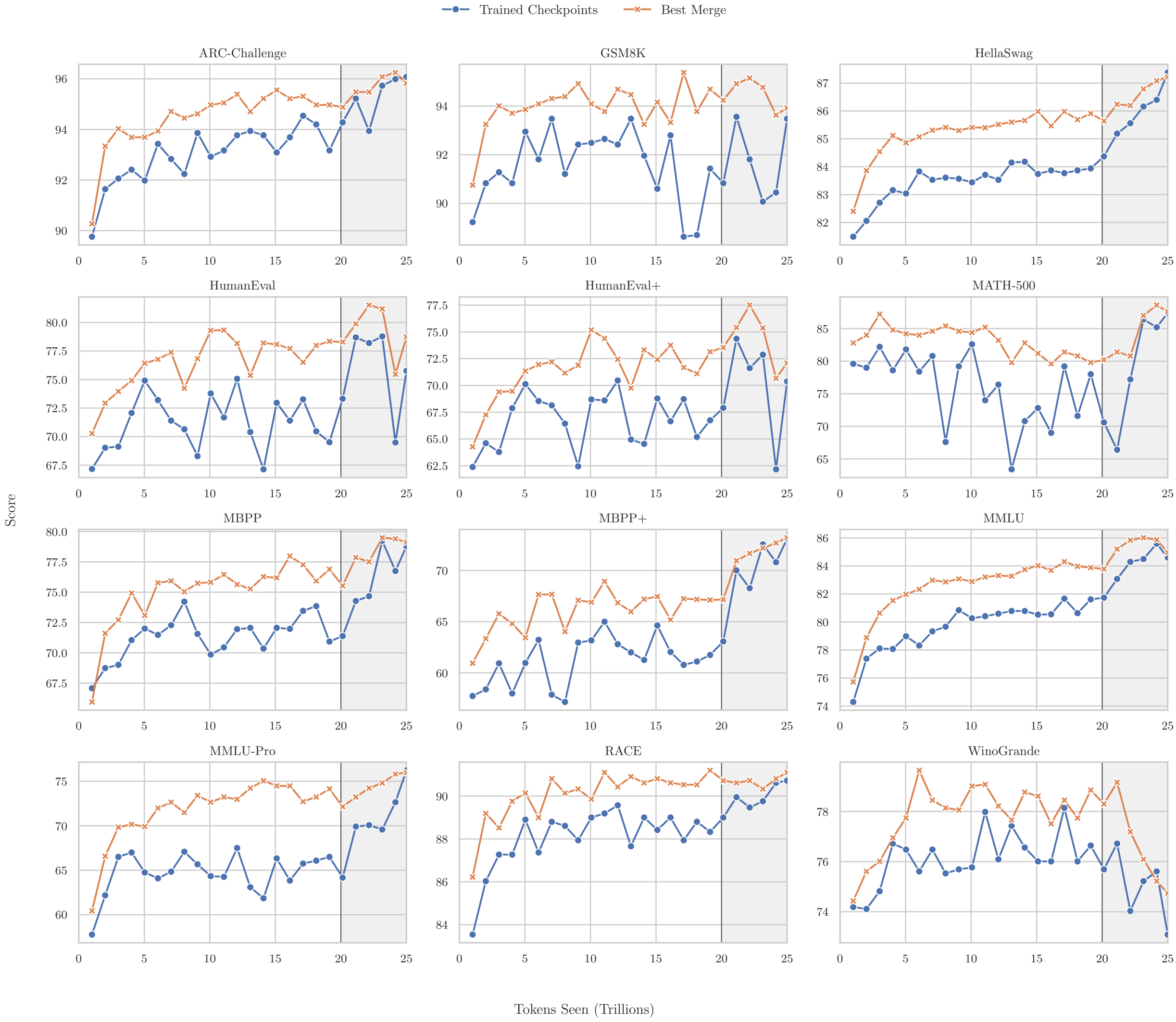


Figure 17 | Per-benchmark accuracy for trained checkpoints versus the best offline checkpoint merge across the full 25T-token training run. The 12 benchmarks span general knowledge (MMLU-Pro, MMLU), code generation (HumanEval, HumanEval+, MBPP, MBPP+), mathematical reasoning (GSM8K, MATH-500), and commonsense understanding (RACE, ARC-Challenge, HellaSwag, WinoGrande). The shaded region indicates the LR decay phase.

B. FP4 Post Training Quantization (PTQ) Algorithm Details

B.1. PTQ Algorithm Ablation

This section shows evaluation accuracy for various PTQ algorithms. In these experiments, all linear layers except the final classification layer and attention linear layers are quantized to NVFP4.

Table 10 | PTQ Algorithm Ablation

Algorithm	**Details**	**MMLU-Pro**	**GPQA**	**LiveCodeBench**	**AA-LCR**
BF16	—	83.49	79.92	72.907	53.00
Default NVFP4 PTQ (Baseline algorithm)	Static per-tensor scales are computed using max-value calibration; per-block scales are computed dynamically from block maximum values.	82.99	79.29	70.18	55.50
Weight per-block scales minimizing MSE	Weight per-block scales are swept to minimize per-block MSE.	83.31	79.92	71.37	56.75
Weight per-block scales to minimize output MSE	Weight per-block scales are swept independently to minimize GEMM output MSE.	83.05	78.98	71.00	57.06
GPTQ	GPTQ (Frantar et al., 2023) is used for weight quantization.	83.11	80.05	69.79	57.87

B.2. AutoQuantize Algorithm

The sensitivity of each operation is measured at its immediate (or closest available) output using a second-order metric:

$$S(\mathrm{Op}_i, Q_{i,f}) = \left(Y_i^{\mathrm{bf16}} - Y_i^{Q_{i,f}}\right)^\top H_i \left(Y_i^{\mathrm{bf16}} - Y_i^{Q_{i,f}}\right)$$

where i indexes operators, $Q_{i,f}$ denotes the quantized operator under format choice f for operator i, and H_i is the local Hessian approximation at the selected measurement point. The measurement point Y_i can be any location where quantization error is compared against BF16; for linear layers, we use the linear-layer output.

Computing the full Hessian is expensive, so we use a diagonal Hessian approximation and estimate it empirically with the diagonal Fisher information matrix. Define

$$\Delta Y_i = Y_i^{\mathrm{bf16}} - Y_i^{Q_{i,f}} \qquad g_i = \nabla_{Y_i}\mathcal{L}$$

where $Y_i, g_i \in \mathbb{R}^d$. The resulting sensitivity proxy is

$$S(\mathrm{Op}_i, Q_{i,f}) \approx \sum_{k=1}^{d} (\Delta Y_{i,k})^2 (g_{i,k})^2$$

The performance cost is defined as:

$$C(\mathrm{Op}_i, Q_{i,f}) = \mathrm{FLOPs}(\mathrm{Op}_i, Q_{i,f})$$

AutoQuantize then solves the constrained optimization:

$$\min_{\{f\}} \sum_i S(\mathrm{Op}_i, Q_{i,f}) \quad \text{s.t.} \quad \sum_i C(\mathrm{Op}_i, Q_{i,f}) \le B$$

where $Q_{i,f}$ is the chosen format for operator i, and B is the total deployment cost budget.

B.2.1. Deployment-Restriction-Aware Search

1) Linear layer fusion. Inference runtimes often fuse linear operators, which imposes a shared quantization format across the fused group. This fusion constraint is applied within each layer: only that layer's Q, K, and V projections are fused and required to share one quantization format. For the fused QKV projection, we model the group as one decision variable and aggregate sensitivity and cost as

$$S(\mathrm{Op}_{\mathrm{qkv}}, Q_{\mathrm{qkv},f}) = S(\mathrm{Op}_{\mathrm{q}}, Q_{\mathrm{qkv},f}) + S(\mathrm{Op}_{\mathrm{k}}, Q_{\mathrm{qkv},f}) + S(\mathrm{Op}_{\mathrm{v}}, Q_{\mathrm{qkv},f})$$
$$C(\mathrm{Op}_{\mathrm{qkv}}, Q_{\mathrm{qkv},f}) = C(\mathrm{Op}_{\mathrm{q}}, Q_{\mathrm{qkv},f}) + C(\mathrm{Op}_{\mathrm{k}}, Q_{\mathrm{qkv},f}) + C(\mathrm{Op}_{\mathrm{v}}, Q_{\mathrm{qkv},f})$$

This formulation is valid under our second-order sensitivity approximation: when the fused operator enforces one shared format but preserves additive branch contributions, the quadratic sensitivity term for the fused output remains additive across Q, K, and V.

2) MoE layer constraints. vLLM and TensorRT-LLM quantized MoE APIs require all sparse experts in a constrained MoE group to share one quantization format. This shared-format restriction is also applied within each MoE layer: only sparse experts inside the same layer are coupled. In Nemotron 3 Super, each sparse expert contains `up_proj` and `down_proj`, and these sparse-expert projections must therefore be assigned jointly. We formulate the sparse-expert set as one operator-level decision,

$$\mathrm{Op}_{\mathrm{moe}} = \bigcup_{e \in \mathcal{E}} \{\mathrm{Op}_{e,\mathrm{up\_proj}}, \mathrm{Op}_{e,\mathrm{down\_proj}}\}$$

where moe denotes the MoE sparse-expert group (all coupled sparse-expert `up_proj`/`down_proj` operators within one MoE layer). We measure sensitivity at the MoE block output so that the metric captures the combined contribution from all sparse experts,

$$S(\mathrm{Op}_{\mathrm{moe}}, Q_{\mathrm{moe},f}) = \left(Y_{\mathrm{moe}}^{\mathrm{bf16}} - Y_{\mathrm{moe}}^{Q_{\mathrm{moe},f}}\right)^{\top} H_{\mathrm{moe}} \left(Y_{\mathrm{moe}}^{\mathrm{bf16}} - Y_{\mathrm{moe}}^{Q_{\mathrm{moe},f}}\right)$$

and define the deployment cost as the sum over sparse experts,

$$C(\mathrm{Op}_{\mathrm{moe}}, Q_{\mathrm{moe},f}) = \sum_{e \in \mathcal{E}} \Big(C(\mathrm{Op}_{e,\mathrm{up\_proj}}, Q_{\mathrm{moe},f}) + C(\mathrm{Op}_{e,\mathrm{down\_proj}}, Q_{\mathrm{moe},f}) \Big)$$

Other linear layers in the MoE block, such as latent projection layers and shared experts, are not part of this sparse-expert coupling constraint and can be assigned different quantization formats.