PRIMING BASE MODELS FOR AGENTIC POST-TRAINING: AN OPEN MIDTRAINING RECIPE

Anonymous authors
Paper under double-blind review

ABSTRACT

Frontier models, general-purpose and domain-specific alike, now include a midtraining stage between pretraining and post-training, but little has been published about how this stage is built. We present an open midtraining recipe that primes base models for agentic post-training, and we show its effectiveness on two Nemotron models of different sizes. Pretraining typically runs at short context and sees almost no agent interactions, while SFT data is mostly synthetic, and very long SFT stages can hurt the model's later RL trainability. Our midtraining stage sits between the two. It trains at 256K context for hundreds of billions of tokens, before the short final 1M extension stage, on a diverse mixture of natural software-engineering data like packed repositories, pull requests, commit histories, and build logs together with a small share of agentic traces. On Nemotron 3.5 Lightning (NVIDIA, 2026b), 500B midtraining tokens improve SWE-bench Verified from 60.7 to 64.5, SWE-bench Multilingual from 55.8 to 62.8, and SWE-bench Pro from 32.0 to 35.5 after identical full SFT. On a larger model based on Nemotron 3 Super, 280B midtraining tokens improve DeepSWE, a harder benchmark of long-horizon tasks, from 11.7 to 15.7 and SWE-bench Verified from 72.9 to 74.3 after identical full SFT. Midtraining also improves robustness across agent harnesses on both models. We report the ablations behind the recipe and an evaluation protocol built on short fixed SFT runs, since base-model proxy metrics did not correlate reliably with post-training results.

1 INTRODUCTION

Between pretraining and post-training, frontier coding models such as Qwen3-Coder-Next (Cao et al., 2026), GLM-5 (GLM-5 Team, 2026), Kimi-Dev (Yang et al., 2026), and Composer 2 (Cursor Research, 2026) include a *midtraining* stage of hundreds of billions of tokens of agentic, long-context data on top of a converged base model. Technical reports describe this stage at a high level, but rarely explain how its data is constructed, filtered, and weighted, or measure the impact of each design choice. Following Qwen-Coder (Hui et al., 2024; Cao et al., 2026), we present an open midtraining recipe for agentic coding, developed and demonstrated on Nemotron base models. We describe how each data source is built, and we report the ablations that guided our training choices.

Base models are usually pretrained at short sequence lengths and then extended to long context in a short final stage before SFT. Nemotron models, for example, are pretrained at 8K and extended to 1M. The data that matters most for software-engineering agents does not fit this pipeline. Pretraining corpora contain almost no agent interactions, and the natural data around software engineering, such as whole repositories, code evolution through commit histories, pull requests, and build logs, often runs to tens or hundreds of thousands of tokens, too long for pretraining and too large for a short long-context stage. Adding agent traces to pretraining is also not straightforward, since pretraining typically applies no loss masking, and it is unclear how the tool responses (environment outputs) in those traces should be handled. SFT, on the other hand, consists mostly of synthetic chat data, and very long SFT stages, at the scale of hundreds of billions of tokens, can, in our experience, reduce the model's later RL trainability.

Midtraining fills this gap. Before the final long-context stage, it trains for hundreds of billions of tokens at 256K, long enough to hold whole repositories, commit histories, and long agent traces (§5.2). The data is mostly natural software-engineering data, with a bounded share of agent traces

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

whose tool responses are masked from the loss. Much of this learning can therefore move out of SFT, so SFT can stay short and preserve the model's later RL trainability.

We develop the recipe on Nemotron 3.5 Lightning-30B-A3B (NVIDIA, 2026b) and then scale it to a larger model based on Nemotron 3 Super-120B-A12B (NVIDIA, 2026c). On N-3.5 Lightning, 500B midtraining tokens improve SWE-bench Verified, Multilingual, and Pro by 3.8, 7.0, and 3.5 points over the same model without midtraining after identical full SFT, with the most sustained gains on SWE-bench Multilingual (§4.3). On N-3 Super, 280B midtraining tokens improve DeepSWE from 11.7 to 15.7 and SWE-bench Verified from 72.9 to 74.3 after identical full SFT (§4.4). Midtraining also improves robustness to the agent harness, which matters for open models that are used with many different harnesses. Both models improve on every harness we tested, and on N-3 Super after 65B tokens and short SFT, the average gain across harnesses is 8.6 points on SWE-bench Verified and 6.8 on Terminal-Bench (§4.5).

We make the following contributions.

- A fully documented midtraining data recipe, showing that a small share of agentic traces adds clear gains on top of natural data (§3 and §4.2).
- An evaluation protocol based on identical SFT runs, since base-model proxy metrics such as bits-per-byte and LoCoDiff did not reliably predict post-training results (§4.1).
- We show that gains hold as we scale both model size and midtraining tokens up to 500B, across benchmarks and agent harnesses (§4).
- Ablations of modeling choices, such as the learning rate, the share of replayed pretraining data, tool-response masking, and midtraining's effect on long-context ability (§5).

2 Related Work

Midtraining in frontier coding models. Although midtraining is growing in popularity, only a few works have presented detailed recipes and ablation results to the community. Qwen3-Coder-Next provides one such recipe, midtraining at a 256K context on trillions of tokens, including 600B tokens of repository-level data across 370 programming languages (Cao et al., 2026). Its recipe suggests combining mined PR data with agentic traces but does not explain exactly how to do so. GLM-5 reports midtraining on approximately 1.55T tokens with progressive context extension, including 160B tokens of software-engineering data constructed from roughly 10M issue–PR pairs (GLM-5 Team, 2026), but does not provide details of the recipe. Kimi-Dev, one of the primary works on which we build, provides a more detailed recipe (Yang et al., 2026). It performs midtraining on approximately 150B tokens of diff patches, commit sequences, and synthetic agent interactions, and shows that a model with this prior needs far fewer SFT tokens than the raw base model to reach the same agent performance. daVinci-Dev advocates trajectory-style midtraining data and matches Kimi-Dev's performance using fewer than half as many tokens (Zeng et al., 2026). Composer 2 shows that code-focused continued pretraining improves downstream RLVR performance (Cursor Research, 2026). AgentFounder similarly shows that continued pretraining benefits search agents (Su et al., 2026), while Liu et al. (2026) study when midtraining effectively bridges the distributions of pretraining and post-training data. Relative to these works, we provide the ablation-level detail missing from the technical reports and evidence that the gains persist through full SFT.

Repository-level data construction. StarCoder2 concatenates repository files in random order (Lozhkov et al., 2024); DeepSeek-Coder orders files by import dependencies (Guo et al., 2024); SPLiCe shows that structured packing improves long-context use (Staniszewski et al., 2025); ProLong randomizes on purpose to avoid recency bias (Gao et al., 2025). Sapronov & Glukhov (2025) find that most of the gain from repository-level pretraining for project-level code completion comes from adapting to a longer context rather than from how the repository context is assembled. Best-Fit Packing reduces truncation harms (Ding et al., 2024), and In-Context Pretraining shows related-document packing helps beyond code (Shi et al., 2024). Our repository data differs in *selection*. We keep only repositories whose files reference each other through imports, and order files so that related code is adjacent (§3).

Evaluating base models for agentic potential. Agentic benchmarks are unreliable before post-training, so Composer 2 selects among base models using held-out codebase perplexity, state track-

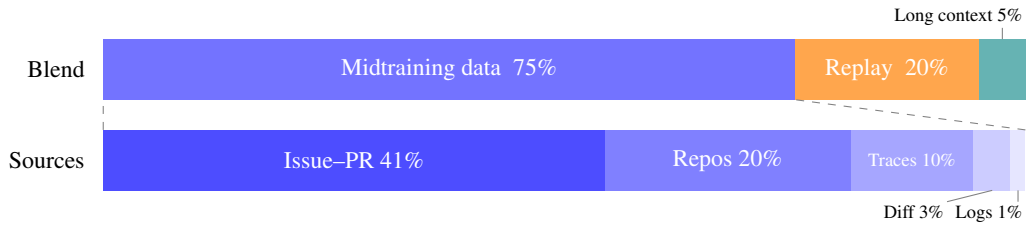

Figure 1: The midtraining blend, by share of training tokens. Replay is drawn from the base model's pretraining data. Token counts and epochs per source are in Appendix A.

ing, and adversarially built QA instead (Cursor Research, 2026). Other benchmarks target specific skills. LoCoDiff measures file-state reconstruction from git history (Mentat AI Team (AbanteAI), 2025), RepoBench and CrossCodeEval measure cross-file completion (Liu et al., 2024; Ding et al., 2023), and APTBench converts agent tasks into a format base models can answer (Qin et al., 2025). Whether such base-model measures predict downstream ability is still contested (Liu et al., 2023; Chen et al., 2025). In our experiments, they did not track post-training results reliably (§4.1).

3 The Midtraining Recipe

Figure 1 summarizes the blend, which is designed for diversity. Most of it is natural software-engineering data, with a small share of synthetic agent traces. We render these traces in the Nemotron chat template and keep them to 10–15% of the blend to avoid overfitting to SFT-like data, which in our experience can hurt later RL trainability. Everything else stays in natural formats, without chat templates. Every source keeps only permissive licenses and goes through decontamination against SWE-bench and Terminal-Bench, secret redaction, and loss masking of content the model should not imitate, such as repeated delimiters, hashes, and timestamps.

Issue–PR. We construct a large-scale code-change corpus from merged pull requests in public GitHub repositories, combining repository snapshots with PR descriptions, linked issues, review discussions, and observed patch evolution when available. We restrict collection to repositories with at least ten stars in supported programming languages. We deduplicate examples by repository and PR identifier and remove records with unavailable repositories, missing Git objects, failed patch reconstruction, empty or excessively large changes, generated content, or other low-quality patches. Each example is laid out as a sequence in which repository metadata, date, file tree, and code context condition the model's reconstruction of the submitted changes. No chat template is applied to this data, but we mask repetitive formatting text from the training loss. In total, this process leaves us with 12.1 million PRs and 200 billion to 2 trillion tokens depending on how much repository context we include in each example. For issue-linked PRs, a structured LLM judge additionally checks that the issue and the patch match, and PR-only examples are filtered with deterministic rules (Appendix A.1).

Repositories. Pretraining corpora have historically presented code as individual files rather than whole repositories, and even when files from one repository are packed together, an 8K pretraining context holds only a small part of it. Agents, however, work inside repositories far longer than that and must use code defined far from where it is called. Midtraining is where the model first reads whole repositories and predicts code that depends on files tens of thousands of tokens earlier. This helps most when the files actually depend on each other, and in many repositories they do not, since many are collections of scripts, exercises, or snippets that never import one another. We start from the repositories in the Nemotron 3 pretraining corpus (NVIDIA, 2025), build an import graph for every repository in 14 languages, and keep only repositories in which at least 20% of code files have an import edge to another file in the repository (10% for Ruby). Kept repositories are serialized with the top-level README first and connected files next to each other, and packed at 256K, with longer repositories split between connected components rather than truncated, giving about 104B tokens. Compared to alphabetical ordering, dependency ordering better separates stronger from weaker models. It lowers bits-per-byte for stronger models and raises it for weaker ones, which

suggests that it exposes cross-file structure that weaker models have not yet learned. The languages, filters, and blend weights are listed in Appendix A.2.

Diff data. Much of software engineering is changing existing code, such as making an edit that stays consistent across several files or keeping track of what a file looks like after a long series of changes. Repository snapshots only show code in its final state, so we build two kinds of examples from commit histories. *Multi-file edits* take a merged pull request, show the model the affected files before the change, and train it to produce them after the change. We keep only changes whose files are linked in the import graph used for the repository data, so that the edits are coupled. *Code-evolution documents* give the initial state of a file, the sequence of real commits that changed it from oldest to newest, and its final state. This trains the long-range state tracking that LoCoDiff measures (Mentat AI Team (AbanteAI), 2025). Histories range from a single commit to several dozen. Diff headers, commit hashes, authors, and dates are masked from the loss. Sources, filters, and statistics are in Appendix A.3.

Build and CI logs. Agents constantly read logs, stack traces, and build output, but PR data lacks the environment's response, so we add raw system logs from the full LogHub archives (Zhu et al., 2023) and CI/CD logs from GitHub Actions run logs and workflow-file histories (Moriconi et al., 2025; Cardoen et al., 2024), plus a small curated slice pairing log windows with diagnoses (Appendix A.4).

Agentic traces. We include agent trajectories to teach interaction patterns that natural software artifacts lack, such as multi-step tool use, testing changes, and recovering from failures. These trajectories are separate from the SFT data. The highest-quality trajectories are reserved for SFT, and midtraining puts the rest to use, including trajectories from earlier teacher models and tasks that do not fit post-training.

The corpus used in our blend contains about 39B tokens (over one million trajectories, about 500K distinct queries, and 30M tool calls, including SWE-Zero and SWE-Hero trajectories (Ludwig et al., 2026)). It covers more than ten agent scaffolds, including OpenHands (Wang et al., 2025), OpenCode, SWE-agent (Yang et al., 2024), Codex, Claude Code, and Terminus, across software engineering, terminal use, search, and general tool use, because traces from a single scaffold transfer poorly to others (Cao et al., 2026). Each trajectory keeps its scaffold's instructions and tool definitions and is rendered in the model's chat template. Appendix A.5 gives the statistics and distributions.

Filtering targets shortcuts and undesirable interaction patterns, including solution leakage through Git history or Internet lookup, unfinished trajectories, malformed tool calls, and repetitive file-inspection loops. As in SFT, we train only on assistant turns, including tool calls, since also training on tool responses hurts results after SFT (§5.3).

3.1 TRAINING

Midtraining starts from a converged base model, so we choose the training setup to add new capabilities without degrading what the base model already knows. We train at 256K context, long enough to hold almost every midtraining document, many of which are far longer than the 8K pretraining context (Figure 2). Our first N-3.5 Lightning experiments start from the 8K pretraining checkpoint, N-3.5 Lightning-8K. For the N-3.5 Lightning scaling run and all N-3 Super runs, we start from the 1M long-context checkpoint, which lets us go straight from midtraining to SFT, keeping evaluations simple. In a mature pipeline, midtraining would come before the final long-context stage (§5.2). We set the replay share to 20% (§5.4). The learning rate warms up briefly and then stays constant, at 5e-5 for N-3.5 Lightning, which favors agentic gains, and 1e-5 for the larger N-3 Super, which favors preserving general capabilities (§5.4). Appendix B lists the full settings.

	SWE-V pass@1 / pass@3	SWE-Multi pass@1 / pass@3	TB 2.0 pass@1 / pass@8
Short SFT (33B tokens)			
Base-8K	57.1 / 69.2	48.6 / 65.0	**22.1** / **40.5**
Midtrained, no traces	56.1 / 68.2	48.4 / 65.3	19.8 / **40.5**
Midtrained	**57.9** / **71.4**	**56.8** / **70.0**	20.1 / 39.3
Full SFT (100B tokens)			
Base-8K	59.7 / 69.8	55.9 / 72.7	22.9 / **44.9**
Midtrained, no traces	60.5 / 72.6	57.1 / 72.7	18.5 / 34.8
Midtrained	**63.5** / **73.2**	**59.6** / **74.0**	**25.1** / 43.8

Table 1: Early experiment on the N-3.5 Lightning-8K checkpoint, midtrained for 100B tokens with and without agentic traces and compared with the base model after identical short or full SFT (%).

4 Experiments and Results

4.1 Evaluation protocol

The success of midtraining is ultimately measured by how well the model does after post-training. Our evaluation protocol is therefore holistic. It runs the full post-training pipeline and also tracks base-model post-trainability metrics and pretraining general-capability benchmarks.

- **Short and full SFT.** We use a general SFT blend (Appendix B.1) at two budgets, 33B tokens (short SFT) and 100B tokens (full SFT), and compare each midtrained model with its starting checkpoint after identical SFT. Full SFT was close to saturation in our runs and supports the main claims, while short SFT gives a quicker signal.
- **Fixed evaluations.** We evaluate on SWE-bench Verified, Multilingual, and Pro, DeepSWE, Terminal-Bench (version 2.0, or 2.1 where noted), and a BrowseComp subset. Unless stated otherwise, SWE-bench runs in OpenHands and Terminal-Bench in Terminus-2, with pass@1 averaged over 3 and 8 rollouts per task (Appendix C.1 gives the setup and references). We also report pass@k, since a higher pass@k has been shown to indicate more room for RL to improve (Yue et al., 2025).
- **Base-model post-trainability metrics.** LoCoDiff (Mentat AI Team (AbanteAI), 2025), RepoBench (Liu et al., 2024), and bits-per-byte on held-out code repositories and on agent traces (Appendix C.2) measure capabilities that midtraining targets, such as file-state tracking and cross-file modeling. For traces we score the decisive turns of DeepSWE trajectories, which correlated best with post-training results. LoCoDiff is averaged over 3 greedy runs on all 198 cases, and the other metrics are single runs. They improve with midtraining (Table 3), but did not track post-training results closely enough to guide recipe decisions.
- **Regression checks.** During midtraining we also track the pretraining general-capability benchmarks from NVIDIA (2026a) in four groups, general knowledge, math, code, and commonsense understanding (Appendix C.1), to catch regressions (§5.4).

4.2 Blend validation on N-3.5 Lightning

Our first experiment midtrains the N-3.5 Lightning-8K base model for 100B tokens, once on the full blend and once without agentic traces, and compares both with the base model after identical SFT (Table 1). Gains seen after short SFT carry over to full SFT, so short SFT can serve as a quick check. Midtraining improves SWE-bench Verified and Multilingual pass@1 by 0.8 and 8.2 points after short SFT, and both gains remain after full SFT (3.8 and 3.7 points). Terminal-Bench is noisier and improves only in full-SFT pass@1. Agentic traces give the majority of the gain after SFT. The no-traces model, trained on natural data alone, gains less than one point on SWE-bench Verified after full SFT, while the full blend improves on it by 3.0, 2.5, and 6.6 points on SWE-bench Verified, Multilingual, and Terminal-Bench. We still keep natural data as the large majority of the blend. Its benefits show up beyond SWE-bench pass@1, in file-state tracking and cross-file modeling (Tables 3 and 4) and robustness across agent harnesses (§4.5), and a blend of mostly traces would amount to a much longer SFT, which in our experience can hurt later RL trainability.

Midtraining tokens (B)	SWE-V pass@1 / pass@3	SWE-Multi pass@1 / pass@3	TB 2.0 pass@1 / pass@8	SWE-Pro pass@1 / pass@3
0 (base)	60.7 / 72.2	55.8 / 72.3	22.6 / 43.8	32.0 / 41.5
100	64.1 / 74.0	57.4 / 71.7	22.5 / 44.9	34.9 / 45.1
200	64.3 / 73.4	59.7 / 76.7	21.4 / 39.3	35.8 / 45.3
300	63.5 / 73.6	62.0 / 76.7	**24.9 / 48.3**	**36.3 / 46.4**
500	**64.5 / 75.0**	**62.8 / 77.0**	23.6 / 46.1	35.5 / 45.4

Table 2: Midtraining scaling on N-3.5 Lightning-LC after identical full SFT. After 500B tokens, midtraining improves both pass@1 and pass@k on every benchmark, so it raises what the model can solve rather than only its reliability.

	LoCoDiff		DeepSWE	Repository	RepoBench
	Exact (%) ↑	Distance ↓	BPB ↓	BPB ↓	↑
N-3.5 Lightning-LC	0.0	0.903	0.171	0.238	70.09
Midtrained, 100B	21.5	0.228	0.144	0.228	71.22
Midtrained, 500B	**26.8**	**0.164**	**0.140**	**0.217**	**72.41**

Table 3: Base-model post-trainability metrics for N-3.5 Lightning-LC and its midtrained checkpoints from the scaling run. All metrics improve as midtraining scales.

Midtraining data	Tokens	Exact (%) ↑	Distance ↓
Blend without diff data	10B	1.3	0.507
50% diff data	10B	**29.8**	**0.150**
Full blend (§4.3)	500B	26.8	0.164

Table 4: LoCoDiff on midtrained N-3.5 Lightning base models. The two 10B-token runs start from N-3.5 Lightning-8K and differ only in whether half of the blend is diff data. Diff data drives most of the LoCoDiff gain.

4.3 Scaling N-3.5 Lightning to 500B tokens

Our main N-3.5 Lightning experiment uses the long-context checkpoint N-3.5 Lightning-LC as the base model (§3.1). The blend has about 380B unique tokens (Appendix A), enough to midtrain for 500B tokens while seeing each natural source about once. We apply identical full SFT to the base model and to midtrained checkpoints from 100B to 500B tokens (Table 2).

Midtraining improves every benchmark over the base model. After 500B tokens, pass@1 rises by 3.8 points on SWE-bench Verified, 7.0 on SWE-bench Multilingual, 3.5 on SWE-bench Pro, and 1.0 on Terminal-Bench, and pass@k rises on all four, by 2.3 to 4.7 points. Most of the SWE-bench Verified and Pro gain arrives by 100B tokens, SWE-bench Multilingual keeps improving up to 500B, and Terminal-Bench is noisier and peaks at 300B. Unlike Table 1, these results start from N-3.5 Lightning-LC. After identical full SFT, its scores differ slightly from those of N-3.5 Lightning-8K, but stay within run-to-run variation. Midtraining also changes how the agent uses its context. Without midtraining, 4.1% of SWE-bench Pro rollouts fail by filling the 256K context window, compared with 0.6% after 500B tokens (Appendix E.1).

The base-model post-trainability metrics show the same trend (Table 3). All four improve over the base model and keep improving from 100B to 500B tokens, reflecting the capabilities that the natural SWE data and traces add. On LoCoDiff, for example, the base model rebuilds no files exactly, while the midtrained model rebuilds 21.5% after 100B tokens and 26.8% after 500B. Most of this gain comes from the diff data, which raises exact match from about 1% to 30% in a controlled 10B-token comparison in Table 4.

4.4 Scaling to N-3 Super 120B

We further apply the recipe to the larger N-3 Super to test how it scales with model size. We midtrain for 65B and 280B tokens with the blend in Figure 1. As in the scaling run, we start from the long-context checkpoint and compare each midtrained model with the base model after identical short and

Midtraining	SWE-V	SWE-Multi	SWE-Pro	DeepSWE	BrowseComp
Short SFT (33B tokens)					
0 (base)	71.3	65.9	52.1	6.4	64.2
65B	71.9	65.6	51.7	10.4	65.0
280B	**72.1**	**67.4**	**52.2**	**12.4**	**67.2**
Full SFT (100B tokens)					
0 (base)	72.9	**69.0**	**53.9**	11.7	61.8
65B	72.5	68.2	52.4	13.5	61.8
280B	**74.3**	67.9	51.8	**15.7**	**64.8**

Table 5: N-3 Super after 65B and 280B midtraining tokens and identical short or full SFT (pass@1, %). Midtraining gives a large gain on DeepSWE, the hardest benchmark, and also improves SWE-bench Verified and BrowseComp.

	N-3 Super, short SFT						N-3.5 Lightning-8K, full SFT			
	SWE-bench Verified			Terminal-Bench 2.1			SWE-bench Verified			
Harness	Base	Mid 65B	Δ	Base	Mid 65B	Δ	Base	Mid 100B (no traces)	Mid 100B	Δ
Pi	57.4	67.8	+10.4	30.3	35.2	+4.9	45.7	48.8	59.9	+14.2
OpenCode	54.4	65.8	+11.4	35.2	38.2	+3.0	52.0	50.1	63.1	+11.1
Hermes	59.8	63.2	+3.4	44.2	53.9	+9.7	43.3	43.3	50.0	+6.7
Codex	48.4	61.8	+13.4	27.0	39.7	+12.7	42.1	45.9	61.1	+19.0
Claude Code	53.8	58.0	+4.2	39.7	43.5	+3.8	46.0	50.7	60.9	+14.9
OpenHands	–	–	–	–	–	–	59.7	60.5	63.5	+3.8
Mean	54.8	**63.3**	+8.6	35.3	**42.1**	+6.8	48.1	49.9	**59.8**	+11.6

Table 6: **Harness robustness:** midtraining improves pass@1 on every agent harness we evaluated, for both models, with the largest gain on the harness where the base model is weakest (%).

full SFT (Table 5). The clearest gain is on DeepSWE (Huang et al., 2026), the hardest benchmark, whose long-horizon tasks are written from scratch. After full SFT, 280B tokens of midtraining raise it from 11.7 to 15.7, and after short SFT from 6.4 to 12.4. SWE-bench Verified also improves, from 72.9 to 74.3 after full SFT, and BrowseComp by 3 points after both short and full SFT, while SWE-bench Multilingual and Pro stay within about two points.

4.5 Harness robustness

Open models are used with many agent harnesses, not only the one their SFT data was built for, so harness robustness is a goal of our midtraining and the reason we draw traces from many scaffolds. Table 6 shows that midtraining makes both models more robust. On N-3 Super after short SFT, the midtrained model is better on every harness, by 8.6 points on average on SWE-bench Verified and 6.8 on Terminal-Bench 2.1. On N-3.5 Lightning-8K after full SFT, midtraining improves SWE-bench Verified on all six harnesses, by 11.6 points on average. Midtraining on the natural data alone already improves over the base model on four of the six harnesses, by up to 4.7 points, and adding the traces gives the largest gains. On SWE-bench Verified, midtraining also narrows the spread across harnesses (from 11.4 to 9.8 points on N-3 Super and from 17.6 to 13.5 on N-3.5 Lightning-8K), so the midtrained model depends less on the choice of harness. We did not train on traces from a single scaffold, so these results do not separate the effect of scaffold diversity from that of adding traces.

5 Analysis and Ablations

5.1 Persistence of gains through reinforcement learning

As a preliminary check on whether midtraining gains persist under RL, we run an identical short stage of RL with verifiable rewards (RLVR), 20 GRPO steps (Shao et al., 2024) over 14 verifiable environments (Appendix D), from N-3.5 Lightning-8K and its midtrained checkpoint after full SFT.

SWE-V (pass@1 / pass@3)	SFT	RL step 10	Step 15	Step 20
Base-8K	61.2 / 71.8	60.4 / 70.8	61.4 / 72.2	60.4 / 71.7
Midtrained-100B	**63.2 / 74.1**	**63.0 / 74.0**	**63.5 / 73.7**	**63.0 / 73.0**
TB 2.1 (pass@1 / pass@8)	SFT	RL step 10	Step 15	Step 20
Base-8K	23.0 / **46.1**	**24.7** / 44.9	22.9 / 41.6	20.8 / 39.3
Midtrained-100B	**24.2** / 44.9	23.5 / 44.9	**24.4 / 44.9**	**23.2 / 46.1**

Table 7: RLVR from N-3.5 Lightning-8K and its midtrained checkpoint after full SFT. SFT-start scores are re-evaluated with the RL checkpoints. The midtrained model leads on SWE-bench, and its Terminal-Bench pass@8 holds.

	8K	16K	32K	64K	128K	256K
8K → long-context @1M	**94.1**	**91.9**	**90.5**	85.9	81.3	76.6
8K → midtraining @256K	92.3	91.4	87.8	81.9	76.1	62.2
8K → midtraining @256K → long-context @1M	92.1	91.0	89.8	**86.6**	**83.8**	**79.7**

Table 8: N-3.5 Lightning RULER scores up to 256K. Midtraining alone brings N-3.5 Lightning-8K close to N-3.5 Lightning-LC up to 128K, acting like a long-context stage.

Twenty steps is too short to measure an RL gain, so we only ask whether the midtrained model's advantage survives the start of RL.

The midtrained model starts about two points ahead on SWE-bench Verified pass@1 and keeps that lead at every RL step, and both models stay close to their SFT scores (Table 7). On Terminal-Bench 2.1, the base model's pass@8 falls from 46.1 to 39.3 over 20 steps while the midtrained model's holds at 45 to 46, although with 89 tasks this drop amounts to only six tasks. Training-time signals, including mean reward, generation length, and format violations, also favor the midtrained model (Appendix D).

With a single seed and 20 steps, these results do not show that midtraining increases what RL can add, only that the midtrained model's advantage after SFT is not lost when RL begins.

5.2 Midtraining as a long-context stage

Because midtraining trains at 256K, it also extends the model's context. Table 8 compares the usual order, where N-3.5 Lightning-8K goes straight to the long-context stage, with a 100B-token N-3.5 Lightning midtraining run at 256K before a long-context stage. Midtraining alone brings N-3.5 Lightning-8K within about five points of the long-context base model on RULER (Hsieh et al., 2024) up to 128K, but trails by 14 points at 256K. After the long-context stage, the midtrained model beats the long-context base model at every length from 64K to 256K and stays within two points at shorter lengths. Midtraining thus does part of the work of the long-context stage, and this carries over to the final model. Appendix E.2 gives more results for N-3 Super.

5.3 Masking tool responses in traces

Agent traces interleave the model's own actions with tool responses, the outputs the environment returns after each tool call. Some recent work also trains on tool responses, so that the model learns to predict how the environment reacts, a form of world modeling (FAIR CodeGen Team, 2025; Zeng et al., 2026).

We tested this with the setup of §4.2, midtraining N-3.5 Lightning-8K for 100B tokens with and without tool responses in the loss, followed by identical SFT with an earlier version of our SFT recipe. In our experiments, training on tool responses did not work out of the box and lowered SWE-bench Verified pass@1 by about 10 points (Table 9), so we train only on assistant turns, as

SWE-V	pass@1	pass@3
Unmasked	27.5	41.6
Masked	**37.8**	**53.0**

Table 9: Tool-response masking.

Midtraining	Knowledge	Math	Code	Commonsense
0 (base)	79.7	86.4	**80.8**	**87.5**
25B	**79.9**	85.9	80.0	87.1
65B	79.8	**87.0**	79.6	86.7

Table 10: N-3 Super pretraining benchmarks before and after midtraining with 20% replay.

LR	MMLU	MMLU-Pro	MATH-500
Base	**79.4**	64.3	**79.7**
1e-5	78.5	**66.3**	78.8
3e-5	76.7	61.5	74.3
5e-5	76.8	63.3	61.7

Table 11: N-3.5 Lightning-8K after 100B midtraining tokens at three learning rates.

in SFT. Tool responses are often long and noisy, such as file dumps, test logs, and error traces, and predicting them may take capacity away from learning the agent's own actions.

5.4 Pretraining replay and learning rate

Replay keeps the model close to its pretraining distribution, so that midtraining shifts the data distribution gradually rather than abruptly. We started from 20% replay as a reasonable default, with the goal of no regression on general-capability benchmarks. Table 10 shows that N-3 Super does not regress through 65B tokens, with every benchmark group within 1.2 points of the starting checkpoint. We also tried 40% replay, which gave no clear benefit over 20%, so we kept 20%.

The learning rate also affects general capabilities. Pretraining ends with a small, decayed learning rate, and midtraining needs a higher one to learn the new data. In our early runs, midtraining at 1e-5 gave smaller gains after SFT than higher rates did. Raising the rate too far, however, hurts general capabilities even with replay. Table 11 compares a constant 1e-5 with warming up to 3e-5 or 5e-5. Both higher rates lower general capabilities, most clearly MATH-500, which falls from 79.7 to 61.7 at 5e-5. We therefore recommend a moderate increase and checking general-capability benchmarks while tuning it.

6 Conclusion

We presented an open midtraining recipe for agentic coding, with full details of how each data source is built, filtered, and weighted and how the models are trained. After identical SFT, the recipe gives clear gains as midtraining scales to 500B tokens on N-3.5 Lightning, and it carries over to the larger N-3 Super, where it raises DeepSWE, the hardest benchmark we evaluate, from 11.7 to 15.7 and also improves SWE-bench Verified and BrowseComp. On both models, midtraining improves robustness on every agent harness we tested. Our experiments show that agentic traces are essential, since natural software-engineering data alone gives much smaller gains after SFT, and that a short SFT run gives a more direct post-training signal than base-model post-trainability metrics, which remain useful to track. Midtraining at 256K also works as an intermediate long-context stage, and we explain our other modeling choices, such as tool-response masking and pretraining replay.

Limitations and future work. Agentic benchmarks are noisy, and each comparison is a single run, so differences of one or two points, especially on Terminal-Bench, may come from run-to-run variation. We did not compare against spending the same tokens on longer pretraining or SFT, since midtraining adds a separate stage that complements SFT rather than replacing it. We tested the recipe only on Nemotron models, not on other model families, and mainly on agentic coding. Our RL results are also limited, since clear RL gains on top of a strong SFT checkpoint were difficult to get at the N-3.5 Lightning scale and RL was too compute intensive at the N-3 Super scale, so we report only a short, single-seed run on N-3.5 Lightning-8K. For future work, since traces drove most of the gains, we plan to scale trace data with more diversity, and we have already expanded our trace corpus to about 90B tokens. We also plan to extend the agentic data to more capabilities and to move more post-training data, such as reasoning and chat data, into midtraining.

Reproducibility Statement

Appendix A documents every data source, including how it is built, filtered, and weighted, and Appendix B lists the midtraining and SFT settings. Appendix C.1 gives the evaluation harnesses, sam-

pling settings, and number of rollouts, and Appendix C.2 describes the base-model post-trainability metrics.

AI USE STATEMENT

Large language models were used as writing assistants under author supervision. The authors wrote the initial drafts, and language models helped revise and polish sections of the paper based on those drafts and on the authors' notes and experiment records. They also assisted with literature search and review, and helped assemble and check the bibliography. The authors reviewed all claims, numbers, and final text and checked them against the underlying experiment outputs. The research itself, including the design of the recipe, the experiments, and the analysis, was carried out by the authors. Parts of the midtraining data are synthetic and were generated with large language models (§3). AI coding tools also assisted with data curation and with running evaluations, within pipelines that the authors designed, built, and verified, and helped monitor training and evaluation jobs launched by the authors. The authors take full responsibility for all content of this paper.

REFERENCES

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

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020. URL https://arxiv.org/abs/1911.11641.

Ruisheng Cao et al. Qwen3-Coder-Next technical report. *arXiv preprint arXiv:2603.00729*, 2026.

Guillaume Cardoen, Tom Mens, and Alexandre Decan. A dataset of GitHub Actions workflow histories. In *Proceedings of the 21st International Conference on Mining Software Repositories (MSR)*, pp. 677–681. ACM, 2024. doi: 10.1145/3643991.3644867. Dataset version 2024-10-25, Zenodo record 13985548, https://doi.org/10.5281/zenodo.13985548.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Yangyi Chen et al. Scaling laws for predicting downstream performance in LLMs. *Transactions on Machine Learning Research*, 2025. URL https://arxiv.org/abs/2410.08527.

Neil Chowdhury et al. Introducing SWE-bench Verified. https://openai.com/index/introducing-swe-bench-verified/, 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 preprint arXiv:1803.05457*, 2018.

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

Cursor Research. Composer 2 technical report. *arXiv preprint arXiv:2603.24477*, 2026.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, et al. SWE-Bench Pro: Can AI agents solve long-horizon software engineering tasks? In *International Conference on Machine Learning (ICML)*, 2026. URL https://arxiv.org/abs/2509.16941.

Hantian Ding et al. Fewer truncations improve language modeling. In *International Conference on Machine Learning (ICML)*, 2024. URL https://arxiv.org/abs/2404.10830.

Yangruibo Ding et al. CrossCodeEval: A diverse and multilingual benchmark for cross-file code completion. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023. URL https://arxiv.org/abs/2310.11248.

FAIR CodeGen Team. CWM: An open-weights LLM for research on code generation with world models. *arXiv preprint arXiv:2510.02387*, 2025.

Tianyu Gao, Alexander Wettig, Howard Yen, and Danqi Chen. How to train long-context language models (effectively). In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2025. URL https://arxiv.org/abs/2410.02660.

GLM-5 Team. GLM-5: From vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Daya Guo et al. DeepSeek-Coder: When the large language model meets programming – the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations (ICLR)*, 2021a. URL https://arxiv.org/abs/2009.03300.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2021b. 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? In *Conference on Language Modeling (COLM)*, 2024. URL https://arxiv.org/abs/2404.06654.

Wenqi Huang, Charley Lee, Leonard Tng, and Serena Ge. DeepSWE: Measuring frontier coding agents on original, long-horizon engineering tasks. *arXiv preprint arXiv:2607.07946*, 2026.

Binyuan Hui, Jian Yang, Zeyu Cui, et al. Qwen2.5-Coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

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. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2403.07974.

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? In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2310.06770.

Emmy Liu, Graham Neubig, and Chenyan Xiong. Midtraining bridges pretraining and posttraining distributions. In *International Conference on Machine Learning (ICML)*, 2026. URL https://arxiv.org/abs/2510.14865.

Hong Liu, Sang Michael Xie, Zhiyuan Li, and Tengyu Ma. Same pre-training loss, better downstream: Implicit bias matters for language models. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, pp. 22188–22214, 2023.

Tianyang Liu, Canwen Xu, and Julian McAuley. RepoBench: Benchmarking repository-level code auto-completion systems. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2306.03091.

Anton Lozhkov et al. StarCoder 2 and The Stack v2: The next generation. *arXiv preprint arXiv:2402.19173*, 2024.

Nikolai Ludwig, Wasi Uddin Ahmad, Somshubra Majumdar, and Boris Ginsburg. From SWE-ZERO to SWE-HERO: Execution-free to execution-based fine-tuning for software engineering agents. In *Conference on Language Modeling (COLM)*, 2026. URL https://arxiv.org/abs/2604.01496.

Mentat AI Team (AbanteAI). LoCoDiff: Natural long context code bench. https://abanteai.github.io/LoCoDiff-bench/, 2025.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, et al. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2601.11868.

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 *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018. URL https://arxiv.org/abs/1809.02789.

Florent Moriconi, Thomas Durieux, Jean-Rémy Falleri, Raphaël Troncy, and Aurélien Francillon. GHALogs: Large-scale dataset of GitHub Actions runs. In *Proceedings of the 22nd IEEE/ACM International Conference on Mining Software Repositories (MSR)*, pp. 669–673. IEEE, 2025. doi: 10.1109/MSR66628.2025.00104.

NVIDIA. NVIDIA Nemotron 3: Efficient and open intelligence. *arXiv preprint arXiv:2512.20856*, 2025.

NVIDIA. Nemotron 3 Super: Open, efficient mixture-of-experts hybrid Mamba-transformer model for agentic reasoning. *arXiv preprint arXiv:2604.12374*, 2026a.

NVIDIA. NVIDIA Nemotron 3.5 Lightning 30B-A3B Base. https://huggingface.co/nvidia/NVIDIA-Nemotron-3.5-Lightning-30B-A3B-Base-BF16, 2026b.

NVIDIA. NVIDIA Nemotron 3 Super 120B-A12B Base. https://huggingface.co/nvidia/NVIDIA-Nemotron-3-Super-120B-A12B-Base-BF16, 2026c.

Jiarui Qin et al. APTBench: Benchmarking agentic potential of base LLMs during pre-training. *arXiv preprint arXiv:2510.24397*, 2025.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020. URL https://arxiv.org/abs/1907.10641.

Maksim Sapronov and Evgeniy Glukhov. On pretraining for project-level code completion. *arXiv preprint arXiv:2510.13697*, 2025.

Zhihong Shao et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Weijia Shi et al. In-context pretraining: Language modeling beyond document boundaries. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2310.10638.

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

Konrad Staniszewski et al. Structured packing in LLM training improves long context utilization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2025. URL https://arxiv.org/abs/2312.17296.

Liangcai Su et al. Scaling agents via continual pre-training. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2509.13310.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, et al. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2407.16741.

Xingyao Wang, Simon Rosenberg, Juan Michelini, Calvin Smith, Hoang Tran, Engel Nyst, Rohit Malhotra, Xuhui Zhou, et al. The OpenHands software agent SDK: A composable and extensible foundation for production agents. In *Proceedings of Machine Learning and Systems (MLSys)*, 2026. URL https://arxiv.org/abs/2511.03690.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, et al. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024. URL https://arxiv.org/abs/2406.01574.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, et al. BrowseComp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*, 2025.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://arxiv.org/abs/2405.15793.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, et al. SWE-smith: Scaling data for software engineering agents. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2025. URL https://arxiv.org/abs/2504.21798.

Zonghan Yang et al. Kimi-Dev: Agentless training as skill prior for SWE-Agents. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2509.23045.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://arxiv.org/abs/2504.13837.

Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Linhao Zhang, Shulin Xin, Lu Chen, Qi Liu, et al. Multi-SWE-bench: A multilingual benchmark for issue resolving. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2025. URL https://arxiv.org/abs/2504.02605.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019. URL https://arxiv.org/abs/1905.07830.

Ji Zeng et al. daVinci-Dev: Agent-native mid-training for software engineering. In *International Conference on Machine Learning (ICML)*, 2026. URL https://arxiv.org/abs/2601.18418.

Jieming Zhu, Shilin He, Pinjia He, Jinyang Liu, and Michael R. Lyu. Loghub: A large collection of system log datasets for AI-driven log analytics. In *Proceedings of the 34th IEEE International Symposium on Software Reliability Engineering (ISSRE)*, pp. 355–366. IEEE, 2023. doi: 10.1109/ISSRE59848.2023.00071. URL https://github.com/logpai/loghub.

Source	Unique tokens	Share of blend	Tokens seen (500B run)	Epochs
Issue–PR	216.9B	40.8%	205.2B	0.95
Repositories	104.1B	20.0%	100.7B	0.97
Agentic traces	39.4B	10.0%	50.3B	1.28
Diff data	16.0B	3.0%	15.1B	0.95
Build and CI logs	6.5B	1.2%	6.1B	0.95
Pretraining replay	∼2.0T	20.0%	100.7B	0.05
Long-context data	67.4B	5.0%	25.2B	0.37

Table 12: Sources of the midtraining blend in Figure 1. Replay is drawn from the base model's final-phase pretraining blend. Epochs are tokens seen divided by unique tokens.

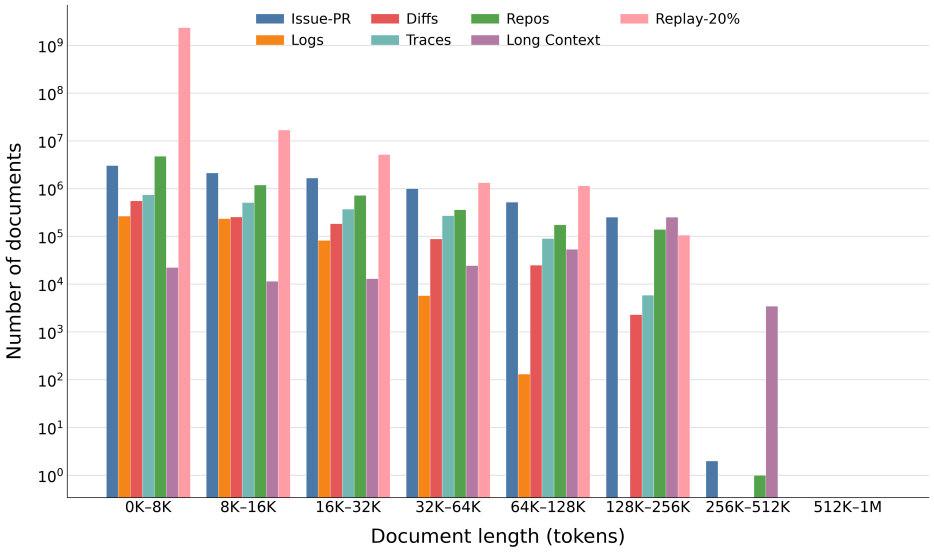

Figure 2: Document lengths in tokens for each source in the midtraining blend (log scale).

A Midtraining data

A.1 Issue–PR

For PRs linked to an issue, an LLM judge checks whether the issue and the PR match. The judge reads the full issue and PR text, the final patch, and the paths of the changed files, and labels each pair as clean or assigns one of seven failure modes, namely a revert, an incidental issue link, a wrong issue reference, a partial fix, an issue with no clear action, an underspecified issue, or a PR that does much more than the issue asks. We keep clean pairs and remove pairs that cannot be fixed. We repair the others by linking the PR to the correct issue, narrowing its scope, keeping it as a PR-only example, or splitting it from unrelated changes. For PRs without a linked issue, we remove those that change only README files or three lines or fewer.

A.2 Repositories

The repository data starts from the code repositories in the Nemotron 3 pretraining corpus (NVIDIA, 2025), a deduplicated collection of public repositories with permissive licenses, and covers 14 languages (Table 13).

Connectedness. For each language, we parse the references that link one file to another, such as imports in Python, includes in C and C++, and module paths in Go. A repository's connected fraction is the share of its code files that import another file in the repository, and we keep repositories where this share is at least 20%. For Ruby we lower the threshold to 10%, because Rails autoloading

hides most references from static parsing. We exclude languages with little cross-file structure that parsing can recover, such as Shell, R, and Swift, as well as markup and configuration languages.

Serialization. The code files follow the README in depth-first order over the import graph, so the files of each connected component stay together and each file sits near the files it imports. Each file starts with a header that gives the repository name and file path, and the header is masked from the loss.

Filtering. We deduplicate files by path within each repository and drop Python 2 code, removing the whole repository when most of it is Python 2. We drop repositories that contain server-side secrets, such as cloud or API keys, and redact client-side keys, which are common in mobile and web projects. We also strip license and auto-generated headers, and drop repositories above 2M tokens or those that lose most of their files in deduplication. To avoid contamination, we remove SWE-bench repositories and their forks, as well as files that contain HumanEval, MBPP, or LiveCodeBench (Jain et al., 2025) solutions.

Blend weights. We start from each language's natural token share, upweight Python, and downweight Dart, Scala, and Lua (Table 13).

Language	Records	Tokens (B)	Natural share (%)	Blend weight (%)
Python	1,798,331	24.51	23.5	31.0
Java	945,513	18.71	18.0	17.0
TypeScript	1,273,349	13.20	12.7	12.0
C#	460,597	11.66	11.2	10.0
JavaScript	1,289,843	10.01	9.6	8.6
C++	400,540	8.27	7.9	7.6
PHP	329,934	5.30	5.1	4.7
Dart	386,648	3.24	3.1	1.4
C	106,799	2.81	2.7	2.7
Rust	115,314	2.22	2.1	1.9
Go	119,005	1.68	1.6	1.5
Ruby	89,846	1.00	1.0	0.9
Scala	39,618	0.99	1.0	0.4
Lua	21,770	0.51	0.5	0.3
Total	7,377,107	104.12	100.0	100.0

Table 13: Repository data by language. Tokens are measured with the model tokenizer.

A.3 Diff data

Multi-file edits come from merged pull requests in the Issue–PR corpus, with the full patch and every changed file before the change. Code-evolution documents come from the full git histories of 6,550 permissively licensed repositories from the same corpus, chosen to match the language mix of SWE-bench Multilingual. We keep only patches that apply exactly and histories that replay byte for byte. In multi-file edits, the loss covers the original files, the diffs, and the changed lines of the new files with two lines of context. In code-evolution documents, it covers the commit messages, diffs, and final state, and license blocks are masked in both. We remove all repositories used in SWE-bench Verified, SWE-bench Pro, Multi-SWE-bench (Zan et al., 2025), Terminal-Bench, and LoCoDiff, deduplicate examples, and drop GPL-licensed documents, secrets, lockfiles, and generated files. Histories have a median of 3 commits and up to 59, and the median document is about 6K tokens (Table 14).

A.4 Build and CI logs

Raw logs come from the full LogHub archives (BGL, Thunderbird, HPC, HDFS, Spark, and Windows). We cut them into non-overlapping windows of 120 context lines followed by 16 target lines. We deduplicate HDFS logs after masking block identifiers, hosts, and timestamps. No single archive is more than 28% of this source. Supercomputer logs are at least 30%, and CI/CD logs from

Type	Examples	Document tokens	Trained tokens	Trained share
Multi-file edits	733,705	13.3B	6.8B	51%
Code evolution	377,618	3.5B	3.2B	91%
Total	1,111,323	16.8B	10.0B	60%

Table 14: Diff data used in our experiments. Trained tokens exclude masked context. Counts use the tokenizer of our data pipeline. With the model tokenizer, the total is 16.0B (Table 12).

Scaffold label	Trajectories	Share (%)
`codex`	166,624	19.39
`openhands`	120,086	13.98
`generic_code_cli`	107,291	12.49
`gemini_cli`	104,717	12.19
`terminus`	99,915	11.63
`taubench_react`	50,399	5.87
`swe_agent`	46,162	5.37
`opencode`	44,920	5.23
`mini_swe_agent`	35,260	4.10
`search_browse`	26,168	3.05
`unknown`	25,866	3.01
`chat`	20,426	2.38
`claude_code`	11,342	1.32
Total	859,176	100.00

Table 15: Trajectories by scaffold label in the early snapshot. Shares are of trajectories, not tokens.

GHALogs run logs and workflow-file histories are at least 12%. We check every source against a license registry and exclude sources with unknown licenses. We also build 632 diagnosis examples. Each centers a window of HPC logs on a warning, error, or critical event, and a frontier model writes the diagnosis from the window and from findings that simple rules extract. We redact secrets and mask timestamps, hostnames, hashes, and repeated boilerplate from the loss.

A.5 Agentic traces

The blend in Table 12 uses 39.4B tokens of traces. The statistics below come from an early filtered snapshot of 859,176 trajectories. Scaffold labels name the agent interface, not the task domain. For example, a software-engineering task can use shell commands, file-editing tools, or both. We downsample general tool-use data, mainly `taubench_react`, to give more weight to software engineering. Tables 15, 16, and 17 give the scaffolds, the most frequent tools, and the number of assistant turns per trajectory. In the snapshot, filtering (§3) removed 8,935 trajectories, about 1%, mostly unfinished runs (6,960) or repeated loops that ended in failure (1,959).

B Training setup

Table 18 lists the settings for midtraining and SFT. All runs use Megatron-LM (Shoeybi et al., 2019) on NVIDIA GB300 GPUs, with inter-document masking so that documents packed into the same sequence do not attend to each other. Both models keep their two multi-token-prediction (MTP) layers during midtraining.

B.1 SFT data

Both models use the same SFT data and settings. The SFT blend is general-purpose, with 60% agentic data and 40% general data by tokens (Figure 3). The agentic part has about 430K trajectories from several scaffolds, most of them generated by GLM-5.2. They cover software engineering, terminal use, search, conversational tool use, and knowledge work. The general part has about 1.7M examples. The reasoning data covers math, proofs, HLE-style questions, and code, generated by

Tool name	Calls	Share (%)
`bash`	3,659,199	19.6
`execute_bash`	3,633,328	19.4
`str_replace_editor`	2,913,745	15.6
`keystrokes`	2,029,095	10.9
`exec_command`	736,655	3.9
`write`	524,113	2.8
`browser.search`	317,169	1.7
`shell`	280,803	1.5
`read`	261,285	1.4
`todowrite`	260,147	1.4

Table 16: The ten most frequent tool names in the early snapshot, as each scaffold records them, so similar tools such as `bash` and `execute_bash` are listed separately. Shares are of all tool calls.

Assistant turns	Trajectories	Share (%)
1	146,334	17.03
2	56,929	6.63
3	47,362	5.51
4–5	77,700	9.04
6–10	138,036	16.07
11–20	111,777	13.01
21–50	161,295	18.77
51–100	105,711	12.30
101–200	12,833	1.49
201–300	1,195	0.14
301–400	4	< 0.01
401+	0	0.00
Total	859,176	100.00

Table 17: Trajectory length in assistant turns, in the early snapshot.

DeepSeek-V4-Pro. The chat and instruction-following data uses responses from GLM-5 and gpt-oss. The rest is long-context, safety, multilingual, and identity data.

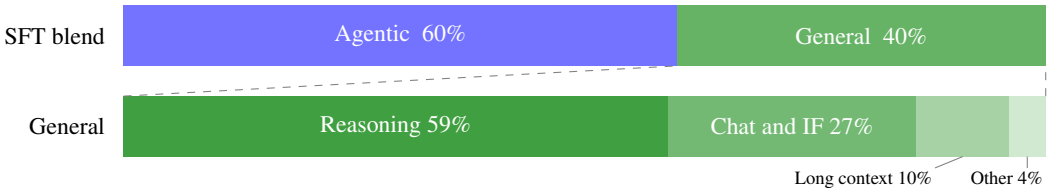

Figure 3: The SFT blend. The top bar shows token shares. The lower bar splits the general data by number of examples. IF is instruction following, and Other is safety, multilingual, and identity data.

C Evaluation

C.1 Agentic benchmarks

For all agentic benchmarks, we sample with temperature 1.0 and top-p 0.95. SWE-bench (Jimenez et al., 2024) Verified (Chowdhury et al., 2024) (500 tasks), SWE-bench Multilingual (Yang et al., 2025) (300), and SWE-bench Pro (Deng et al., 2026) (731) use the OpenHands agent (Wang et al., 2025; 2026) (SDK version 1.17.0) with 3 rollouts per task. Each rollout has at most 200 turns, or 250 for SWE-bench Pro. Terminal-Bench 2.0 (Merrill et al., 2026) (89 tasks) uses the Terminus-2 agent (Merrill et al., 2026) with a one-hour limit per task and 8 rollouts per task. The harness sweep and the RL evaluation use Terminal-Bench 2.1, which has the same 89 tasks with 28 of them fixed, and we compare scores only within a version. DeepSWE (Huang et al., 2026) pass@1 is averaged

	N-3.5 Lightning midtraining	N-3 Super midtraining	SFT (both models)
Parameters (total / active)	30B / 3B	120B / 12B	–
Sequence length (packed)	256K	256K	256K
Global batch size (sequences)	64	64	64
Peak learning rate	5×10^{-5}	1×10^{-5}	2×10^{-5}
Warmup (sequences)	32K	30K	–
Schedule after warmup	constant	constant	cosine to 5×10^{-6}
Optimizer	Adam, $\beta = (0.9, 0.95)$, weight decay 0.1, gradient clip 1.0		
Precision	BF16		
Parallelism (TP / CP / EP)	2 / 4 / 8	2 / 8 / 64	–
MTP layers (loss weight)	2 (0.1)	2 (0.3)	–
Tokens	up to 500B	65B and 280B	33B or 100B

Table 18: Training settings. TP, CP, and EP are tensor, context, and expert parallelism, and MTP is multi-token prediction.

over 4 runs. BrowseComp (Wei et al., 2025) is evaluated on a subset of the benchmark in a custom agent harness, with Exa as the search provider. For N-3 Super, run-to-run variation is about 1 point on SWE-bench Verified and Pro and about 2 points on DeepSWE. We report pass@1 as the mean success rate over rollouts and compute pass@k from the same rollouts.

General-capability benchmarks. The regression checks use the base-model evaluation suite from NVIDIA (2026a). General knowledge averages MMLU (Hendrycks et al., 2021a) (5-shot) and MMLU-Pro (Wang et al., 2024) (5-shot, chain of thought). Math averages GSM8K (Cobbe et al., 2021) (8-shot) and MATH (Hendrycks et al., 2021b) (4-shot). Code averages HumanEval (Chen et al., 2021) (0-shot) and MBPP-Sanitized (Austin et al., 2021) (3-shot). Commonsense understanding averages ARC-Challenge (Clark et al., 2018), HellaSwag (Zellers et al., 2019), OpenBookQA (Mihaylov et al., 2018), PIQA (Bisk et al., 2020), and WinoGrande (Sakaguchi et al., 2020).

C.2 Base-model post-trainability metrics

We track four base-model post-trainability metrics during midtraining.

- **LoCoDiff** (Mentat AI Team (AbanteAI), 2025) gives the model the full git history of a file and asks it to write the file's final state. We use all 198 cases, zero-shot with greedy decoding and a 16K-token generation budget, which covers the longest reference file (11.7K tokens). Exact match is the share of files reproduced byte for byte, and normalized edit distance is capped at 1.2 per case. Greedy decoding still varies slightly across runs because inference is not bit-exact, so we average both over 3 runs.
- **RepoBench** (Liu et al., 2024) measures next-line code completion that requires context from other files in the repository. We report edit similarity in the cross-file-first setting, with greedy decoding.
- **Repository bits-per-byte** follows Composer 2 (Cursor Research, 2026). We concatenate the files of held-out repositories and compute bits-per-byte over long windows. Our version puts the README first, orders files by dependency, keeps the file tree across windows, and does not score file headers.
- **Trace bits-per-byte** is measured on the assistant turns of held-out agent traces. The version in Table 3 scores only the decisive turns of DeepSWE traces, the turns that most affect the outcome, and masks tool-call formatting, which otherwise dominates the score.

D RL setup

The RLVR prompt blend is drawn from 14 verifiable environments (tool use, code generation, SWE and search pivots, instruction following, reasoning, and formatting). We reward-profile a seed blend against both checkpoints and keep only prompts with mixed rewards, meaning that for each checkpoint some but not all of its sampled rollouts succeed, so every prompt is neither trivial nor unsolvable for either model. Training is GRPO with a leave-one-out baseline and no reference-policy KL

penalty, 256 prompts $\times$ 16 generations per step, a 32K-token output cap within a 131K context, a learning rate of 10^{-6} with 10 warmup steps, and 20 optimizer updates. The two runs differ only in their starting checkpoint; they use the same blend, seed, and prompt order. We evaluate SWE-bench Verified with 5 rollouts per task and Terminal-Bench 2.1 with 8, for both the SFT starting points and the RL checkpoints.

Training-time metrics, averaged over matched steps, favor the midtrained model. It earns 4.3% higher train reward (0.308 vs. 0.295) with 12% shorter generations (4.6K vs. 5.3K mean tokens), and it produces format violations (a stray end-of-sequence token or malformed thinking tags, which zero the reward) about ten times less often (0.03% vs. 0.34% of trajectories). Its truncation rate is also lower (0.056% vs. 0.073%). Together these make RL from the midtrained checkpoint about 15% faster per trajectory on the same hardware.

E Additional results

E.1 Context-limit failures

Rollouts that fill the 256K context window count as failures. Under the same harness and settings, they are much more common without midtraining (Table 19), especially on SWE-bench Pro. On SWE-bench Verified, rollouts also hit the 200-turn cap 11 times without midtraining and once after 500B tokens. Midtraining appears to make the agent more economical with its context and turns, not only more accurate.

Benchmark	Rollouts	N-3.5 Lightning-LC	Midtrained, 500B
SWE-bench Verified	1,500	40	26
SWE-bench Multilingual	900	25	14
SWE-bench Pro	2,193	91	13
Terminal-Bench	712	8	1

Table 19: Rollouts that fail by filling the 256K context window, after identical full SFT.

E.2 Long-context results at 1M

To keep evaluation simple, our N-3 Super runs start midtraining from the 1M long-context checkpoint (§3.1). Midtraining then trains at 256K, so RULER at 1M naturally drops, from 79.9 to 33.0 after 25B tokens. A short long-context stage of about 9B tokens, on long-context data only, brings it back to 81.0, above the starting checkpoint, and the 65B N-3 Super checkpoint evaluated in §4.4 includes this short stage, while the 280B checkpoint goes straight to SFT. This drop is a side effect of our evaluation setup. In a mature pipeline, midtraining would come before the final long-context stage, as in §5.2.