OpenThoughts Agent

Data Recipes for Agentic Models

Negin Raoof[*1], Richard Zhuang[*2], Marianna Nezhurina[*3,4,5], Etash Guha[*2], Atula Tejaswi[6], Ryan Marten[7], Charlie F. Ruan[1], Tyler Griggs[1], Alexander Glenn Shaw[8], Hritik Bansal[9], E. Kelly Buchanan[2], Artem Gazizov[10], Reinhard Heckel[11], Chinmay Hegde[12], Sankalp Jajee[13], Daanish Khazi[14], Emmanouil Koukoumidis[15], Xiangyi Li[16], Hange Liu[17], Shlok Natarajan[2], Harsh Raj[18], Nicholas Roberts[19], Ethan Shen[20], Nishad Singhi[21], Michael Siu[22], Ashima Suvarna[9], Hanwen Xing[22], Patrick Yubeaton[12], Robert Zhang[6], Leon Liangyu Chen[2], Xiaokun Chen[2], Steven Dillmann[2], Saadia Gabriel[9], Xunyi Jiang[23], Anurag Kashyap[24], Boxuan Li[25], Yein Park[26], Minh Pham[12], Sujay Sanghavi[6], Lin Shi[27], Ke Sun[17], Yixin Wang[28], Zhiwei Xu[28], Erica Zhang[2], Siyan Zhao[9], Wanjia Zhao[2], Jenia Jitsev[3,4,5], Alex Dimakis[1,7], Benjamin Feuer[†2,15], Ludwig Schmidt[†2]

[1]UC Berkeley, [2]Stanford University, [3]JSC, [4]LAION, [5]Open-Ψ (Open-Sci) Collective, [6]University of Texas at Austin, [7]Bespoke Labs, [8]Laude Institute, [9]UCLA, [10]Harvard University & Harvard Medical School, [11]TU Munich & Munich Center for Machine Learning, [12]New York University, [13]Medical University of South Carolina, [14]The LLM Data Company, [15]Oumi.AI, [16]BenchFlow, [17]Independent Researcher, [18]Northeastern University, [19]University of Wisconsin–Madison, [20]University of Washington, [21]TU Darmstadt, [22]University of Southern California, [23]UC San Diego, [24]Amazon, [25]Microsoft, [26]Korea University, [27]Cornell Tech, [28]University of Michigan

[*]Equal contribution. [†]Equal contribution.

Abstract

Agentic language models dramatically expand the applications of AI yet little is publicly known about how to curate training data for broadly capable agents. Existing open efforts such as SWE-Smith, SERA, and Nemotron-Terminal typically target a single benchmark, leaving open the question of how to train models that generalize across diverse agentic tasks. The OpenThoughts-Agent (OT-Agent) project addresses this gap with a fully open data curation pipeline for training agentic models. We conduct more than 100 controlled ablation experiments to systematically investigate each stage of the pipeline, yielding insights on the importance of task sources and diversity. We then assemble a training set of 100K examples from our pipeline and fine-tune Qwen3-32B on this dataset, which yields an average accuracy of 44.8% across seven agentic benchmarks and a 3.9 percentage point improvement over the strongest existing open data agentic model (Nemotron-Terminal-32B, 40.9%). Moreover, our training data exhibits strong scaling properties, outperforming alternative open datasets at every training set size in compute-controlled comparisons. We publicly release our training sets, data pipeline, experimental data, and models at openthoughts.ai to support future open research on agentic model training.

Preprint.

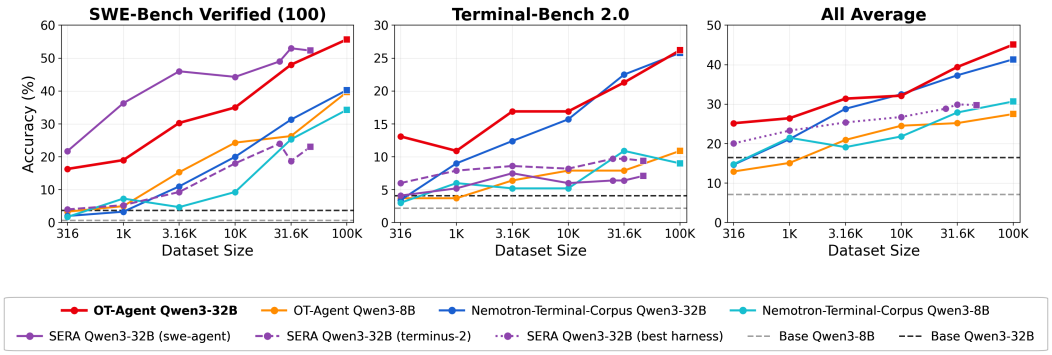

Figure 1: **The OpenThoughts-Agent-SFT dataset leads to SotA performance on Terminal-Bench 2.0 and an 100-subset of SWE-Bench Verified at large dataset scales.** The all-benchmark average is over the seven benchmarks reported in Table 1. Additional notes on SERA below.[1]

1 Introduction

Agentic language models have dramatically expanded the applications of AI. Instead of only answering questions, this new generation of models can perform a wide range of complex tasks that involve using a computer in intricate ways. As a result, AI agents such as Claude Code, Codex, and OpenClaw have rapidly grown in popularity. To enable these agentic applications, the underlying models have improved in their ability to use tools and reason coherently over a long horizon. Further developing these models towards better agents is one of the most important research directions in AI.

At the same time, the public literature contains little information on how to train state-of-the-art agentic models, especially when it comes to training data. The recent DeepSeekV4 release is an illustrative example [10]: the model weights are open and the accompanying paper contains more than 50 pages with details on the architecture and training process, but the paper describes the training data only at a high level with two paragraphs. While there are initial efforts to curate training data for agents in the open such as SWE-Smith [47], SERA [38], Nemotron-Terminal [35], and OpenSWE [11], these efforts usually focus only on one benchmark at a time, e.g., SWE-Bench [23] or Terminal-Bench [28]. Hence, it is currently difficult for open AI research to understand how broadly capable agentic models are trained, and to contribute to their improvement.

We take a step towards open training data for agentic models with the OpenThoughts-Agent (OT-Agent) project. Building on the insights from the prior OpenThoughts work [16], we focus on post-training data for supervised fine-tuning (SFT), now with the goal of improving a model's performance on multiple agentic benchmarks. Our first contribution is a comprehensive data curation pipeline for agentic SFT data on which we conduct more than 100 ablation experiments. Our experiments lead to the following key findings:

- As with reasoning data, the choice of instructions is among the most important factors in our data pipeline.
- The strongest model by benchmark performance does not necessarily make the best teacher.
- Filtering training data to retain the execution traces with more model turns improves the resulting training sets.
- Repeating the top few sources leads to diminishing returns in our largest training runs, and we therefore expand the set of data sources to increase diversity.

Building on our experiments, we assemble a state-of-the-art training set for fine-tuning agentic models. In particular, we fine-tune the Qwen3-32B model on 100k data points from our pipeline and achieve the best performance on a broad suite of agentic tasks compared to other open data models with a Qwen3 or earlier base model at <=32B scale (see Table 1). Our model achieves 54.0% on SWE-Bench Verified and 26.2% on Terminal-Bench 2.0, compared to 41.9% and 25.1% for Nemotron-Terminal-32B. In addition, our model also outperforms prior work on further agentic benchmarks including Aider Polyglot [2], BFCL-Parity [33], GAIA-127 [29], and FinanceAgent-Terminal [6]. Figure 1

[1] SERA uses SWE-agent for data generation and evaluation, so we report both the SWE-agent (solid) and Terminus-2 (dashed) harnesses on SWE-Bench and Terminal-Bench, and the best of the two harnesses per benchmark for the average. SERA's dataset contains at most 47K examples, so its scaling curve stops at 47K.

	Benchmark	OpenThinkerAgent-32B	Nemotron-Terminal-32B	SWE-Lego-Qwen3-32B	SERA-32B	SA-SWE-32B	DeepSWE-Preview	Base Model Qwen3-32B
	Train Size	100K	264K	18K	25K	4.5K	4.5K	N/A
	Method	SFT	SFT	SFT	SFT	RL	RL	N/A
	Average (core and OOD evals)	**44.8**	40.9	34.7	28.1	26.9	26.7	22.8
Core	SWE-Bench-Verified	**54.0**	41.9	51.0	49.4	39.4	42.2	29.1
Core	Terminal-Bench 2.0	**26.2**	**25.1**	16.1	9.7	16.2	4.9	7.5
OOD	Aider-Polyglot	**32.4**	24.9	30.1	26.7	17.3	27.3	28.9
OOD	BFCL-Parity	**85.9**	69.1	81.0	69.1	74.8	77.2	68.3
OOD	MedAgentBench	47.8	**62.6**	36.2	15.6	15.8	8.7	6.8
OOD	GAIA-127	**23.6**	**22.3**	12.9	8.7	11.5	16.5	9.7
OOD	FinanceAgent-Terminal	**44.0**	40.7	15.3	17.3	13.3	10.0	9.3

Table 1: **OpenThinkerAgent-32B is the best open-data model (Qwen-3 model family or earlier, up to 32B scale) on an average of seven agentic benchmarks.** Our model is the best overall on SWE-Bench-Verified and Terminal-Bench 2.0, and also generalizes well on OOD benchmarks we did not use during development. We evaluate each model both in Terminus-2 and in the model's original harness and, for each model, report the maximum accuracy over both harnesses. All models in this table are trained from Qwen3-32B. Bolded cells are within one standard error of the maximum in each row; more detail about standard error can be found in Appendix Table 23.

shows that our training set is not only good because of its size but also exhibits strong scaling trends, outperforming other open datasets in a compute-controlled way for every training set size.

In addition to our SFT investigation, we also study data curation for reinforcement learning (RL). We briefly document the challenges of existing open-data RL curation efforts (such as reproducibility, usability, and scaling) and introduce a newly curated RL dataset. We then validate the efficacy of this data by post-training an 8B model in two stages (SFT + RL) which outperforms our best single-stage 8B model, as well as the strongest existing <=8B baselines, on average across 7 agentic benchmarks (see Table 10 for details).

We publicly release our training sets, data pipeline, experimental data, and models at `https://www.openthoughts.ai/`, so that future open research can build on our artifacts.

2 Related Work

Data curation. Data curation includes data sourcing, data labeling / verification, and sometimes filtration [12, 26]. The advent of large language models and scaling laws progressively made clear the large role data curation plays in AI capabilities, and led to considerable industrial research and rapid advances in the state-of-the-art, as well as a series of public data curation efforts for vision-language models, language models and reasoning models [12, 26, 16, 5]. Rigorous public research on data curation for agents, however, remains scarce, providing a core motivation for this work.

Agents and their benchmarks. The evaluation of large language models has progressed rapidly in the space of the last five years; where pioneering models such as GPT-3 reported primarily on multiple-choice benchmarks such as MMLU, evaluated by probing logprobs for continuation tokens, the community quickly advanced to evaluating the open-ended completions of generative models directly. With the debut of OpenAI's O series of models, specialized thinking formats grew popular and specialized benchmarks such as AIME (static) and LiveBench and LiveCodeBench (dynamic) were developed to challenge thinking models.

Agentic Models. Most recently, the frontier has extended again, with benchmarks such as SWE-Bench and Terminal-Bench assigning scores to discrete tasks such as GitHub issue resolution [28, 23]. Throughout this process, standardized evaluation platforms such as Evalchemy and Harbor have made benchmarks more affordable and reproducible [18, 37]. Over the past year and a half, as agentic AI has entered the research mainstream, various data curation strategies have been proposed for the

Figure 2: **Six-stage SFT data pipeline for OpenThoughts-Agent.** Each stage is ablated independently in Sections 3.1–3.6.

effective post-training of AI agents, spanning both SFT and RL data curation [47, 13, 35, 38, 49, 27, 8]. However, these works have two key limitations; firstly, they almost exclusively focus *either* on SFT *or* on RL, with little attention paid to how these steps intersect; secondly, they tend to focus on a single benchmark or a small cluster of closely related agentic benchmarks. By contrast, our work focuses on generalization across agentic benchmarks, the interaction between SFT and RL, across several popular model scales, and utilizes a range of agentic benchmarks, including OpenThoughts-TB-Lite, new in this work. Recent works use Qwen-3.5 as base model for their experiments [20]. We have decided to keep Qwen-3 as base model to enable consistent comparisons throughout our project. Porting our data improvements to Qwen-3.5 and studying interactions between base model and SFT / RL data are important directions for future work.

Public frameworks for building agents. Although a detailed discussion is beyond the scope of this paper, we wish to emphasize that sound engineering practices constituted a central focus (and a central challenge) of this work, and acknowledge those we used. Many public frameworks for SFT are reasonably mature; we selected a fork of Llama-Factory, extending it to support ALST long-sequence training [50, 4]. Our reinforcement learning framework was an extended version of the popular SkyRL framework; most of the improvements are described in [30]. We used [18] for environment, benchmark and harness management.

3 SFT Data Pipeline

Pipeline This section introduces our experimental pipeline for building the OpenThoughts-Agent dataset. Our goal is to create the best dataset of (task, trajectory) pairs for supervised finetuning coding and terminal agents. The best dataset is the one that produces the highest-performing agent on downstream benchmarks. To do so, we ablate each pipeline step independently, and select the best-performing strategy based on the average z-score across three benchmarks. We compute the z-score of every candidate strategy's accuracy across the stage's full candidate set (subtracting the stage's per-benchmark mean and dividing by its standard deviation), then average the three resulting per-benchmark z-scores. This standardization gives each benchmark equal weight in the ranking despite differing accuracy ranges across the candidate set. Sections 3.1 to 3.6 describe each stage of the pipeline in detail. Unless otherwise specified, trajectories are generated by GLM-4.7-AWQ [41] acting as the teacher in the `terminus-2` harness inside Daytona sandboxes. We conduct our experiments on datasets of size $10{,}000$ since that is small enough to be cost-effective yet large enough to provide a meaningful signal.

Training For each ablation experiment, we utilize the full pipeline to generate $10{,}000$ trajectories for each data strategy, and we finetune Qwen3-8B [45] on each dataset using full-parameter SFT, learning rate $4\text{e}{-5}$ with cosine schedule, global batch size 96, 7 epochs, and $32{,}768$ context length. Each $10{,}000$ finetune takes 160 GPU-hours on GH200s, allowing us to run dozens of pipeline ablations in parallel. These experiments inform the design choices for the final OpenThoughts-Agent pipeline. We include additional details on hyperparameters, training infrastructure, and per-stage SFT settings in the Appendix Section C.

Evaluation Setup We evaluate our models on a set of agent benchmarks that probe long-horizon software-engineering and terminal-use behavior. Our core suite consists of three benchmarks: (1) **OpenThoughts-TBLite** (100 tasks) [32]: a curated collection of 100 Terminal-Bench style tasks that balanced across four difficulty buckets and engineered as a fast proxy for the full Terminal-Bench 2.0 performance, (2) **SWE-Bench Verified-100** (100 tasks) [23]: a stratified subsample (by repository) of the 500-task human-validated SWE-Bench Verified split; an agent produces a patch against a

Python repo at a fixed commit, binary-scored by the upstream test harness, and (3) **Terminal Bench 2** (89 tasks) [28]: hand-crafted, human-verified tasks that spans SWE, biology, security, system administration, and machine learning.

All evaluations run inside isolated Daytona sandboxes [9] using the `terminus-2` [28] agent harness, with $n = 3$ stochastic re-runs per task and standard error reported across trials. To measure generalization, our pipeline experiments exclude a held-out set of out-of-distribution benchmarks, which are only measured once pipeline experimentation is complete. This held-out set consists of Aider Polyglot [2], BFCL [33], MedAgentBench [22], GAIA [29], and FinanceAgent-Terminal [6]. Section G contains further details on evaluation infrastructure and per-benchmark configuration.

3.1 Sourcing Tasks

		Benchmarks			Average	
Rank	Strategy	SWE-bench Verified (100)	OT-TBLite	Terminal-Bench 2.0	Raw	Normalized
1	SWESmith	**32.33** 1.83	17.63 1.63	6.37 1.18	**18.78**	**+1.92**
2	StackExchange SuperUser	13.33 1.41	16.68 1.63	**10.86** 1.35	13.62	+1.51
3	StackExchange Tezos	16.33 1.41	16.94 1.83	9.36 1.06	14.21	+1.45
4	IssueTasks	24.00 1.76	16.44 1.49	6.74 1.24	15.73	+1.37
		...				
95	AgentTuning-OS	0.00 0.00	5.64 1.12	0.37 0.37	2.00	-2.31

Table 2: **Task source choice has the largest spread of any pipeline stage.** Issue-resolution tasks and human-written infrastructure questions dominate the top, but improvements are spiky across benchmarks. Full ranking in Appendix A.1.

Since an SFT agentic dataset comprises task descriptions and agent trajectory pairs, the strategy for generating task descriptions can change downstream accuracy by up to 30 pp on SWE-Bench Verified-100 and 10 pp on Terminal-Bench 2.0(Table 12, ranks 1 vs 95). The design space for task generation strategies is large, encompassing synthetic task generation, human task generation, and more. Moreover, the domain knowledge and skills covered by a task generation strategy can also lead to large differences in downstream performance.

To efficiently find the best task data generation strategies, we ablate a set of 95 task generation strategies covering different initial sources, modes of generation (synthetic vs. not synthetic), and knowledge domains for those tasks. We report the findings in Table 2. The top-performing task generation strategies include synthetic issue-resolution tasks such as SWE-Smith and our own IssueTasks datasets, human-written computer-use questions such as StackExchange SuperUser and Tezos, and other strategies. The effect of choosing different task generation strategies leads to a large difference in downstream evals; for example, TerminalBench 2.0 scores range from 10.9% to 0.4%. We also observe that performance improvements at the top end of data generation strategies are spiky: the top coding-related datasets, such as SWE-Smith, greatly improve SWE-Bench, whereas the human-written infrastructure questions, such as StackExchange SuperUser, improve TerminalBench 2.0.

3.2 Mixing Tasks

Our set of task descriptions will come from a mix of the task generation strategies from Section 3.1. While there is a rich literature on data mixing strategies, we simply ablate the use of the top 1, top 2, and so on datasets in the mix, given the relative ranking of each task generation strategy from Section 3.1. For the Top-N mixing strategy, we take the top N data generation strategies and sample $\frac{10{,}000}{N}$ task descriptions from each.

The results of mixing strategy are in Table 3. Mixing around the Top 4 or Top 8 task generation strategies works best and outperforms the non-mixed Top 1 baseline, since it performs well on all our benchmarks rather than solely on SWE-Bench.

Rank	Mixing Strategy	Benchmarks			Average	
		SWE-Bench Verified (100)	OT-TBLite	Terminal-Bench 2.0	Raw	Normalized
1	Top 4	**29.33** 1.63	**17.00** 1.71	**8.24** 1.24	**18.19**	**+0.49**
2	Top 2	**29.00** 1.60	**18.12** 1.72	**7.12** 1.06	**18.08**	**+0.48**
3	Top 8	28.00 1.70	15.86 1.70	**8.61** 1.24	**17.49**	**+0.19**
			...			
6	Top 1	**30.67** 1.67	14.80 1.40	4.49 0.84	16.65	-0.57

Table 3: **Mixing top-ranked task generation strategies, random shuffle within task.** Mixing the top-4 to top-8 strategies yields the strongest balanced performance, outperforming the unmixed top-1 baseline by avoiding over-specialization.

Rank	Augmentation Strategy	Benchmarks			Average	
		SWE-bench Verified (100)	OT-TBLite	Terminal-Bench 2.0	Raw	Normalized
1	Original (no augmentation)	20.67 1.56	17.22 1.48	**8.99** 1.12	**15.62**	**+0.39**
2	Constrain → Harden	17.33 1.49	17.56 1.59	**9.36** 1.35	**14.75**	**+0.27**
3	Mixed (Original + Hardened)	20.33 1.73	15.39 1.46	**9.74** 1.40	**15.15**	**+0.23**
4	Mixed across sources	15.33 1.33	**19.02** 1.72	**8.24** 1.24	14.20	+0.10
5	Trace hints	**22.67** 1.63	17.17 1.48	5.99 1.12	15.27	-0.13
6	Harden	10.67 1.29	17.73 1.64	**8.61** 1.18	12.34	-0.41
7	Constrain	**24.33** 1.56	13.91 1.43	5.62 1.12	14.62	-0.62

Table 4: **Task description augmentation strategies are within noise.** No LLM-driven augmentation strategy reliably outperforms the un-augmented baseline.

Rank	Filtering Strategy	Benchmarks			Average	
		SWE-bench Verified (100)	OT-TBLite	Terminal-Bench 2.0	Raw	Normalized
1	Response Length (GPT-5, longest)	**22.67** 1.53	**19.51** 1.72	**10.11** 1.35	**17.43**	**+1.38**
2	Response Length (GPT-5, shortest)	**21.33** 1.67	16.91 1.65	**9.74** 1.24	15.99	+0.31
3	AskLLM	**20.67** 1.63	15.87 1.52	**10.86** 1.18	15.80	+0.19
4	Embedding diversity	20.00 1.63	16.41 1.50	7.87 0.99	14.76	-0.72
5	Random (baseline)	19.67 1.70	14.77 1.36	7.87 1.35	14.10	-1.17

Table 5: **Filtering task descriptions with LLM-based difficulty signals improves performance.** Selecting tasks for which GPT-5 produces longer responses improves over random selection by ∼3pp on average.

3.3 Task Augmentation Strategies

After initially generating the task descriptions, a natural hypothesis is that refining them by clarifying their requirements or increasing their difficulty can improve dataset quality. We explore different methods for augmenting task descriptions, such as using an LLM to combine tasks from different sources, add new constraints to each task, harden the task descriptions, and more. The results of these interventions are in Table 4. We find that all interventions fail to improve the baseline of leaving the task description untouched after generation.

3.4 Filtering Tasks

After determining the final mix of task generation strategies, filtering out poor tasks from a set of task descriptions can improve downstream performance. We repeat the ablation of the set of task description filters from OpenThoughts. Table 5 holds the results. We find similar results: LLM task description filters yield the largest improvement among the filters we tested (+3 pp avg, Table 5). Filtering tasks to those that GPT-5 requires more tokens to solve finds tasks that lead to roughly 3 percentage points improvement across all benchmarks.

3.5 Teacher Model

After selecting the set of methods for generating task descriptions, we ablate the design choices for generating the agentic rollouts. Prior work has shown that the choice of teacher can make a significant difference in downstream evals [16]. Building on this awareness, we take the best mix of task descriptions from Section 3.2 and generate a dataset with different teachers that perform well on

Rank	Teacher Model	Benchmarks			Average	
		SWE-bench Verified (100)	OT-TBLite	Terminal-Bench 2.0	Raw	Normalized
1	GLM 4.7 (Quantized)	28.00 1.76	**17.86** 1.60	**8.61** 1.40	**18.16**	**+0.73**
2	Kimi K2.5	**33.33** 1.83	14.19 1.51	**8.24** 1.06	**18.59**	**+0.66**
3	GLM 5	**33.00** 1.60	14.30 1.78	**7.50** 1.18	**18.27**	**+0.66**
4	GLM 4.6 (Quantized)	26.00 1.67	**16.99** 1.80	6.37 1.24	16.45	+0.08
5	GPT-5.3-Codex	21.67 1.33	10.42 1.55	3.75 0.99	11.94	-1.47

Table 6: **Teacher model ablation: stronger model $\neq$ better teacher.** Despite GPT-5.3-Codex being the strongest model on these benchmarks, it is the weakest teacher, underperforming GLM 4.7 AWQ by $\sim$5pp on Terminal-Bench 2.0.

Rank	Trajectory Filter	Benchmarks			Average	
		SWE-bench Verified (100)	OT-TBLite	Terminal-Bench 2.0	Raw	Normalized
1	Min turns ≥ 5	**29.00** 1.56	**19.10** 1.57	**11.61** 1.30	**19.90**	**+1.25**
2	Filter timeouts	26.67 1.70	**18.03** 1.60	**10.49** 1.18	18.39	-0.35
3	Filter subagent traces	23.00 1.63	**17.31** 1.58	**10.86** 1.35	17.06	-0.90

Table 7: **Filtering agent rollouts: keeping longer trajectories helps.** The minimum-turns filter outperforms timeout and subagent filters across all three benchmarks.

TerminalBench 2.0, including GPT-5.3-Codex, Kimi K2.5, GLM-4.6-AWQ, GLM 5, and our baseline GLM-4.7-AWQ. Our results are in Table 6. Despite GPT-5.3-Codex being the best-performing model, it is a worse teacher than GLM-4.7-AWQ, representing a roughly 5% decrease in performance on TerminalBench 2.0. The best teacher is GLM 4.7, despite being older and less performant (e.g., on Terminal Bench) than Kimi K2.5.

3.6 Filtering Agent Rollouts

Filtering out lower-quality agentic rollouts is one way to improve dataset quality. We apply simple heuristic filters, including removing traces that hit a timeout during generation, removing subagent traces, and removing traces with fewer than 5 turns. Our results are in Table 7. We again find that methods that yield longer agentic traces yield better performance, with filtering traces with fewer than 5 turns yielding the largest improvement in downstream evaluation scores. Because longer traces also carry more tokens, we verify in Section A.3 that this gain persists at a matched token budget, confirming it stems from higher-quality multi-turn supervision rather than additional training compute.

4 Scaling Up SFT Data

Scaling dataset size is a powerful method for improving downstream performance. After the pipeline ablations, our final pipeline yields a dataset of 10K datapoints. To increase the dataset size, several simple strategies are possible: (1) using the same task descriptions and generating more agentic rollouts per task; (2) using more task descriptions from the original sources; (3) using synthetic augmentation to create new task descriptions; and (4) using more initial sources.

We start with Method 1 since it is the simplest. The results are in Figure 3. We find that performance plateaus from 31.6K to 100K (+3pp on SWE-Bench Verified-100, $-$2pp on Terminal-Bench 2.0; both within standard error), suggesting that task-description diversity is the bottleneck. While Method 2 would be a simple fix, we are limited by our initial sources; for example, Tezos contains only 997 unique task descriptions.

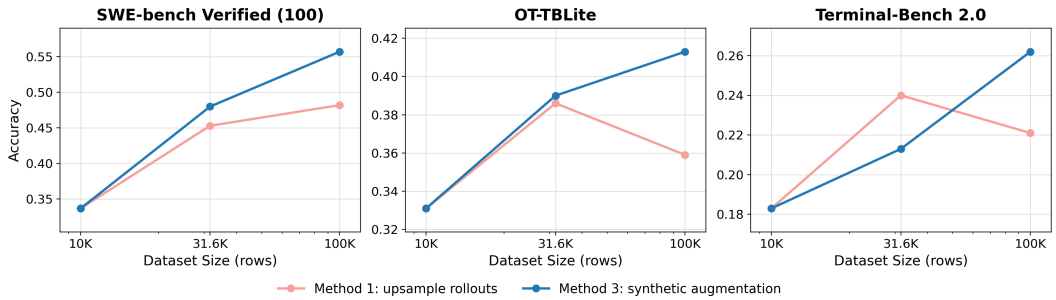

Figure 3: **Synthetic augmentation scales past the upsampling plateau.** Both methods build on the same 10K base and diverge only when scaling beyond it. Method 1 (upsampling additional rollouts per task description) plateaus from 31.6K to 100K, while Method 3 (synthetic task augmentation) continues to improve on all three benchmarks. Error bars are standard error across three stochastic re-runs.

Source Mix	SWE-Bench Verified-100	OT-TBLite	Terminal-Bench 2.0
Top-4	45.33 1.73	36.90 1.82	21.72 1.54
Top-8	**49.00** 1.45	**38.87** 2.09	**22.85** 1.30
Top-16	40.33 1.41	33.14 2.01	20.60 1.67

Table 8: **Task-Source diversity has a negligible effect at larger data scales.** Adding sources beyond Top-4 does not reliably lift performance. All rows are 32B SFT models trained on a 100K-row dataset with the ≥ 5 turns trace filter.

We attempt sourcing from more tasks from Top-4, Top-8, and Top-16 datasets at the 100K scale (Method 4 results in Table 8). Adding more sources beyond Top-4 does *not* reliably help: Top-8 does not significantly outperform Top-4 on every benchmark, while broadening to Top-16 *hurts* on every benchmark. We therefore retain the original Top-4 source mix for the remainder of our experiments and do not pursue Method 4 further.

We turn to synthetic task augmentation (Method 3) and report the results in Figure 3. Concretely, we take the four highest-scoring sources from Section 3.2 (swe-smith, stackexchange-superuser, stackexchange-tezos, and issue-tasks) and replace the Tezos subset with a synthetically augmented version of itself. Tezos contributes the fewest unique task descriptions (only 997 unique tasks). We replace it with a synthetically augmented version of itself: we apply the instruction-rewriting strategies from Section 3.3 to those same 997 base problems, expanding their distinct surface forms from $\sim$902 to over 21K without introducing any new underlying problems. We then use the `gpt-5-nano` response-length signal from Section 3.4 as upsampling *weights* rather than as a hard top-k filter — every unique task receives at least one rollout, with the remaining capacity allocated proportionally to score. Our goal is to preserve full task coverage at every dataset scale. We then apply the ≥ 5-turn trace filter uniformly across all four sources. We see continued performance gains at larger dataset scales, demonstrating that augmentation overcomes the limited task-description-diversity bottleneck observed in Method 1.

OpenThoughts-Agent-v2 At the 100K scale, we observe the best performance for the 32B SFT model, reaching **26.2%** on Terminal-Bench 2.0, **41.3%** on OT-TBLite, and **55.7%** on SWE-Bench Verified-100, showing a monotonic improvement from 31.6K of +7.7pp on SWE-Bench Verified-100 and +5.0pp on Terminal-Bench 2.0. We report the full data generation pipeline for our final 100k-sized dataset, **OpenThoughts-Agent-v2** in Figure 4. We begin from our initial 4 task sources, including synthetic GitHub Issues, human-written Linux tasks, and human-written cryptocurrency questions. We repeat task descriptions and synthetically augment the Tezos questions. We generate agentic rollouts using GLM-4.7-AWQ and filter out traces with fewer than 5 turns.

5 Reinforcement Learning

Many high-profile RL datasets, including SWE-Smith and R2EGym [47, 21], followed a similar approach: select a representative GitHub repository whose state and dependencies can be captured in a Docker container, select or synthesize a flawed commit with failing tests, and generate a natural-

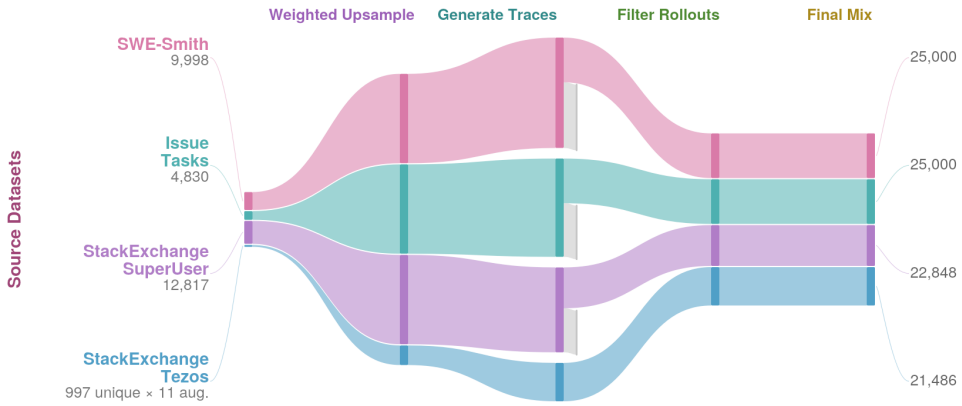

Figure 4: **OpenThoughts-Agent Full Data Pipeline.** Our final SFT dataset is 100k agentic traces.

		Benchmarks			Average	
Rank	RL Data Source	SWE-bench Verified (100)	OT-TBLite	Terminal-Bench 2.0	Raw	Normalized
1	`pymethods2test`	**35.67** 1.83	**16.02** 1.58	**13.48** 1.50	**21.72**	**+1.73**
2	`r2egym`	28.67 1.67	**16.84** 1.64	6.74 1.45	**17.42**	**+0.50**
3	`nemotron-code-oracle`	25.00 1.86	**16.78** 1.67	6.74 1.24	**16.17**	**+0.22**
4	`llm-verifier-freelancer`	22.33 1.53	**14.87** 1.27	8.61 1.35	15.27	-0.24
5	`inferredbugs`	26.00 1.67	14.30 1.55	6.37 0.99	15.56	-0.45
6	`swesmith`	24.33 1.73	14.30 1.51	6.74 1.12	15.12	-0.51
7	`code-contests`	23.67 1.67	13.75 1.55	7.87 1.12	15.10	-0.56
8	`nl2bash`	21.00 1.49	14.51 1.44	6.74 1.12	14.08	-0.70

Table 9: **Data source strongly influences agentic RL performance.** Across eight 8B RL runs that hold the training pipeline fixed and vary only the source, performance varies well beyond noise.

language problem statement describing the issue and requesting the agent turn the failing tests green. We incorporate these efforts into a larger-scale and more systematic series of ablation studies – similar to our work in the SFT domain, we investigate to what extent model performance depends on the *source* of the RL training data. To isolate this, we run a controlled data-source ablation, which we describe below.

5.1 Experimental Details

To control compute consumption, we focus our RL investigations in the 8B regime. We run async RL using the RLOO algorithm [1] with standard binary rewards on verifier success (expect PASS : PASS, expect FAIL : FAIL for all tests). We conduct RL starting from a distilled 8B checkpoint (OT-Agent-ColdSFT) trained on SWE-Smith traces generated by a GLM 4.7 AWQ teacher with thinking, which we also train. Our hero run trained on 24xA100 80GB GPUs with a batch size of 64 and a total wallclock time of $\approx$46 hours. For additional technical details and complete hyperparameters, please refer to Section D.

5.2 Sourcing Tasks in RL

In our ablation, every run uses identical hyperparameters and evaluation criteria, the only thing we vary is the dataset the agent trains on. We compare six sources spanning competitive programming recast as Python contracts (`pymethods2test`), real-repository bug-fixing (`inferredbugs`), competitive-programming environments (`code-contests`), an LLM-filtered Nemotron code-oracle mix (`nemotron-code-oracle`), an LLM-verified freelancer-task set (`llm-verifier-freelancer`), and natural-language-to-Bash tasks (`nl2bash`), as well as SWE-Smith and R2EGym, with the same evaluation conditions described in Section 3. The training logs indicate all sources are learnable to some degree – `inferredbugs` and `nemotron-code-oracle` runs show healthy, monotonically rising mean reward over training (roughly $0.21 \rightarrow 0.46$ and $0.06 \rightarrow 0.36$ across their exported steps, respectively), whereas `code-contests` plateaus at a much lower reward ceiling ($0.06 \rightarrow 0.14$)

	Benchmark	OT-Agent-ColdSFT+RL-8B	OT-Agent-SFT-8B (100K)	Nemotron-Terminal-8B	OT-Agent-SFT-8B (10K)	SWE-Lego-Qwen3-8B	Endless Terminals	Base Model Qwen3-8B
	Base Model							N/A
	Train Size	10K+5K	100K	264K	10K	18K	15K+3K	N/A
	Method	SFT+RL	SFT	SFT	SFT	SFT	SFT+RL	N/A
	Average	**27.9**	27.4	26.0	24.3	18.5	17.4	10.2
Core	SWE-bench Verified	31.9	**38.9**	22.1	22.7	37.6	12.5	13.2
Core	Terminal-Bench 2.0	**13.5**	10.9	13.1	7.9	0.7	8.2	2.2
OOD	Aider Polyglot	18.1	15.9	12.6	14.2	3.7	**18.8**	11.7
OOD	BFCL-Parity	71.5	65.9	59.9	**79.4**	75.3	48.8	34.4
OOD	MedAgentBench	31.1	36.2	**48.6**	25.8	5.0	18.4	3.8
OOD	GAIA-127	6.8	6.6	**14.4**	5.5	6.0	5.2	5.0
OOD	FinanceAgent-Terminal	**22.7**	17.3	11.3	14.7	1.3	10.0	1.3

Table 10: **RL main results (8B scale).** Each cell reports the highest accuracy (%) attained by the model across all evaluated agent harnesses for that benchmark. **Average** is the mean of evaluated cells per column over the seven benchmarks reported here. denotes a model trained from Qwen3-8B. denotes a model trained from OpenThinker-Agent-v1-SFT.

The data source matters. Table 9 shows that source ablation spans a 7.6-point range in raw average accuracy, larger than the 2.0-point run-to-run reproducibility variance (see Table 20), but smaller than the variance derived from SFT source ablation as described in Section 3.

Our strongest source, `pymethods2test`, is a mix of Codeforces / CodeChef / TopCoder style competitive-programming problems that have been re-cast as single-function Python contracts with synthesized docstring-style task descriptions and auto-generated unittest suites. There is no multi-file editing, no repository navigation, no shell-state accumulation across turns. Skills exercised include 1D/2D dynamic programming, string and pattern matching (KMP), combinatorics, matrix/grid construction, ad-hoc puzzles, and formatted table/string generation. Reference solutions average around 20 LOC and task description around 200 words. This dataset is highly reproducible (no potentially stale github references), highly usable (all tasks use the same build environment) and has a clear, and in this case appropriately moderate, difficulty ceiling. The concise but challenging source tasks induce the cold-start model to adopt a consistent problem-solving pattern; during RL, it replaces loops of thinking and exploratory (sed and grep) tool calls with a compact explore, patch, and submit policy. We further analyze the emergent behavioral changes behind the `pymethods2test` result – and why its RL signal pushes the policy to explore more aggressively than the alternatives – in Section F.

Among the alternatives, the synthetic-and-competitive-programming sources (`inferredbugs`, `code-contests`) lead on ID, while the more heterogeneous tool-use sources (`llm-verifier-freelancer`, `nl2bash`) are competitive on OOD despite weaker ID scores. The moderate ID / OOD decoupling suggests that data sources emphasizing single-function code correctness transfer most cleanly to the SWE/terminal Core benchmarks, while broader tool-use data buys OOD generalization at some ID cost; only `pymethods2test` sits at the top of both.

5.3 Results

In Table 10, we show that the complete post-training pipeline (SFT + RL) outperforms the baslines on both core benchmarks, as well as on average across the entire benchmark suite, improving by 18 points, on average, above the Qwen3-8B base model.

"Undertrained" SFT models benefit more from RL, and ultimately outperform pure-distilled models and RL-only models. Consistent with prior work, we find in Table 11 that RL provides the most gains when the SFT model is selected with RL in mind [24]. The Qwen3-8B model, which performs poorly in the `terminus-2` harness on agentic benchmarks, is unable to benefit from agentic RL. The model SFT'd on a smaller amount of data provides the better starting point.

Rank	Model (training recipe)	Benchmarks			Average	
		SWE-bench Verified (100)	OT-TBLite	Terminal-Bench 2.0	Raw	Normalized
1	OT-Agent-ColdSFT+RL-8B	**35.7** 1.8	**16.0** 1.6	**13.5** 1.5	**21.7**	**+1.2**
2	OT-Agent-SFT-8B (10K)	24.3 1.9	15.6 1.5	7.9 1.4	15.9	+0.5
3	OT-Agent-ColdSFT	23.7 1.6	14.8 1.4	6.7 1.1	15.1	+0.4
4	RL on Qwen3-8B (no SFT)	1.0 0.5	7.8 1.1	1.9 0.7	3.6	-0.9
5	Qwen3-8B	5.3 1.1	1.3 0.3	1.5 0.4	2.7	-1.1

Table 11: **RL on top of moderately strong SFT outperforms other strategies.** Our SFT + RL model (1), which starts from a medium-strength SFT base model (3), outperforms both RL from a weaker SFT base model (4) and SFT alone (3). Each cell reports the highest accuracy (%) attained in the Terminus-2 harness. Normalized = mean of per-benchmark z-scores across 5 entries.

6 Conclusion

Agents are increasingly central to science and technology, yet little is publicly known about the data curation techniques used to train them. We address this gap by conducting controlled ablations on a six-stage SFT data curation pipeline, alongside a focused study of agentic RL data. Using the resulting dataset, OpenThoughts-Agent-v2, we finetune Qwen3-32B into OpenThinker-Agent-32B, the strongest open-data <=32B model (Qwen3 family or earlier) on the average of seven agentic benchmarks spanning software engineering, terminal use, tool calling, healthcare, finance, and general assistant tasks. At the 8B scale, combining our SFT data with our `pymethods2test` RL dataset further outperforms the strongest existing <=8B baselines, providing initial evidence that the SFT and RL stages of agentic post-training can be designed to compose. We release the data, pipeline, and models at https://openthoughts.ai to enable broader open research on agentic models.

Limitations include our RL investigation which was conducted only at the 8B scale due to compute constraints; whether the same RL recipe transfers to the 32B regime remains an open question. We also do not ablate the choice of base model: all SFT runs begin from the Qwen3 family, so the contribution of base-model pretraining to the final performance is not isolated. Finally, our largest training set contains 100K trajectories; whether the trends we observe extrapolate to multi-million-trajectory regimes remains untested.

Broader Impacts This work aims to advance open research on agentic models by releasing the dataset, pipeline, and trained models. Open release accelerates scientific progress and lowers the barrier to entry for academic and independent research. However, agentic models are inherently dual-use technologies: the same capabilities that enable beneficial automation can also enable misuse, including unauthorized actions on shared systems. We encourage downstream users to deploy these models with appropriate sandboxing and human oversight.

Acknowledgments and Disclosure of Funding

MN and JJ acknowledge funding by EU Horizon under grant no. 101214398 (ELLIOT) and co-funding by EU from Digital Europe Programme under grant no. 101195233 (openEuroLLM), co-funding from EU under Digital Europe Programme under grant no. 101198470 (LLMs4EU) and from EuroHPC Joint Undertaking programme under grant no. 101182737 (MINERVA), funding by the German Federal Ministry of Research, Technology and Space (BMFTR) under grant no. 01IS24085C (OPENHAFM), under the grant 01IS22094B (WestAI – AI Service Center West), and under the grant 16HPC117K (MINERVA).

BF gratefully acknolwedges resources of the Oak Ridge Leadership Computing Facility (OLCF) and Argonne Leadership Computing Facility (ALCF)] which are a DOE Office of Science User Facility. This work was supported by an award from the ASCR Leadership Computing Challenge (ALCC) under project ERCAP0034861, and the ongoing support of Oumi.AI.

LS gratefully acknowledges the Open Philanthropy Institute for Foundations of Machine Learning (IFML), Apple, and the Microsoft Grant in Customer Experience Innovation for their support on this project.

The entire team wishes to acknowledge the Gauss Centre for Supercomputing e.V. (GCS) for funding this work by providing computing time through the John von Neumann Institute for Computing

(NIC) on the supercomputer JUWELS Booster and JUPITER at Jülich Supercomputing Centre (JSC), EuroHPC Joint Undertaking for computing time and storage on the EuroHPC supercomputer LEONARDO, hosted by CINECA (Bologna, Italy) and the LEONARDO consortium through an EuroHPC AI Factory Science and Innovation grant EHPC-AIF-2025SC04-290 and on EuroHPC supercomputer MareNostrum5 hosted by BSC (Barcelona, Spain) through EuroHPC development access grant EHPC-DEV-2026D01-097, storage resources on JUST granted and operated by JSC and supported by Helmholtz Data Federation (HDF), computing time granted by the JARA and JSC on the supercomputer JURECA at JSC, computing time granted on prototype JEDI via JUREAP (JUPITER Early Access Program) grant at JSC and computing time granted via Gauss AI Competition (reformo) on JUPITER through GCS and German Federal Ministry of Research, Technology and Space (BMFTR). LAION further acknowledges public storage grant by HuggingFace that allows us to provide convenient access to the output of the open-source research to broad community via HF repository. Further thanks go for support provided by supercomputing facilities and their teams, especially to Bjoern Hagemeier and Mathis Bode from Juelich Supercomputer Center (JSC, Germany). This project also benefited from the support of the TACC, NYU Torch, and ZIH Capella supercompute clusters, as well as Modal and Google, and Anyscale for hosting our group meetings.

Finally, we owe a deep debt of gratitude to Daytona.io for providing a robust and scalable container solution for our agentic post-training experiments, and to the Laude Institute for supporting our project with a Slingshots // TWO grant, and the Harbor Framework for allowing us to provide input in the development of the their sandboxing evaluation and optimization environment.

References

[1] Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce-style optimization for learning from human feedback in llms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12248–12267, 2024.

[2] Aider. o1 tops aider's new polyglot leaderboard. https://aider.chat/2024/12/21/polyglot.html, December 2024. Aider blog post. Introduces the Polyglot coding-edit benchmark covering C++, Go, Java, JavaScript, Python, and Rust across 225 exercises. Accessed 2026-05-18.

[3] Alpay Ariyak, Junda Zhang, Junxiong Wang, Shang Zhu, Federico Bianchi, Sanjana Srivastava, Ashwinee Panda, Siddhant Bharti, Chenfeng Xu, John Heo, Xiaoxia Shirley Wu, James Zhou, Percy Liang, Leon Song, Ce Zhang, Ben Athiwaratkun, Zhongzhu Zhou, and Qingyang Wu. Coderforge-preview: Sota open dataset for training efficient agents, February 2026. Project core leads: Alpay Ariyak; Zhongzhu Zhou; Qingyang Wu.

[4] Stas Bekman, Samyam Rajbhandari, Michael Wyatt, Jeff Rasley, Tunji Ruwase, Zhewei Yao, Aurick Qiao, and Yuxiong He. Arctic long sequence training: Scalable and efficient training for multi-million token sequences, 2025.

[5] Akhiad Bercovich, Itay Levy, Izik Golan, Mohammad Dabbah, Ran El-Yaniv, Omri Puny, Ido Galil, Zach Moshe, Tomer Ronen, Najeeb Nabwani, Ido Shahaf, Oren Tropp, Ehud Karpas, Ran Zilberstein, Jiaqi Zeng, Soumye Singhal, Alexander Bukharin, Yian Zhang, Tugrul Konuk, Gerald Shen, Ameya Sunil Mahabaleshwarkar, Bilal Kartal, Yoshi Suhara, Olivier Delalleau, Zijia Chen, Zhilin Wang, David Mosallanezhad, Adi Renduchintala, Haifeng Qian, Dima Rekesh, Fei Jia, Somshubra Majumdar, Vahid Noroozi, Wasi Uddin Ahmad, Sean Narenthiran, Aleksander Ficek, Mehrzad Samadi, Jocelyn Huang, Siddhartha Jain, Igor Gitman, Ivan Moshkov, Wei Du, Shubham Toshniwal, George Armstrong, Branislav Kisacanin, Matvei Novikov, Daria Gitman, Evelina Bakhturina, Prasoon Varshney, Makesh Narsimhan, Jane Polak Scowcroft, John Kamalu, Dan Su, Kezhi Kong, Markus Kliegl, Rabeeh Karimi Mahabadi, Ying Lin, Sanjeev Satheesh, Jupinder Parmar, Pritam Gundecha, Brandon Norick, Joseph Jennings, Shrimai Prabhumoye, Syeda Nahida Akter, Mostofa Patwary, Abhinav Khattar, Deepak Narayanan, Roger Waleffe, Jimmy Zhang, Bor-Yiing Su, Guyue Huang, Terry Kong, Parth Chadha, Sahil Jain, Christine Harvey, Elad Segal, Jining Huang, Sergey Kashirsky, Robert McQueen, Izzy Putterman, George Lam, Arun Venkatesan, Sherry Wu, Vinh Nguyen, Manoj Kilaru, Andrew Wang, Anna Warno, Abhilash Somasamudramath, Sandip Bhaskar, Maka Dong, Nave Assaf, Shahar Mor, Omer Ullman Argov, Scot Junkin, Oleksandr Romanenko, Pedro Larroy, Monika

Katariya, Marco Rovinelli, Viji Balas, Nicholas Edelman, Anahita Bhiwandiwalla, Muthu Subramaniam, Smita Ithape, Karthik Ramamoorthy, Yuting Wu, Suguna Varshini Velury, Omri Almog, Joyjit Daw, Denys Fridman, Erick Galinkin, Michael Evans, Shaona Ghosh, Katherine Luna, Leon Derczynski, Nikki Pope, Eileen Long, Seth Schneider, Guillermo Siman, Tomasz Grzegorzek, Pablo Ribalta, Monika Katariya, Chris Alexiuk, Joey Conway, Trisha Saar, Ann Guan, Krzysztof Pawelec, Shyamala Prayaga, Oleksii Kuchaiev, Boris Ginsburg, Oluwatobi Olabiyi, Kari Briski, Jonathan Cohen, Bryan Catanzaro, Jonah Alben, Yonatan Geifman, and Eric Chung. Llama-nemotron: Efficient reasoning models, 2025.

[6] Antoine Bigeard, Langston Nashold, Rayan Krishnan, and Shirley Wu. Finance agent benchmark: Benchmarking llms on real-world financial research tasks, 2025.

[7] Shiyi Cao, Sumanth Hegde, Dacheng Li, Tyler Griggs, Shu Liu, Eric Tang, Jiayi Pan, Xingyao Wang, Akshay Malik, Graham Neubig, Kourosh Hakhamaneshi, Richard Liaw, Philipp Moritz, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Skyrl-v0: Train real-world long-horizon agents via reinforcement learning, 2025.

[8] Shiyi Cao, Dacheng Li, Fangzhou Zhao, Shuo Yuan, Sumanth R. Hegde, Connor Chen, Charlie Ruan, Tyler Griggs, Shu Liu, Eric Tang, Richard Liaw, Philipp Moritz, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Skyrl-agent: Efficient rl training for multi-turn llm agent, 2025.

[9] Daytona Platforms, Inc. Daytona: Secure and elastic infrastructure for running ai-generated code, 2026.

[10] DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026.

[11] Dayuan Fu, Shenyu Wu, Yunze Wu, Zerui Peng, Yaxing Huang, Jie Sun, Ji Zeng, Mohan Jiang, Lin Zhang, Yukun Li, Jiarui Hu, Liming Liu, Jinlong Hou, and Pengfei Liu. davinci-env: Openswe environment synthesis at scale, 2026.

[12] Samir Yitzhak Gadre, Gabriel Ilharco, Alex Fang, Jonathan Hayase, Georgios Smyrnis, Thao Nguyen, Ryan Marten, Mitchell Wortsman, Dhruba Ghosh, Jieyu Zhang, Eyal Orgad, Rahim Entezari, Giannis Daras, Sarah Pratt, Vivek Ramanujan, Yonatan Bitton, Kalyani Marathe, Stephen Mussmann, Richard Vencu, Mehdi Cherti, Ranjay Krishna, Pang Wei Koh, Olga Saukh, Alexander Ratner, Shuran Song, Hannaneh Hajishirzi, Ali Farhadi, Romain Beaumont, Sewoong Oh, Alexandros G. Dimakis, Jenia Jitsev, Yair Carmon, Vaishaal Shankar, and Ludwig Schmidt. Datacomp: In search of the next generation of multimodal datasets, 2023.

[13] Kanishk Gandhi, Shivam Garg, Noah D. Goodman, and Dimitris Papailiopoulos. Endless terminals: Scaling rl environments for terminal agents, 2026.

[14] GLM-5-Team, :, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, Chenzheng Zhu, Congfeng Yin, Cunxiang Wang, Gengzheng Pan, Hao Zeng, Haoke Zhang, Haoran Wang, Huilong Chen, Jiajie Zhang, Jian Jiao, Jiaqi Guo, Jingsen Wang, Jingzhao Du, Jinzhu Wu, Kedong Wang, Lei Li, Lin Fan, Lucen Zhong, Mingdao Liu, Mingming Zhao, Pengfan Du, Qian Dong, Rui Lu, Shuang-Li, Shulin Cao, Song Liu, Ting Jiang, Xiaodong Chen, Xiaohan Zhang, Xuancheng Huang, Xuezhen Dong, Yabo Xu, Yao Wei, Yifan An, Yilin Niu, Yitong Zhu, Yuanhao Wen, Yukuo Cen, Yushi Bai, Zhongpei Qiao, Zihan Wang, Zikang Wang, Zilin Zhu, Ziqiang Liu, Zixuan Li, Bojie Wang, Bosi Wen, Can Huang, Changpeng Cai, Chao Yu, Chen Li, Chengwei Hu, Chenhui Zhang, Dan Zhang, Daoyan Lin, Dayong Yang, Di Wang, Ding Ai, Erle Zhu, Fangzhou Yi, Feiyu Chen, Guohong Wen, Hailong Sun, Haisha Zhao, Haiyi Hu, Hanchen Zhang, Hanrui Liu, Hanyu Zhang, Hao Peng, Hao Tai, Haobo Zhang, He Liu, Hongwei Wang, Hongxi Yan, Hongyu Ge, Huan Liu, Huanpeng Chu, Jia'ni Zhao, Jiachen Wang, Jiajing Zhao, Jiamin Ren, Jiapeng Wang, Jiaxin Zhang, Jiayi Gui, Jiayue Zhao, Jijie Li, Jing An, Jing Li, Jingwei Yuan, Jinhua Du, Jinxin Liu, Junkai Zhi, Junwen Duan, Kaiyue Zhou, Kangjian Wei, Ke Wang, Keyun Luo, Laiqiang Zhang, Leigang Sha, Liang Xu, Lindong Wu, Lintao Ding, Lu Chen, Minghao Li, Nianyi Lin, Pan Ta, Qiang Zou, Rongjun Song, Ruiqi Yang, Shangqing Tu, Shangtong Yang, Shaoxiang Wu, Shengyan Zhang, Shijie Li, Shuang Li, Shuyi Fan, Wei Qin, Wei Tian, Weining Zhang, Wenbo Yu, Wenjie Liang, Xiang Kuang, Xiangmeng Cheng, Xiangyang Li, Xiaoquan Yan, Xiaowei Hu, Xiaoying Ling, Xing Fan, Xingye Xia, Xinyuan Zhang, Xinze Zhang, Xirui Pan, Xu Zou, Xunkai Zhang, Yadi Liu, Yandong Wu, Yanfu Li, Yidong Wang, Yifan Zhu, Yijun

Tan, Yilin Zhou, Yiming Pan, Ying Zhang, Yinpei Su, Yipeng Geng, Yong Yan, Yonglin Tan, Yuean Bi, Yuhan Shen, Yuhao Yang, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yurong Wu, Yutao Zhang, Yuxi Duan, Yuxuan Zhang, Zezhen Liu, Zhengtao Jiang, Zhenhe Yan, Zheyu Zhang, Zhixiang Wei, Zhuo Chen, Zhuoer Feng, Zijun Yao, Ziwei Chai, Ziyuan Wang, Zuzhou Zhang, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-5: from vibe coding to agentic engineering, 2026.

[15] Tyler Griggs, Sumanth Hegde, Eric Tang, Shu Liu, Shiyi Cao, Dacheng Li, Charlie Ruan, Philipp Moritz, Kourosh Hakhamaneshi, Richard Liaw, Akshay Malik, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Evolving skyrl into a highly-modular rl framework, 2025. Notion Blog.

[16] Etash Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Sprague, Ashima Suvarna, Benjamin Feuer, Liangyu Chen, Zaid Khan, Eric Frankel, Sachin Grover, Caroline Choi, Niklas Muennighoff, Shiye Su, Wanjia Zhao, John Yang, Shreyas Pimpalgaonkar, Kartik Sharma, Charlie Cheng-Jie Ji, Yichuan Deng, Sarah Pratt, Vivek Ramanujan, Jon Saad-Falcon, Jeffrey Li, Achal Dave, Alon Albalak, Kushal Arora, Blake Wulfe, Chinmay Hegde, Greg Durrett, Sewoong Oh, Mohit Bansal, Saadia Gabriel, Aditya Grover, Kai-Wei Chang, Vaishaal Shankar, Aaron Gokaslan, Mike A. Merrill, Tatsunori Hashimoto, Yejin Choi, Jenia Jitsev, Reinhard Heckel, Maheswaran Sathiamoorthy, Alexandros G. Dimakis, and Ludwig Schmidt. Openthoughts: Data recipes for reasoning models, 2025.

[17] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, September 2025.

[18] Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments, January 2026.

[19] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-coder technical report, 2024.

[20] Hamish Ivison, Junjie Oscar Yin, Rulin Shao, Teng Xiao, Nathan Lambert, and Hannaneh Hajishirzi. Tmax: A simple recipe for terminal agents, 2026.

[21] 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.

[22] Yixing Jiang, Kameron C Black, Gloria Geng, Danny Park, James Zou, Andrew Y Ng, and Jonathan H Chen. Medagentbench: A virtual ehr environment to benchmark medical llm agents. *NEJM AI*, page AIdbp2500144, 2025.

[23] 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?, 2024.

[24] Feiyang Kang, Michael Kuchnik, Karthik Padthe, Marin Vlastelica, Ruoxi Jia, Carole-Jean Wu, and Newsha Ardalani. Quagmires in sft-rl post-training: When high sft scores mislead and what to use instead, 2025.

[25] 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.

[26] Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, Saurabh Garg, Rui Xin, Niklas Muennighoff, Reinhard Heckel, Jean Mercat, Mayee Chen, Suchin Gururangan, Mitchell Wortsman, Alon Albalak, Yonatan Bitton, Marianna Nezhurina, Amro Abbas, Cheng-Yu Hsieh, Dhruba Ghosh, Josh Gardner, Maciej Kilian, Hanlin Zhang, Rulin Shao, Sarah Pratt, Sunny Sanyal, Gabriel Ilharco, Giannis Daras, Kalyani Marathe, Aaron Gokaslan, Jieyu Zhang, Khyathi Chandu, Thao Nguyen, Igor Vasiljevic, Sham Kakade, Shuran Song, Sujay Sanghavi, Fartash Faghri, Sewoong Oh, Luke Zettlemoyer, Kyle Lo, Alaaeldin El-Nouby, Hadi Pouransari, Alexander Toshev, Stephanie Wang, Dirk Groeneveld, Luca Soldaini, Pang Wei Koh, Jenia Jitsev, Thomas Kollar, Alexandros G. Dimakis, Yair Carmon, Achal Dave, Ludwig Schmidt, and Vaishaal Shankar. Datacomp-lm: In search of the next generation of training sets for language models, 2025.

[27] Michael Luo, Naman Jain, Jaskirat Singh, Sijun Tan, Ameen Patel, Qingyang Wu, Alpay Ariyak, Colin Cai, Tarun Venkat, Shang Zhu, Ben Athiwaratkun, Manan Roongta, Ce Zhang, Li Erran Li, Raluca Ada Popa, Koushik Sen, and Ion Stoica. Deepswe: Training a state-of-the-art coding agent from scratch by scaling rl. https://www.together.ai/blog/deepswe, 2025. Notion Blog.

[28] 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, Alexandros G. Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces, 2026.

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

[30] NovaSky AI Team. Train your terminal-use agent with SkyRL + Harbor. https://novasky-ai.notion.site/skyrl-harbor, February 2026. Blog post, UC Berkeley Sky Computing Lab in collaboration with Anyscale and Laude Institute.

[31] NVIDIA, :, Aaron Blakeman, Aaron Grattafiori, Aarti Basant, Abhibha Gupta, Abhinav Khattar, Adi Renduchintala, Aditya Vavre, Akanksha Shukla, Akhiad Bercovich, Aleksander Ficek, Aleksandr Shaposhnikov, Alex Kondratenko, Alexander Bukharin, Alexandre Milesi, Ali Taghibakhshi, Alisa Liu, Amelia Barton, Ameya Sunil Mahabaleshwarkar, Amir Klein, Amit Zuker, Amnon Geifman, Amy Shen, Anahita Bhiwandiwalla, Andrew Tao, Ann Guan, Anubhav Mandarwal, Arham Mehta, Ashwath Aithal, Ashwin Poojary, Asif Ahamed, Asma Kuriparambil Thekkumpate, Ayush Dattagupta, Banghua Zhu, Bardiya Sadeghi, Barnaby Simkin, Ben Lanir, Benedikt Schifferer, Besmira Nushi, Bilal Kartal, Bita Darvish Rouhani, Boris Ginsburg, Brandon Norick, Brandon Soubasis, Branislav Kisacanin, Brian Yu, Bryan Catanzaro, Carlo del Mundo, Chantal Hwang, Charles Wang, Cheng-Ping Hsieh, Chenghao Zhang, Chenhan Yu, Chetan Mungekar, Chintan Patel, Chris Alexiuk, Christopher Parisien, Collin Neale, Damon Mosk-Aoyama, Dan Su, Dane Corneil, Daniel Afrimi, Daniel Rohrer, Daniel Serebrenik, Daria Gitman, Daria Levy, Darko Stosic, David Mosallanezhad, Deepak Narayanan, Dhruv Nathawani, Dima Rekesh, Dina Yared, Divyanshu Kakwani, Dong Ahn, Duncan Riach, Dusan Stosic, Edgar Minasyan, Edward Lin, Eileen Long, Eileen Peters Long, Elena Lantz, Ellie Evans, Elliott Ning, Eric Chung, Eric Harper, Eric Tramel, Erick Galinkin, Erik Pounds, Evan Briones, Evelina Bakhturina, Faisal Ladhak, Fay Wang, Fei Jia, Felipe Soares, Feng Chen, Ferenc Galko, Frankie Siino, Gal Hubara Agam, Ganesh Ajjanagadde, Gantavya Bhatt, Gargi Prasad, George Armstrong, Gerald Shen, Gorkem Batmaz, Grigor Nalbandyan, Haifeng Qian, Harsh Sharma, Hayley Ross, Helen Ngo, Herman Sahota, Hexin Wang, Himanshu Soni, Hiren Upadhyay, Huizi Mao, Huy C Nguyen, Huy Q Nguyen, Iain Cunningham, Ido Shahaf, Igor Gitman, Ilya Loshchilov, Ivan Moshkov, Izzy Putterman, Jan Kautz, Jane Polak Scowcroft, Jared Casper, Jatin Mitra, Jeffrey Glick, Jenny Chen, Jesse Oliver, Jian Zhang, Jiaqi Zeng, Jie Lou, Jimmy Zhang, Jining Huang, Joey Conway, Joey Guman, John Kamalu, Johnny Greco, Jonathan Cohen, Joseph Jennings, Joyjit Daw, Julien Veron Vialard, Junkeun Yi, Jupinder Parmar, Kai Xu, Kan Zhu, Kari Briski, Katherine Cheung, Katherine Luna, Keshav Santhanam, Kevin Shih, Kezhi Kong, Khushi Bhardwaj, Krishna C. Puvvada, Krzysztof Pawelec, Kumar Anik, Lawrence McAfee, Laya Sleiman, Leon Derczynski, Li Ding, Lucas Liebenwein, Luis Vega, Maanu Grover, Maarten Van Segbroeck, Maer Rodrigues de Melo, Makesh Narsimhan Sreedhar, Manoj Kilaru, Maor Ashkenazi, Marc Romeijn, Mark Cai, Markus Kliegl, Maryam Moosaei, Matvei Novikov, Mehrzad Samadi, Melissa Corpuz, Mengru Wang, Meredith Price, Michael Boone, Michael Evans, Miguel Martinez, Mike Chrzanowski, Mohammad Shoeybi, Mostofa Patwary, Nabin Mulepati, Natalie Hereth, Nave Assaf, Negar Habibi, Neta Zmora, Netanel Haber, Nicola Sessions, Nidhi Bhatia, Nikhil Jukar, Nikki Pope, Nikolai Ludwig, Nima Tajbakhsh, Nirmal Juluru, Oleksii Hrinchuk, Oleksii Kuchaiev, Olivier Delalleau, Oluwatobi Olabiyi, Omer Ullman Argov, Ouye Xie, Parth Chadha, Pasha Shamis, Pavlo Molchanov, Pawel Morkisz, Peter Dykas, Peter Jin, Pinky Xu, Piotr Januszewski, Pranav Prashant Thombre, Prasoon Varshney, Pritam Gundecha, Qing Miao, Rabeeh Karimi Mahabadi, Ran El-Yaniv, Ran Zilberstein, Rasoul Shafipour, Rich Harang, Rick Izzo, Rima Shahbazyan, Rishabh Garg, Ritika Borkar, Ritu Gala, Riyad Islam, Roger Waleffe, Rohit Watve, Roi Koren, Ruoxi Zhang, Russell J. Hewett, Ryan Prenger, Ryan Timbrook, Sadegh Mahdavi, Sahil Modi, Samuel Kriman, Sanjay Kariyappa, Sanjeev Satheesh, Saori Kaji, Satish Pasumarthi, Sean Narentharen, Sean Narenthiran, Seonmyeong Bak, Sergey Kashirsky, Seth Poulos, Shahar Mor, Shanmugam Ramasamy, Shantanu Acharya, Shaona Ghosh, Sharath Turuvekere Sreenivas, Shelby Thomas, Shiqing Fan, Shreya Gopal, Shrimai Prabhumoye, Shubham Pachori, Shubham Toshniwal, Shuoyang Ding, Siddharth Singh, Simeng Sun, Smita Ithape, Somshubra Majumdar, Soumye Singhal, Stefania Alborghetti, Stephen Ge, Sugam Dipak Devare, Sumeet Kumar Barua, Suseella Panguluri, Suyog Gupta, Sweta Priyadarshi, Syeda Nahida Akter, Tan Bui, Teodor-Dumitru Ene, Terry Kong, Thanh Do, Tijmen Blankevoort, Tom Balough, Tomer Asida, Tomer Bar Natan, Tugrul Konuk, Twinkle Vashishth, Udi Karpas, Ushnish De, Vahid Noorozi, Vahid Noroozi, Venkat Srinivasan, Venmugil Elango, Vijay Korthikanti, Vitaly Kurin, Vitaly Lavrukhin, Wanli Jiang, Wasi Uddin Ahmad, Wei Du, Wei Ping, Wenfei Zhou, Will Jennings, William Zhang, Wojciech Prazuch, Xiaowei Ren, Yashaswi Karnati, Yejin Choi, Yev Meyer, Yi-Fu Wu, Yian Zhang, Ying Lin, Yonatan Geifman, Yonggan Fu, Yoshi Subara, Yoshi Suhara, Yubo Gao, Zach Moshe, Zhen Dong, Zihan Liu, Zijia Chen, and Zijie Yan. Nemotron 3 nano: Open, efficient

mixture-of-experts hybrid mamba-transformer model for agentic reasoning, 2025.

[32] OpenThoughts-Agent team, Snorkel AI, and Bespoke Labs. OpenThoughts-TBLite: A High-Signal Benchmark for Iterating on Terminal Agents. https://www.openthoughts.ai/blog/openthoughts-tblite, February 2026.

[33] Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning (ICML)*, 2025.

[34] Xiaoxuan Peng, Xinyu Lu, Kaiqi Zhang, Taosong Fang, Boxi Cao, and Yaojie Lu. Litecoder: Advancing small and medium-sized code agents, 2026.

[35] Renjie Pi, Grace Lam, Mohammad Shoeybi, Pooya Jannaty, Bryan Catanzaro, and Wei Ping. On data engineering for scaling llm terminal capabilities, 2026.

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

[37] Negin Raoof, Etash Kumar Guha, Ryan Marten, Jean Mercat, Eric Frankel, Sedrick Keh, Hritik Bansal, Georgios Smyrnis, Marianna Nezhurina, Trung Vu, Zayne Rea Sprague, Mike A Merrill, Liangyu Chen, Caroline Choi, Zaid Khan, Sachin Grover, Benjamin Feuer, Ashima Suvarna, Shiye Su, Wanjia Zhao, Kartik Sharma, Charlie Cheng-Jie Ji, Kushal Arora, Jeffrey Li, Aaron Gokaslan, Sarah M Pratt, Niklas Muennighoff, Jon Saad-Falcon, John Yang, Asad Aali, Shreyas Pimpalgaonkar, Alon Albalak, Achal Dave, Hadi Pouransari, Greg Durrett, Sewoong Oh, Tatsunori Hashimoto, Vaishaal Shankar, Yejin Choi, Mohit Bansal, Chinmay Hegde, Reinhard Heckel, Jenia Jitsev, Maheswaran Sathiamoorthy, Alexandros Dimakis, and Ludwig Schmidt. Evalchemy, 2025.

[38] Ethan Shen, Danny Tormoen, Saurabh Shah, Ali Farhadi, and Tim Dettmers. Sera: Soft-verified efficient repository agents, 2026.

[39] Qijia Shen, Jay Rainton, Aznaur Aliev, Ahmed Awelkair, Boyuan Ma, Zhiqi (Julie) Huang, Yuzhen Mao, Wendong Fan, Philip Torr, Bernard Ghanem, Changran Hu, Urmish Thakker, and Guohao Li. SETA: Scaling Environments for Terminal Agents, January 2026. Blog: https://eigent-ai.notion.site/SETA-Scaling-Environments-for-Terminal-Agents-2d2511c70ba280a9b7c0fe3e7f1b6ab8.

[40] Chaofan Tao, Jierun Chen, Yuxin Jiang, Kaiqi Kou, Shaowei Wang, Ruoyu Wang, Xiaohui Li, Sidi Yang, Yiming Du, Jianbo Dai, Zhiming Mao, Xinyu Wang, Lifeng Shang, and Haoli Bai. Swe-lego: Pushing the limits of supervised fine-tuning for software issue resolving, 2026.

[41] GLM Team, Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, Yean Cheng, Yifan An, Yilin Niu, Yuanhao Wen, Yushi Bai, Zhengxiao Du, Zihan Wang, Zilin Zhu, Bohan Zhang, Bosi Wen, Bowen Wu, Bowen Xu, Can Huang, Casey Zhao, Changpeng Cai, Chao Yu, Chen Li, Chendi Ge, Chenghua Huang, Chenhui Zhang, Chenxi Xu, Chenzheng Zhu, Chuang Li, Congfeng Yin, Daoyan Lin, Dayong Yang, Dazhi Jiang, Ding Ai, Erle Zhu, Fei Wang, Gengzheng Pan, Guo Wang, Hailong Sun, Haitao Li, Haiyang Li, Haiyi Hu, Hanyu Zhang, Hao Peng, Hao Tai, Haoke Zhang, Haoran Wang, Haoyu Yang, He Liu, He Zhao, Hongwei Liu, Hongxi Yan, Huan Liu, Huilong Chen, Ji Li, Jiajing Zhao, Jiamin Ren, Jian Jiao, Jiani Zhao, Jianyang Yan, Jiaqi Wang, Jiayi Gui, Jiayue Zhao, Jie Liu, Jijie Li, Jing Li, Jing Lu, Jingsen Wang, Jingwei Yuan, Jingxuan Li, Jingzhao Du, Jinhua Du, Jinxin Liu, Junkai Zhi, Junli Gao, Ke Wang, Lekang Yang, Liang Xu, Lin Fan, Lindong Wu, Lintao Ding, Lu Wang, Man Zhang, Minghao Li, Minghuan Xu, Mingming Zhao, Mingshu Zhai, Pengfan Du, Qian Dong, Shangde Lei, Shangqing Tu, Shangtong Yang, Shaoyou Lu, Shijie Li, Shuang Li, Shuang-Li, Shuxun Yang, Sibo Yi, Tianshu Yu, Wei Tian, Weihan Wang, Wenbo Yu, Weng Lam Tam, Wenjie Liang, Wentao Liu, Xiao Wang, Xiaohan Jia, Xiaotao Gu, Xiaoying Ling, Xin Wang, Xing Fan, Xingru Pan, Xinyuan Zhang, Xinze Zhang, Xiuqing Fu, Xunkai Zhang, Yabo Xu, Yandong Wu, Yida Lu, Yidong Wang, Yilin Zhou, Yiming Pan, Ying Zhang, Yingli Wang, Yingru Li, Yinpei Su, Yipeng Geng, Yitong Zhu, Yongkun Yang, Yuhang Li, Yuhao Wu, Yujiang Li, Yunan Liu, Yunqing

Wang, Yuntao Li, Yuxuan Zhang, Zezhen Liu, Zhen Yang, Zhengda Zhou, Zhongpei Qiao, Zhuoer Feng, Zhuorui Liu, Zichen Zhang, Zihan Wang, Zijun Yao, Zikang Wang, Ziqiang Liu, Ziwei Chai, Zixuan Li, Zuodong Zhao, Wenguang Chen, Jidong Zhai, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models, 2025.

[42] Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, S. H. Cai, Yuan Cao, Y. Charles, H. S. Che, Cheng Chen, Guanduo Chen, Huarong Chen, Jia Chen, Jiahao Chen, Jianlong Chen, Jun Chen, Kefan Chen, Liang Chen, Ruijue Chen, Xinhao Chen, Yanru Chen, Yanxu Chen, Yicun Chen, Yimin Chen, Yingjiang Chen, Yuankun Chen, Yujie Chen, Yutian Chen, Zhirong Chen, Ziwei Chen, Dazhi Cheng, Minghan Chu, Jialei Cui, Jiaqi Deng, Muxi Diao, Hao Ding, Mengfan Dong, Mengnan Dong, Yuxin Dong, Yuhao Dong, Angang Du, Chenzhuang Du, Dikang Du, Lingxiao Du, Yulun Du, Yu Fan, Shengjun Fang, Qiulin Feng, Yichen Feng, Garimugai Fu, Kelin Fu, Hongcheng Gao, Tong Gao, Yuyao Ge, Shangyi Geng, Chengyang Gong, Xiaochen Gong, Zhuoma Gongque, Qizheng Gu, Xinran Gu, Yicheng Gu, Longyu Guan, Yuanying Guo, Xiaoru Hao, Weiran He, Wenyang He, Yunjia He, Chao Hong, Hao Hu, Jiaxi Hu, Yangyang Hu, Zhenxing Hu, Ke Huang, Ruiyuan Huang, Weixiao Huang, Zhiqi Huang, Tao Jiang, Zhejun Jiang, Xinyi Jin, Yu Jing, Guokun Lai, Aidi Li, C. Li, Cheng Li, Fang Li, Guanghe Li, Guanyu Li, Haitao Li, Haoyang Li, Jia Li, Jingwei Li, Junxiong Li, Lincan Li, Mo Li, Weihong Li, Wentao Li, Xinhang Li, Xinhao Li, Yang Li, Yanhao Li, Yiwei Li, Yuxiao Li, Zhaowei Li, Zheming Li, Weilong Liao, Jiawei Lin, Xiaohan Lin, Zhishan Lin, Zichao Lin, Cheng Liu, Chenyu Liu, Hongzhang Liu, Liang Liu, Shaowei Liu, Shudong Liu, Shuran Liu, Tianwei Liu, Tianyu Liu, Weizhou Liu, Xiangyan Liu, Yangyang Liu, Yanming Liu, Yibo Liu, Yuanxin Liu, Yue Liu, Zhengying Liu, Zhongnuo Liu, Enzhe Lu, Haoyu Lu, Zhiyuan Lu, Junyu Luo, Tongxu Luo, Yashuo Luo, Long Ma, Yingwei Ma, Shaoguang Mao, Yuan Mei, Xin Men, Fanqing Meng, Zhiyong Meng, Yibo Miao, Minqing Ni, Kun Ouyang, Siyuan Pan, Bo Pang, Yuchao Qian, Ruoyu Qin, Zeyu Qin, Jiezhong Qiu, Bowen Qu, Zeyu Shang, Youbo Shao, Tianxiao Shen, Zhennan Shen, Juanfeng Shi, Lidong Shi, Shengyuan Shi, Feifan Song, Pengwei Song, Tianhui Song, Xiaoxi Song, Hongjin Su, Jianlin Su, Zhaochen Su, Lin Sui, Jinsong Sun, Junyao Sun, Tongyu Sun, Flood Sung, Yunpeng Tai, Chuning Tang, Heyi Tang, Xiaojuan Tang, Zhengyang Tang, Jiawen Tao, Shiyuan Teng, Chaoran Tian, Pengfei Tian, Ao Wang, Bowen Wang, Chensi Wang, Chuang Wang, Congcong Wang, Dingkun Wang, Dinglu Wang, Dongliang Wang, Feng Wang, Hailong Wang, Haiming Wang, Hengzhi Wang, Huaqing Wang, Hui Wang, Jiahao Wang, Jinhong Wang, Jiuzheng Wang, Kaixin Wang, Linian Wang, Qibin Wang, Shengjie Wang, Shuyi Wang, Si Wang, Wei Wang, Xiaochen Wang, Xinyuan Wang, Yao Wang, Yejie Wang, Yipu Wang, Yiqin Wang, Yucheng Wang, Yuzhi Wang, Zhaoji Wang, Zhaowei Wang, Zhengtao Wang, Zhexu Wang, Zihan Wang, Zizhe Wang, Chu Wei, Ming Wei, Chuan Wen, Zichen Wen, Chengjie Wu, Haoning Wu, Junyan Wu, Rucong Wu, Wenhao Wu, Yuefeng Wu, Yuhao Wu, Yuxin Wu, Zijian Wu, Chenjun Xiao, Jin Xie, Xiaotong Xie, Yuchong Xie, Yifei Xin, Bowei Xing, Boyu Xu, Jianfan Xu, Jing Xu, Jinjing Xu, L. H. Xu, Lin Xu, Suting Xu, Weixin Xu, Xinbo Xu, Xinran Xu, Yangchuan Xu, Yichang Xu, Yuemeng Xu, Zelai Xu, Ziyao Xu, Junjie Yan, Yuzi Yan, Guangyao Yang, Hao Yang, Junwei Yang, Kai Yang, Ningyuan Yang, Ruihan Yang, Xiaofei Yang, Xinlong Yang, Ying Yang, Yi Yang, Yi Yang, Zhen Yang, Zhilin Yang, Zonghan Yang, Haotian Yao, Dan Ye, Wenjie Ye, Zhuorui Ye, Bohong Yin, Chengzhen Yu, Longhui Yu, Tao Yu, Tianxiang Yu, Enming Yuan, Mengjie Yuan, Xiaokun Yuan, Yang Yue, Weihao Zeng, Dunyuan Zha, Haobing Zhan, Dehao Zhang, Hao Zhang, Jin Zhang, Puqi Zhang, Qiao Zhang, Rui Zhang, Xiaobin Zhang, Y. Zhang, Yadong Zhang, Yangkun Zhang, Yichi Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yushun Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Chenguang Zhao, Feifan Zhao, Jinxiang Zhao, Shuai Zhao, Xiangyu Zhao, Yikai Zhao, Zijia Zhao, Huabin Zheng, Ruihan Zheng, Shaojie Zheng, Tengyang Zheng, Junfeng Zhong, Longguang Zhong, Weiming Zhong, M. Zhou, Runjie Zhou, Xinyu Zhou, Zaida Zhou, Jinguo Zhu, Liya Zhu, Xinhao Zhu, Yuxuan Zhu, Zhen Zhu, Jingze Zhuang, Weiyu Zhuang, Ying Zou, and Xinxing Zu. Kimi k2.5: Visual agentic intelligence, 2026.

[43] The Terminal-Bench Team. Terminal-bench 2.1. https://www.tbench.ai/news/terminal-bench-2-1, May 2026.

[44] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin,

Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2025.

[45] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025.

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

[47] John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents, 2025.

[48] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[49] Ji Zeng, Dayuan Fu, Tiantian Mi, Yumin Zhuang, Yaxing Huang, Xuefeng Li, Lyumanshan Ye, Muhang Xie, Qishuo Hua, Zhen Huang, Mohan Jiang, Hanning Wang, Jifan Lin, Yang Xiao, Jie Sun, Yunze Wu, and Pengfei Liu. davinci-dev: Agent-native mid-training for software engineering, 2026.

[50] Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. LlamaFactory: Unified efficient fine-tuning of 100+ language models. In Yixin Cao, Yang Feng, and Deyi Xiong, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 400–410, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

A Full SFT Pipeline Ablation Tables **21**
A.1 Task Generation Strategies (Full Ranking) 21
A.2 Mixing Strategies (Full Results) 23
A.3 Filtering for Longer Episodes: A Compute-Controlled Ablation 24

B Scaling at 8B **24**

C SFT Hyperparameters **25**
C.1 SFT training hyperparameters per data scale 25

D RL Hyperparameters and Infrastructure **26**
D.1 Hardware 26
D.2 Algorithm and optimizer 27
D.3 Training schedule 27
D.4 Distributed strategy 27
D.5 Generation (vLLM) 27
D.6 Rollout environment (Harbor / terminus-2) 28
D.7 Headline metrics at step 48 28
D.8 Reproducibility Artifacts 28

E RL Run-to-Run Reproducibility **28**

F What RL Learns: Emergent Behavior Behind the `pymethods2test` Result **29**
F.1 Legitimate exploration, not reward hacking 29
F.2 The contrast: the LLM-verifier run compacts instead of explores 30
F.3 Connecting behavior to downstream evals 31

G Eval Setup **31**

A Full SFT Pipeline Ablation Tables

This appendix contains the full ablation tables for the SFT pipeline experiments summarized in Section 3. All experiments follow the setup described in Section 3: 10,000 trajectories per dataset, Qwen3-8B fine-tuned with full-parameter SFT, and three stochastic re-runs per benchmark with standard error reported as a subscript.

A.1 Task Generation Strategies (Full Ranking)

Table 12 and Table 13 report the full ranking of all 95 task generation strategies summarized in Table 2. Strategies are sorted by normalized average z-score across the three benchmarks.

Rank	Strategy	Benchmarks (%)			Average	
		SWE-Bench Verified (100)	OT-TBLite	Terminal-Bench 2.0	Raw	Normalized
1	swe-smith	32.33	17.63	6.37	18.78	+1.92
2	stackexchange-superuser	13.33	16.68	10.86	13.62	+1.51
3	stackexchange-tezos	16.33	16.94	9.36	14.21	+1.45
4	issue-tasks	24.00	16.44	6.74	15.73	+1.37
5	repo-scaffold	11.00	20.68	8.24	13.31	+1.31
6	r2egym	28.33	16.57	4.12	16.34	+1.17
7	stackexchange-tor	15.33	16.11	8.61	13.35	+1.16
8	swegym	27.00	17.09	4.12	16.07	+1.14
9	code-feedback	11.00	17.39	8.61	12.33	+1.04
10	stackexchange-unix	16.00	18.90	5.99	13.63	+1.02
11	taskmaster2	10.00	16.44	8.99	11.81	+0.95
12	staqc	11.00	19.41	6.74	12.38	+0.91
13	multifile-composition	9.67	20.61	5.99	12.09	+0.82
14	stack-pytest-withtests	18.33	18.43	3.75	13.50	+0.70
15	ghactions	8.00	17.58	7.12	10.90	+0.61
16	stack-selfdoc-gpt5mini	11.00	17.79	5.99	11.59	+0.61
17	self-instruct-naive	10.00	14.70	7.87	10.86	+0.57
18	stack-ruby	9.67	16.04	7.12	10.94	+0.55
19	nemotron-junit	7.33	15.90	7.87	10.37	+0.54
20	synatra	7.00	14.11	8.99	10.03	+0.54
21	stack-selfdoc	14.33	15.38	5.62	11.78	+0.49
22	manybugs	18.33	12.37	5.99	12.23	+0.48
23	pr-mining	11.00	16.58	5.99	11.19	+0.48
24	nebius-swe-agent	19.33	12.85	5.24	12.47	+0.45
25	stack-pytest	11.00	16.70	5.62	11.11	+0.43
26	stack-bash-withtests	10.67	16.78	5.62	11.02	+0.42
27	go-browse-wa	12.00	15.97	5.62	11.20	+0.41
28	exercism-python	7.33	12.52	8.99	9.61	+0.39
29	softwareheritage	9.00	16.82	5.99	10.60	+0.39
30	stack-cpp	11.00	17.04	5.24	11.09	+0.39
31	stack-junit	8.67	14.91	7.12	10.23	+0.38
32	github-dockerfiles	9.33	12.41	8.24	9.99	+0.36
33	stack-rust	9.33	19.68	4.12	11.04	+0.36
34	freelancer	8.67	17.07	5.62	10.45	+0.33
35	swegym-openhands	20.00	13.16	4.12	12.43	+0.32
36	nl2bash	2.00	12.88	7.12	10.00	+0.31
37	stack-pytest-gpt5mini	8.00	16.37	5.99	10.12	+0.28
38	stack-dockerfile	8.00	17.56	5.24	10.27	+0.27
39	stackexchange-overflow	9.67	15.89	5.62	10.39	+0.27
40	bugsinpy	12.33	15.44	4.87	10.88	+0.24
41	stack-rspec	8.00	17.72	4.87	10.20	+0.22
42	stackexchange-codereview	11.00	14.50	5.62	10.37	+0.20
43	glaive-code-assistant	10.33	16.81	4.49	10.54	+0.19
44	stack-pytest-synthetic-gpt5nano	10.67	16.53	4.49	10.56	+0.19
45	mind2web	8.67	15.32	5.62	9.87	+0.15
46	stack-jest	9.33	14.99	5.62	9.98	+0.15
47	stack-bash	11.33	12.39	6.37	10.03	+0.14
48	bash-textbook	6.67	16.72	5.24	9.54	+0.11
49	stack-phpunit	7.67	16.76	4.87	9.77	+0.10
50	stack-csharp	8.67	16.12	4.87	9.89	+0.09
51	nemotron-bash-withtests-gpt5mini	9.33	14.52	5.24	9.70	+0.04
52	nemotron-rspec	9.00	15.35	4.87	9.74	+0.03
53	nnetnav-live	11.33	13.35	5.24	9.97	+0.03
54	bugswarm	13.67	11.24	5.62	10.18	+0.02
55	crosscodeeval-csharp	5.67	13.75	6.74	8.72	+0.01
56	crosscodeeval-python	8.00	14.30	5.62	9.31	0.00
57	curriculum-hard	10.33	15.83	3.75	9.97	-0.04
58	nemotron-bash-withtests	5.67	15.22	5.62	8.84	-0.04
59	wizardlm-orca	7.00	15.73	4.87	9.20	-0.04
60	curriculum-easy	10.33	14.22	4.49	9.68	-0.07

Table 12: **Full task generation strategy ranking (Part 1 of 2): ranks 1–60.** Continued in Table 13. Per-benchmark cells: raw accuracy (%). Average columns show raw and normalized z-score averages.

		Benchmarks (%)			Average	
Rank	Strategy	SWE-Bench Verified (100)	OT-TBLite	Terminal-Bench 2.0	Raw	Normalized
61	nemotron-csharp	9.30	12.70	5.62	9.21	-0.08
62	curriculum-medium	8.67	13.50	5.24	9.14	-0.11
63	nemotron-bash	7.33	11.43	6.74	8.50	-0.13
64	stack-bash-withtests-gpt5mini	7.00	14.97	4.49	8.82	-0.19
65	crosscodeeval-typescript	6.67	14.90	4.49	8.69	-0.22
66	defects4j	9.67	11.85	4.87	8.80	-0.29
67	nemotron-cpp	5.67	14.73	4.49	8.30	-0.30
68	agenttuning-alfworld	4.00	12.45	5.62	7.36	-0.42
69	qasper	8.00	11.51	4.87	8.13	-0.42
70	codeactinstruct	4.67	11.21	5.62	7.17	-0.51
71	inferredbugs	14.00	9.83	3.37	9.07	-0.51
72	crosscodeeval-java	6.33	12.81	4.12	7.75	-0.52
73	nemotron-rust	4.70	11.86	5.00	7.19	-0.56
74	taco	6.00	11.06	4.87	7.31	-0.58
75	magicoder	6.00	13.11	3.37	7.49	-0.65
76	toolscale	4.40	15.26	2.62	7.43	-0.65
77	orca-agentinstruct	5.00	13.18	3.37	7.18	-0.70
78	stack-go	7.33	12.42	3.00	7.58	-0.71
79	codenet-python	4.67	10.22	4.87	6.59	-0.75
80	nemotron-pytest	5.33	8.39	4.87	6.20	-0.90
81	codeforces	3.67	8.61	5.24	5.84	-0.91
82	e2egit	2.00	10.75	4.49	5.75	-0.92
83	nemo-prism-math	2.67	11.73	3.75	6.05	-0.92
84	codeelo	3.33	10.86	3.75	5.98	-0.97
85	pymethods2test	6.33	9.68	3.00	6.34	-1.05
86	quixbugs	5.67	12.94	1.30	6.64	-1.06
87	unitsyn-python	3.30	10.68	3.37	5.78	-1.06
88	agenttuning-kg	4.00	7.08	4.49	5.19	-1.18
89	all-puzzles	2.33	8.45	3.75	4.84	-1.27
90	code-contests	2.33	11.09	2.25	5.22	-1.27
91	agenttuning-webshop	1.67	9.14	3.37	4.73	-1.31
92	tulu3-sft-personas-math	3.00	9.30	2.62	4.97	-1.35
93	agenttuning-db	3.67	7.53	1.50	4.23	-1.70
94	agenttuning-mind2web	0.33	4.65	1.12	2.03	-2.26
95	agenttuning-os	0.00	5.64	0.37	2.00	-2.31

Table 13: **Full task generation strategy ranking (Part 2 of 2): ranks 61–95.** Continued from Table 12.

A.2 Mixing Strategies (Full Results)

Table 14 reports both presentation methods (random shuffle within task and sequential round-robin) for all values of N. The random-shuffle Top-4 result is reproduced in Table 3 of the main text.

Rank	Mixing Strategy	Benchmarks			Average	
		SWE-Bench Verified (100)	OT-TBLite	Terminal-Bench 2.0	Raw	Normalized
Random shuffle within task						
1	Top 4	**29.33** 1.63	**17.00** 1.71	**8.24** 1.24	**18.19**	**+0.49**
2	Top 2	**29.00** 1.60	**18.12** 1.72	**7.12** 1.06	**18.08**	**+0.48**
3	Top 8	28.00 1.70	15.86 1.70	**8.61** 1.24	**17.49**	**+0.19**
4	Top 16	26.67 1.49	**19.19** 1.70	4.12 1.06	16.66	-0.10
5	Top 32	20.33 1.67	**18.67** 1.72	5.99 1.06	15.00	-0.48
6	Top 1	**30.67** 1.67	14.80 1.40	4.49 0.84	16.65	-0.57
Sequential round-robin						
1	Top 8	**32.67** 1.86	16.28 1.58	**8.99** 1.40	**19.31**	**+0.46**
2	Top 4	28.00 1.76	17.86 1.60	**8.61** 1.40	**18.16**	**+0.28**
3	Top 2	28.67 1.70	**18.05** 1.60	**7.49** 1.24	**18.07**	**+0.11**
4	Top 32	23.00 1.76	**20.21** 1.64	6.37 1.18	16.53	-0.23
5	Top 16	27.33 1.83	17.68 1.55	5.62 1.18	16.88	-0.62

Table 14: **Full mixing strategy results.** We sweep $N \in \{1, 2, 4, 8, 16, 32\}$, sampling $10{,}000/N$ tasks from each of the top-N sources. Both random shuffling within a training batch and sequential round-robin presentation are evaluated. Mixing the top-4 to top-8 strategies yields the strongest balanced performance under both presentation methods. Per-benchmark cells: raw accuracy (%) with standard error as subscript. Bolded values are within 1 SE of the column-best.

Selection Strategy	Tokens	Benchmarks			Average
		SWE-bench Verified (100)	OT-TBLite	Terminal-Bench 2.0	Raw
Min turns ≥ 5 (filtered)	144.76M	**24.70** 1.56	**18.40** 1.66	**10.90** 1.35	**18.00**
Random subsample (control)	144.78M	19.30 1.63	17.10 1.48	7.10 1.24	14.50

Table 15: **The min-turns filter helps even at a matched token budget.** Both models are Qwen3-8B fine-tuned on subsets, each truncated to an equal $\sim$145M-token budget. Selecting the longest (≥ 5-turn) episodes outperforms a random subsample by $+5.4$pp on SWE-bench Verified-100 and $+3.8$pp on Terminal-Bench 2.0, while OT-TBLite is within noise. The benefit therefore reflects genuine multi-turn supervision rather than additional training compute.

A.3 Filtering for Longer Episodes: A Compute-Controlled Ablation

Section 3 shows that keeping execution traces with more model turns improves the resulting training set. Since longer ($\geq$ 5-turn) traces also pack more tokens into each example, the gain could in principle come from one of two sources: the higher-quality multi-turn supervision we intend to select for, or simply a larger training-compute budget. At a fixed row count, the min-turns filter sees roughly 45% more tokens than an unfiltered subset, so the two explanations are confounded unless we equalize the token budget. We do so directly. We build two training sets that consume the same $\sim$145M tokens: the first keeps the longest ($\geq$ 5-turn) episodes (9,859 rows), and the second draws a random subsample of the pool (14,470 rows). Everything else about the two runs is identical. Table 15 reports the result.

With the token budget held fixed, the min-turns filter still beats the random control by $+3.5$pp on average, with $+5.4$pp on SWE-Bench Verified-100 and $+3.8$pp on Terminal-Bench 2.0 while OT-TBLite moves within the per-benchmark noise. The effect therefore cannot be attributed to extra training compute and longer episodes lead to higher accuracy in multi-turn agentic tasks.

B Scaling at 8B

Our pipeline ablations (Section 3) use Qwen3-8B as the base model. Here we verify that the final OpenThoughts-Agent data recipe also scales at the 8B model size, mirroring the 32B trends in Figure 1. Figure 5 reports SWE-bench Verified-100 and Terminal-Bench 2.0 accuracy as we scale the OpenThoughts-Agent-v2 dataset from 316 to 100K rows, alongside the Nemotron-Terminal-Corpus baseline and the base Qwen3-8B model.

OpenThoughts-Agent leads the Nemotron-Terminal-Corpus baseline at most matched dataset sizes on both benchmarks. For example, at 10K rows it reaches 24.3% versus 9.3% on SWE-bench Verified-100. As with the 32B model, performance continues to improve at the largest scale rather than plateauing: at 100K rows the 8B reaches 39.7% on SWE-bench Verified-100 and 10.9% on Terminal-Bench 2.0 (up from 26.3% and 7.9% at 31.6K), surpassing the Nemotron-Terminal-Corpus baseline on both benchmarks (34.3% and 9.0%, respectively). This mirrors the effect of synthetic task augmentation at 32B (Section 4), where expanding task-description diversity overcomes the upsampling bottleneck.

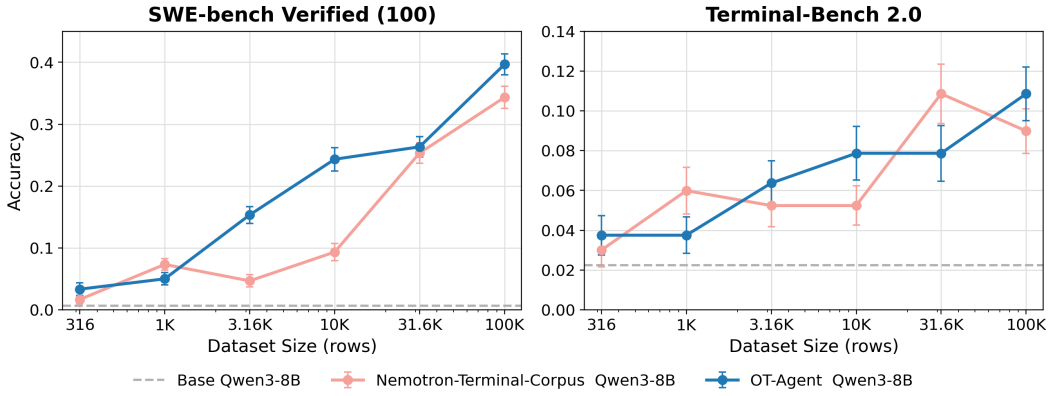

Figure 5: **The OpenThoughts-Agent data recipe scales at 8B.** SWE-bench Verified-100 and Terminal-Bench 2.0 accuracy for Qwen3-8B fine-tuned on OpenThoughts-Agent-v2 across dataset sizes, compared against the Nemotron-Terminal-Corpus baseline and the base Qwen3-8B model (dashed). OpenThoughts-Agent leads at the larger scales and surpasses the baseline on both benchmarks at 100K. Error bars denote standard error across three stochastic re-runs per task.

C SFT Hyperparameters

C.1 SFT training hyperparameters per data scale

Common to all 32B SFT runs. All 32B finetunes start from `Qwen/Qwen3-32B`, use the `qwen3` (thinking) chat template, and share the optimization settings below; only the items in Table 17 vary across data scales.

Optimizer	AdamW, $\beta = (0.9, 0.98)$, weight decay 0.04
Learning rate	4×10^{-5}
Schedule	cosine, warmup ratio 0.1
Global batch size	96
Sequence cutoff	32,768 tokens
Precision	BF16 weights
Distributed strategy	DeepSpeed ZeRO-3 (`ds_z3_accelerate.json`)
Hardware	24 nodes (JUPITER Booster, 4×GH200 per node)
Shuffle examples between epochs	disabled (`disable_shuffling=True`)
Chat template	`qwen3` (thinking)
Base model	`Qwen/Qwen3-32B`

Table 16: **Optimizer and infrastructure settings shared across all 32B SFT runs.**

Per-scale settings (32B). The two ablation knobs that vary across data scales are the number of training epochs and the gradient-clip threshold: smaller scales train for more epochs with looser clipping, larger scales train for fewer epochs with tighter clipping (Table 17).

Common to all 8B SFT runs. 8B runs use the same optimizer and infrastructure family as the 32B runs but with a non-thinking chat template and a longer training horizon (7 epochs at all scales). The constant settings are summarized in Table 18.

Data scale	Epochs	Max grad norm	Recipe label	Approx. wall time (24 nodes)
3.16K	7	1×10^{-4}	`32b_small`	$\sim$0.5 h
10K	7	1×10^{-4}	`32b_small`	$\sim$1.5 h
31.6K	5	1×10^{-3}	`32b_large`	$\sim$3 h
100K	5	1×10^{-3}	`32b_large`	$\sim$5 h

Table 17: **32B per-scale SFT settings.** The smaller-scale runs (3.16K, 10K) use the `32b_small` recipe (7 epochs, looser gradient clipping); the larger-scale runs (31.6K, 100K) use `32b_large` (5 epochs, tighter clipping). All other settings (Table 16) are held constant. Wall time is measured for the diverse-Tezos top-4 family on 24$\times$H100 nodes.

Optimizer	AdamW, $\beta = (0.9, 0.98)$, weight decay 0.04
Learning rate	4×10^{-5}
Schedule	cosine, warmup ratio 0.1
Global batch size	96
Sequence cutoff	$32{,}768$ tokens (long-context variants: $131{,}072$)
Precision	BF16 weights
Distributed strategy	DeepSpeed ZeRO-3
Hardware	24 nodes (JUPITER Booster, $4\times$GH200 per node)
Shuffle examples between epochs	disabled (`disable_shuffling=True`)
Chat template	`qwen3_nothink`
Base model	`Qwen/Qwen3-8B`

Table 18: **Optimizer and infrastructure settings shared across all 8B SFT runs.**

Data scale	Epochs	Max grad norm	Approx. wall time (24 nodes)
1K	7	1.0	$\sim$1 h
3.16K	7	1.0	$\sim$2 h
10K	7	1.0	$\sim$4 h
31.6K	7	1.0	$\sim$10 h
100K	7	1.0	$\sim$30 h

Table 19: **8B per-scale SFT settings.** 8B runs hold all hyperparameters constant across data scales, with only data volume changing. All settings in Table 18 apply.

D RL Hyperparameters and Infrastructure

D.1 Hardware

Site / scheduler	NERSC Perlmutter
Nodes	6 (2 shared policy / reference, 4 inference)
Accelerators	4 $\times$ NVIDIA A100-SXM4-80GB per node (24 GPUs total)
Interconnect / runtime	Cray Shasta, Ray cluster, CUDA 12.8, PyTorch bf16

D.2 Algorithm and optimizer

RL algorithm	RLOO with per-prompt std normalization (`grpo_norm_by_std=true`)
Loss reduction	token-mean
PPO clip range	$[\epsilon_{\text{lo}}, \epsilon_{\text{hi}}] = [0.2, 0.2]$, $c = 3$
KL control	No
Entropy regularization	No
Advantage normalization	per-batch
Optimizer (policy)	AdamW, $\beta = (0.9,\ 0.999)$, weight decay 0
Optimizer (critic)	AdamW, $\beta = (0.9,\ 0.999)$, weight decay 0.01
Learning rate	5×10^{-6}
Schedule	constant (no warmup)
Gradient clipping	global norm ≤ 1.0
Precision	bf16 autocast, fp32 grad accumulation

D.3 Training schedule

Global steps	48
Epochs	2
Train batch (prompts)	64
Samples per prompt	8 (so 512 trajectories per gradient update)
Policy mini-batch	64 prompts
Update epochs per batch	1
Micro-batch (per GPU)	1 (train), 4 (forward)
Sample packing	enabled
Gradient checkpointing	enabled (non-reentrant)
HF Hub save interval	every 5 steps

D.4 Distributed strategy

Strategy	FSDP2, `fsdp_size=4` (intra-node), CPU param offload
Reference / policy colocation	yes (shared 2-node pool, swapped on/off device)
Async generation	enabled, max staleness 16 steps, 768 parallel workers
Sequence parallel	1 (Ulysses backend available, unused)

D.5 Generation (vLLM)

Engine	vLLM async, 16 inference engines, eager mode
Sampling	temperature 0.7, top-p 0.95, top-k 20
Max generation length	4,096 tokens
Max model context	32,768 tokens
Prefix caching / chunked prefill	enabled
KV cache target utilization	0.9 of GPU memory
Eval sampling	greedy ($T = 0$, top-$k = -1$), $n = 8$ trajectories per prompt

D.6 Rollout environment (Harbor / terminus-2)

Harbor agent	`terminus-2` with interleaved thinking enabled
Per-trial sandbox	1 vCPU, 2,048 MB RAM, 2,048 MB storage
Per-trial timeout	1,800 s agent / 120 s verifier
Concurrent trials	280
Number of turns cap per task	unbounded (effectively gated by timeout)
Retry policy	3 retries, exponential backoff (60–600 s)
Failure-mode handling (Sandbox infra, network)	masked transient infra errors
zero reward on `TimeoutError`, parse errors, OOM	

D.7 Headline metrics at step 48

`reward/avg_pass_at_8`	0.281
`reward/avg_raw_reward`	0.107
`policy/policy_entropy`	0.072
`policy/raw_grad_norm`	0.018
Runtime	1.66×10^5 s ($\approx$46 h)

D.8 Reproducibility Artifacts

Pinned source commits (as of run start, 2026-02-27):

Repo	Branch	Commit
open-thoughts/OpenThoughts-Agent	main	`4e2b8422`
penfever/SkyRL (fork)	`penfever/working`	`ada3bd4f`
laude-institute/harbor	`penfever/temp-override`	`94f358bc`

Full wandb run at `dogml`, available on request. Public HF checkpoint available as laion/rl_swesmith-fixthink-pymethods2test-45. We selected a fork of Llama-Factory from [50], extending it to support ALST long-sequence training [4]. Our reinforcement learning framework was an extended version of the popular SkyRL framework from [15]; most of the improvements are described in [7]. We used [18] for environment, benchmark and harness management.

E RL Run-to-Run Reproducibility

A natural concern for any RL result is how much of the reported improvement is signal versus run-to-run noise in the training pipeline. To probe this, we evaluate three near-replicate RL runs of the `pymethods2test` experiment. All three start from the same GLM-4.7-distilled SWE-Smith 8B checkpoint and use the same RLOO recipe, environment, and 24$\times$A100 setup described in Section 5 and Section D; they differ only in minor training choices (the exported step and, for one run, the learning rate). Because the three runs share a configuration up to these small perturbations, the spread of their downstream eval scores is a direct, if conservative, estimate of the reproducibility of the RL pipeline. The headline checkpoint (`pymethods2test-45`) was additionally evaluated twice on every benchmark, which lets us separate *eval noise* (re-running the same checkpoint) from *training-run noise* (re-running the pipeline). Table 20 summarizes the comparison; per-cell variance combines within-eval binomial sampling with between-eval variance using the mixed estimator defined in the caption.

Replicate RL runs differ by only $\approx$1.6 points on ID and $\approx$2.0 points on OOD. The in-distribution (Core) set means lie within 20.2–21.8% (range 1.6 pp, cross-run standard deviation 0.8 pp) and the out-of-distribution means within 26.5–28.5% (range 2.0 pp, cross-run standard deviation 1.0 pp), including the learning-rate variant. This run-level spread is comparable to, and slightly larger than, the per-cell eval noise: the two repeated evaluations of `pymethods2test-45` differ by 0.3–3.7 points on most benchmarks, with the small-N FinanceAgent-Terminal (50 tasks) the main outlier at $\approx$11 points, consistent with its large binomial sampling error. At the benchmark level, larger denominators are correspondingly more stable: SWE-Bench-Verified (500 tasks) varies by only

Rank	RL Run	Benchmarks			Average	
		SWE-bench Verified (100)	OT-TBLite	Terminal-Bench 2.0	Raw	Normalized
1	`pymethods2test-45`	**35.67** 1.83	16.02 1.58	**13.48** 1.50	**21.72**	**+0.32**
2	`pymethods2test` (10-step)	**35.67** 1.83	**19.30** 1.71	8.61 1.18	21.19	+0.20
3	`pymethods2test` (lr5e-6)	31.33 1.63	15.36 1.57	**12.36** 1.45	19.68	-0.51

Table 20: **Run-to-run reproducibility of the 8B agentic RL pipeline.** Three near-replicate RL runs of the `pymethods2test` experiment, all starting from the same GLM-4.7-distilled SWE-Smith 8B checkpoint and trained with RLOO on 24×A100, agree; the `lr5e-6` run uses a learning rate of $5e{-6}$ rather than the default of the other two.

1.5 points across runs while the 50–127-task OOD benchmarks vary by 2.7–5.3 points, indicating that most of the per-benchmark variability is eval sampling noise rather than genuine training-run differences. The two single-eval runs (10-step and lr5e-6) are reported with binomial-only error bars, since a single eval cannot estimate run-level variance; their means nonetheless fall inside the band set by the twice-evaluated headline run.

Takeaway. The pipeline is reproducible at the ∼2-point level: the gains reported in Table 10 – the ≈5-point RL-specific gain on the Core benchmarks over the SFT-only checkpoint (e.g. +5.4 on SWE-Bench-Verified), as well as the ≈18-point gain of the full SFT+RL pipeline over the Qwen3-8B base – are substantially larger than the ≈1.6/2.0-point ID/OOD run-to-run spread measured here, so they are unlikely to be artifacts of training-run noise. Reporting set-level means with the mixed-variance error above, rather than single best-of-many eval numbers, is the appropriate way to compare RL checkpoints at this scale.

F What RL Learns: Emergent Behavior Behind the `pymethods2test` Result

Table 9 establishes *that* `pymethods2test` is the strongest RL data source, but not *why*. Here, we offer a detailed mechanistic account of the behavioral changes that emerge after RL. We compare the headline `pymethods2test` run to the `llm-verifier-freelancer` run, the strongest OOD alternative in Table 9). Our analysis combines three views: (i) time-binned behavioral statistics over the RL traces themselves (∼11k hero rollouts, ∼53k baseline rollouts); (ii) per-trace behavioral deltas between the pre-RL base checkpoint and the post-RL checkpoint, measured on held-out eval traces; and (iii) a pairwise LLM judge (`gpt-5-2025-08-07`) that reads 30 same-task pre/post trace pairs per run and reports a winner, confidence, and behavioral tags.

F.1 Legitimate exploration, not reward hacking

The largest and most consistent shifts are all increases in *exploratory activity*. Reasoning expands sharply – think tokens per trace more than double ($30.3 \to 65.4$, $+116\%$) and think blocks roughly double ($0.21 \to 0.44$) – alongside more self-correction ($0.63 \to 1.14$ phrases per trace, $+81\%$), more tool calls ($31.3 \to 40.9$, $+31\%$), more assistant messages ($20.0 \to 26.3$), and longer conversations ($+12.9$ turns, $+29\%$ tokens). The post-RL policy is thus *more* exploratory than even the already-verbose distilled base. Over 30 same-task pre/post trace pairs, the judge prefers the post-RL hero policy on $25/30$ (83.3%, 0 ties, $24/30$ high-confidence). The most frequent behavioral tags are `more-tool-calls` (73% of pairs), `longer-trace` (53%), `more-tool-errors` (50%), and `different-solution-strategy` (47%) – the judge independently sees the same expansion the metrics show. The change is *not* uniformly "more verbose," however: $8/30$ pairs (27%) are tagged `fewer-tool-calls/shorter-trace`, i.e. on a sizable minority of tasks RL taught the policy to *condense*.

Three observations argue that this is genuine capability change rather than a reward-hacking or formatting artifact. First, the post-RL policy emits *more* tool calls *and* more broken tool calls in absolute terms ($5.9 \to 8.8$ tool errors per trace, $+48\%$), which is the opposite of what a policy gaming a brittle parser would do. Second, the per-call tool *error rate* rises only $+4.1$ pp ($31.6\% \to 35.8\%$): the extra absolute errors are mostly explained by issuing more calls, not by tool-use competence degrading. Third, the share of `mark_task_complete` calls actually *rises* ($1.9\% \to 3.0\%$), inconsistent with a policy that learned to end early to harvest a formatting reward. The behavioral expansion converts to held-out gains: on the 100 shared SWE-Bench-Verified tasks, RL flips 18 tasks from fail to pass against a single regression (`sympy__sympy-15017`).

Per-trace behavioral feature	Pre-RL	Post-RL	Δ (rel.)
think tokens / trace	30.3	65.4	+35.1 (+116%)
think blocks / trace	0.21	0.44	+0.23 (+110%)
self-correction phrases / trace	0.63	1.14	+0.51 (+81%)
assistant tokens / trace	4254	6499	+2245 (+53%)
tool errors / trace	5.9	8.8	+2.9 (+48%)
tool responses / trace	18.5	24.6	+6.1 (+33%)
assistant msgs / trace	20.0	26.3	+6.2 (+31%)
tool calls / trace	31.3	40.9	+9.6 (+31%)
mean tokens / asst. msg	204.7	243.3	+38.6 (+19%)
tool error rate	31.6%	35.8%	+4.1 pp
think / assistant ratio	0.5%	0.6%	+0.1 pp
code fences / trace	0.15	0.14	−0.01 (−7%)
Macro metrics (whole-trace)			
mean turns / trace	40.5	53.3	+12.9 (+32%)
mean tokens / conversation	18432	23686	+5254 (+29%)
Tool-call mix (share of all tool calls)			
`mark_task_complete`	1.9%	3.0%	+1.1 pp
`bash_command`	98.1%	97.0%	−1.1 pp

Table 21: **Behavioral shift induced by `pymethods2test` RL, measured on held-out SWE-bench-Verified eval traces.** Per-trace means computed over 300 eval rows for the pre-RL base checkpoint (`...GLM_4_7_swesmith_sandboxes...`) versus the post-RL checkpoint (`...rl__24GPU_base__exp_rpt_pymethods2test...`); the same 100 underlying SWE-bench-Verified tasks are run ×3 on each side. Features are sorted within each block by the magnitude of the (scale-normalized) delta. The dominant change is a large increase in *exploratory activity*: the agent thinks more, calls more tools, emits more assistant messages, and self-corrects more. Crucially the per-call *tool error rate* rises only +4.1 pp, so the +48% rise in absolute tool errors is mostly a by-product of issuing more calls rather than degraded tool-use competence, and the share of `mark_task_complete` calls actually rises (+1.1 pp) – evidence against a trivial reward-hacking or formatting-only explanation. On these same 100 tasks the policy flips 18 from fail to pass against a single regression.

Why this data source pushed the policy to explore. The RL-time reward trajectory (Figure 6, blue) is the key context: `pymethods2test` presents a *medium, non-saturated* reward (it hovers around 0.47–0.51, far from the ceiling) that is *hard to improve*. Under RLOO this is exactly the regime that rewards trying harder – thinking more, calling more tools, attempting more fixes – because incremental returns come from working a problem more thoroughly rather than from a quick exploit. Across the run, mean conversation length and think budget grow while the reward plateaus, until the policy over-extends. We read the collapse as the downside of the same exploration pressure that produced the gains, not as a separate failure: the reproducibility study in Section E (which exports at step 45, and a 10-step and an `lr5e-6` variant) is the corresponding evidence that the *pre-collapse* regime is stable to re-run.

F.2 The contrast: the LLM-verifier run compacts instead of explores

Running the identical pipeline on `llm-verifier-freelancer` produces the mirror image (Table 22). Its RL-time reward rises *near-monotonically* from ≈ 0.54 to ≈ 0.73 with no collapse (Figure 7), and along the way the policy *compacts*: mean turns fall (−7.8), tool calls fall (−8%), think tokens fall (−42%), and self-correction phrases fall sharply (19.7 → 8.0 per trace, −59%), while tokens per conversation stay essentially flat (+1%). Every behavioral axis on which the hero run moved *up*, the baseline run moved *down*. The LLM judge sees the same thing from the other direction: it prefers the post-RL baseline policy on 22/30 pairs (73.3%), but now the dominant tags are `fewer-tool-calls` (53%), `shorter-trace` (40%), and `different-tool-strategy` (40%). Its top judgments describe a pre-RL policy trapped in loops of malformed tool arguments that RL taught to stop dithering and ship; on `financeagent-14` (post-RL wins, high confidence) "the baseline meandered with many turns (113) and tool calls (55) … The post-RL trace kept things short (12 turns, 6 tool calls) … avoided EDGAR API usage and did not self-correct, proceeding more directly toward the target metric."

Axis (pre-RL → post-RL)	`pymethods2test` (hero)	`llm-verifier-freelancer` (baseline)
RL-time reward trajectory	$0.47 \to 0.51$ peak $\to 0.14$ collapse	$0.54 \to 0.73$ near-monotonic
tail-bin agent-timeout rate	$\approx 80\%$	stayed 18–36% throughout
completed-trial reward (errors excl.)[‡]	$0.652 \to 0.541$ (-0.11)	$0.415 \to 0.493$ ($+0.08$)
mean turns / trace	$+12.9$ ($+32\%$)	-7.8 (-11%)
mean tokens / conversation	$+29\%$	$+1\%$ (flat)
tool calls / trace	$+31\%$	-8%
think tokens / trace	$+116\%$	-42%
self-correction phrases	$+81\%$	-59%
LLM-judge post-RL win rate	83.3% (25/30)	73.3% (22/30)
dominant LLM-judge tag	`more-tool-calls` (73%)	`fewer-tool-calls` (53%)

Table 22: **Two RL runs on the same pipeline learn opposite behavioral policies.** The `pymethods2test` "hero" run and the `llm-verifier-freelancer` "baseline" run share the identical pipeline (GLM-4.7-distilled SWE-Smith 8B base, RLOO recipe, rollout environment, 24×A100), differing only in the RL data source. On *every* behavioral axis the two runs move in opposite directions: the hero policy *expands* (more turns, tokens, tool calls, thinking, self-correction) while the baseline policy *compacts* (fewer turns, calls, thinking, self-correction). The hero's expansion coincides with a non-monotonic reward curve that peaks near step ∼35 and then collapses (Figure 6); the baseline's compaction coincides with a smooth, monotone reward rise (Figure 7). We read this as exploration-purchased-at-the-cost-of-stability (hero) versus tightening-of-an-existing strategy (baseline). RL-time reward and behavioral deltas are from the time-binned trace analysis and the pre/post eval-trace deltas; LLM-judge win rates are pairwise `gpt-5-2025-08-07` comparisons over 30 same-task trace pairs per run. [‡] Mean reward over trials that produced a reward (errored/timed-out trials excluded) – a *conditional* reward, distinct from the all-trial Harbor Mean accuracy (errors= 0) reported in Tables 9 and 10. The hero's conditional reward dips here while its all-trial accuracy roughly doubles ($0.19 \to 0.33$); see Section F.3.

F.3 Connecting behavior to downstream evals

The behavioral changes persist into held-out evaluation and track the downstream ranking in Table 9. On the 300-row SWE-Bench-Verified eval, the pre-collapse hero checkpoint roughly *doubles* the base policy's *all-trial* reward – the Harbor Mean accuracy that counts errored/timed-out trials as 0, the same metric reported in Tables 9 and 10 (post-RL 0.33 vs. pre-RL 0.19; Figure 6, markers) – and flips 18 tasks fail→pass. This all-trial accuracy rises *even though* the mean reward over *completed* trials dips ($0.652 \to 0.541$, Table 22): RL converts many trials the base used to abandon to timeouts into completed attempts, and because those rescued trials are the harder ones, the conditional average falls while the all-trial accuracy climbs The baseline's compaction also helps its own evals (post-RL 0.224 vs. pre-RL 0.173 on the 156-row FinanceAgent eval), which is why `llm-verifier-freelancer` is competitive on OOD in Table 9; but compaction transfers less well to ID Core benchmarks that reward thoroughness.

G Eval Setup

Evaluation Stack Overview For evaluation, we serve models using vLLM [25], orchestrate agent trials through Harbor [18], and isolate each trial in a Daytona [9] sandbox. We evaluate on the following benchmarks:

- **Terminal Bench 2.0** (89 tasks) [28]: Hand-crafted, human-verified tasks that spans SWE, biology, security, system administration, and machine learning.
- **SWE-Bench-Verified-100** (100 tasks) [23]: a stratified subsample (by repository) of the 500-task human-validated SWE-Bench Verified split where agent produces a patch against a Python repo at a fixed commit.
- **SWE-Bench-Verified** (500 tasks) [23]: full SWE-Bench Verified split, same format as above.

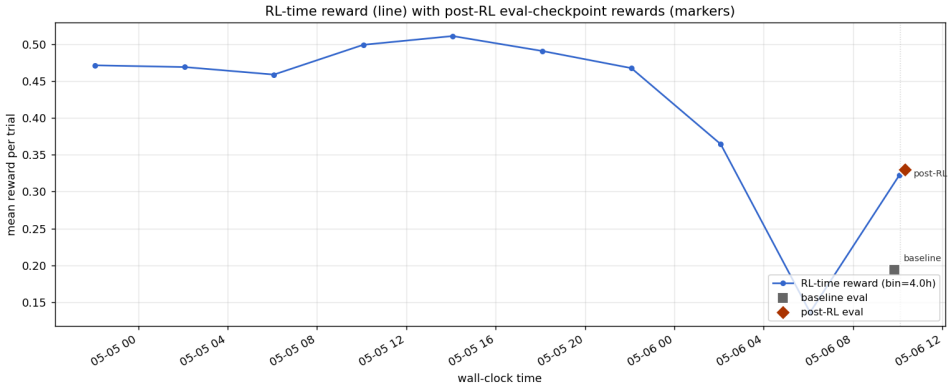

Figure 6: **Hero (`pymethods2test`) RL-time reward peaks and then collapses.** Mean reward per rollout in 4-hour wall-clock bins (blue line) over the hero run, with the post-RL and pre-RL eval-checkpoint mean rewards on held-out SWE-Bench-Verified marked on the right (diamond = post-RL, square = pre-RL base). Reward rises modestly to a peak near 0.51, then collapses to ≈ 0.13 as the policy over-explores (mean turns and think tokens spike while productive tool calls fall and the agent-timeout rate reaches $\approx 80\%$). The deployed checkpoint is taken from *before* this collapse; it roughly doubles the base policy's eval reward (0.33 vs. 0.19).

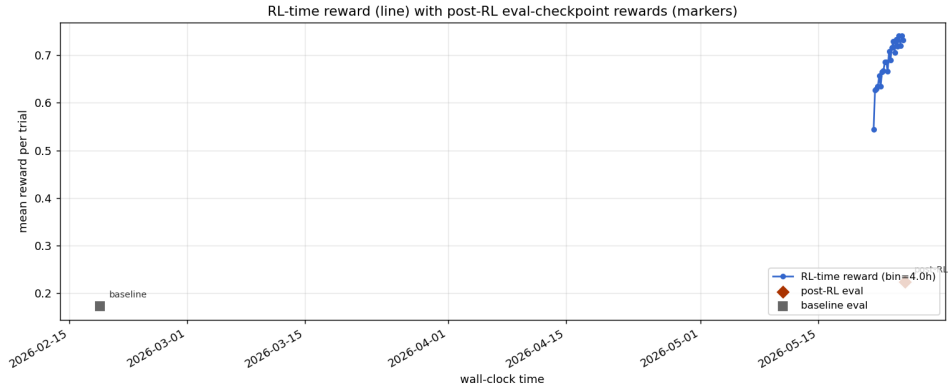

Figure 7: **Baseline (`llm-verifier-freelancer`) RL-time reward rises near-monotonically.** Mean reward per rollout in 4-hour bins (blue line, clustered at right of the wall-clock axis because the pre-RL base eval – gray square – predates training by months) rises smoothly from ≈ 0.54 to ≈ 0.73 with no collapse, the mirror image of Figure 6. The corresponding policy *compacts* (fewer turns, tool calls, and think tokens; see Table 22) rather than expands.

- **OpenThoughts-TBLite** (100 tasks) [32]: a curated collection of 100 Terminal-Bench style tasks that balanced across four difficulty buckets and engineered as a fast proxy for the full Terminal-Bench 2.0 performance;
- **Aider Polyglot** (225 tasks) [2]: A multi-language code-editing benchmark (C++, Go, Java, JavaScript, Python, Rust) sourced from Exercism and filtered to the harder problems.
- **BFCL-Parity** (123 tasks) [33]: Berkeley Function Calling Leaderboard (BFCL) is a comprehensive benchmark for evaluating large language models' ability to call functions/tools correctly based on natural language instructions. This is a stratified random subsample of BFCL v4.
- **MedAgentBench** (300 tasks) [22]: A clinical-agent benchmark over a vendored FHIR server populated with ∼700k records.
- **GAIA-127** (127 tasks) [29]: A text-only subset of the GAIA validation split for general-AI-assistant multi-step factoid QA with web browsing, file handling, and tool use. For this benchmark, we evaluated in a default setting without setting a search API.
- **FinanceAgent-Terminal** (50 tasks) [6]: A terminal-agent variant of Vals AI's Finance Agent Benchmark over SEC 10-K/10-Q filings, with shell access to EDGAR/web/RAG

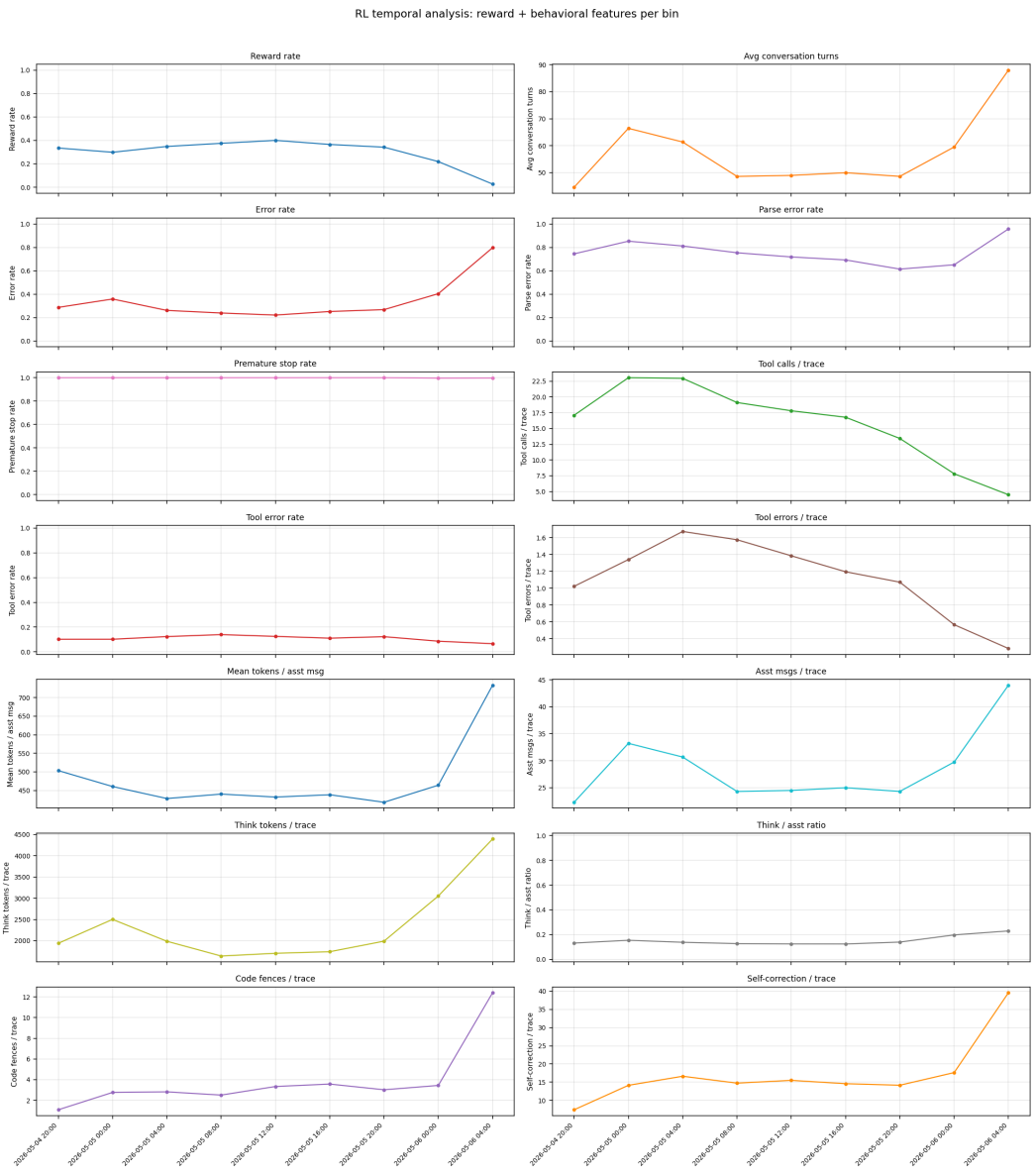

Figure 8: **Time-binned behavioral dynamics of the hero (`pymethods2test`) run.** Reward, error/parse-error/premature-stop rates, and nine per-trace behavioral features over the run (4-hour bins on a shared RL-time axis). The exploration-then-collapse signature is visible across panels: as reward plateaus and then falls, mean conversation turns, mean tokens per assistant message, think tokens per trace, and self-correction per trace all spike in the final bins, while tool calls per trace and tool errors per trace fall – the policy substitutes long internal reasoning for productive tool use and times out. This is the run-level analogue of the per-task over-exploration the LLM judge flags on tasks like `django__django-13128`.

tools. For this benchmark, we evaluated while providing SERP and EDGAR APIs available to the models.

All datasets are prepared in Harbor format, and for OOD benchmarks we generated the dataset using Harbor adapter [18].

Evaluation Setup For every (model, benchmark, harness) combination, we performed 3 repeated runs and report average pass@1 accuracy and standard error across runs. All evaluations with our trained model run with a 32K context window, a 16K output cap, and a proactive summarization

threshold of 2048 tokens. Each fire defaults to 32 concurrent trials, with Daytona snapshot registered and reused whenever available to avoid environment rebuilding during repeated evaluations. This mitigate the occurrence of transient environment build errors and speed up evaluations overall. For every baseline model we report, we run the above procedure twice: once under `terminus-2` (parity with the setup of our trained model) and once under the *preferred scaffold and serving configuration* indicated from the original report.

Timeout-aware Evaluation Agentic trials are token-bound: a trial that takes ten minutes for an 8B model decoding at $\sim$150 tokens/s may take an hour for a 32B model decoding at $\sim$30 tokens/s, with the *same* number of agent turns. Holding per-trial timeouts constant across model sizes therefore confounds accuracy — slower models time out on tasks faster ones complete. To compensate this, we applied a timeout multiplier m (through the `--timeout-multiplier` flag in harbor configuration). We use $m = 2$ for 8B fires and $m = 16$ for 32B fires. To avoid overly time-consuming evaluations, after applying the multipliers we employed a 2-hour cap on agent execution regardless of m:

$$\text{effective\_timeout} \;=\; \min\big(\text{cap},\; \text{base} \times m\big).$$

For larger models where we cannot get comparable throughput, we used third-party API to ensure fair comparison.

Coderforge-Preview We do not include in the main results table open-data releases that did not release a model; the most prominent example of this is CoderForge-Preview [3].

Reproducing OpenSWE OpenSWE [11] reports 62.4% on SWE-Bench-Verified (full 500-task split) training from a Qwen-2.5-32B base. We attempted to reproduce this score using the publicly released `GAIR/OpenSWE-32B` checkpoint. To match their setup, we ran with the same temperature=0.7, 128K-token context window, 300 step budget, and together with a SWE-Agent harness configuration the authors shared with us in correspondence. The OpenSWE repository mainly provides detail for data pipelines, but scaffold or evaluation-harness components were not publicly released at the time of writing, so we cannot guarantee an exact match in every internal detail. On our SWE-Bench-Verified-100 subset we measure 44.0%; we do not yet have a directly comparable number on the full 500-task split, so the gap to the reported 62.4% is partly across different denominators in addition to whatever methodological differences remain. We have not been able to fully attribute the residual gap and continue to investigate. In the meantime, we flag OpenSWE in Table 23 and exclude it from main Table 1, while still citing the reported number with an underline marker.

Evaluation on Terminal-Bench 2.1. Concurrent with our experiments, Terminal-Bench 2.1 was released, which revises 31% of Terminal-Bench 2.0 tasks (28/89 tasks) to correct for three defect classes: specification–verification mismatches, resource mismatches, and benchmark drift [43]. Because Terminal-Bench 2.0 is one of our two core benchmarks, we re-evaluated a representative baseline (Qwen 3.5-27B) on Terminal-Bench 2.1 to verify that our conclusions hold after a new release. The table Table 24 reports the accuracy in TB2.0 and TB2.1. The observed change (3.0 pp) is far smaller than the up-to-12.0 pp shifts the audit reports for frontier agents [43]. We therefore continue to report TB2.0 results throughout, as re-evaluating every model on TB2.1 would be cost-prohibitive.

	Model	OpenThoughts-TBLite	SWE-Bench-Verified-100	SWE-Bench-Verified	Terminal-Bench 2.0	Aider-Polyglot (OOD)	BFCL-Parity	MedAgentBench	GAIA-127	FinanceAgent-Terminal	Avg (Avg)
8B Scale	Nemotron-Terminal-8B [35]	**25.8**$_{1.8}$	21.7$_{1.7}$	22.1$_{0.7}$	**13.1**$_{1.4}$	12.6$_{1.0}$	59.9$_{1.6}$	**48.6**$_{1.2}$	**14.4**$_{1.3}$	11.3$_{2.0}$	26.0
	SWE-Lego-Qwen3-8B [40]	3.2$^{*}_{0.8}$	**42.3**$^{\dagger}_{1.9}$	**37.6**$^{*}_{0.8}$	0.7$_{0.4}$	17.5$^{\dagger}_{1.0}$	**75.3**$^{\dagger}_{1.1}$	5.0$_{0.7}$	6.0$^{*}_{0.8}$	1.3$_{0.9}$	20.5
	Endless-Terminal-OpenThinker-8B [13]	13.6$_{1.4}$	11.3$_{1.5}$	12.5$_{0.6}$	8.2$_{1.1}$	**18.8**$_{1.0}$	48.8$_{1.6}$	18.4$_{1.0}$	5.2$_{0.8}$	10.0$_{2.1}$	17.4
	OpenThinker-Agent-v1	10.9$_{1.4}$	10.0$_{1.4}$	14.0$_{0.6}$	3.7$_{1.0}$	16.7$_{1.1}$	40.9$_{1.7}$	20.0$_{1.0}$	4.5$_{0.8}$	**14.7**$_{2.2}$	16.4
	SERA-8B [38]	5.9$_{1.1}$	31.7$^{\bullet}_{1.7}$	31.5$^{\bullet}_{0.7}$	2.6$^{\bullet}_{0.8}$	**19.1**$^{\bullet}_{1.0}$	24.1$_{1.8}$	12.1$_{2.2}$	2.1$_{0.6}$	1.3$_{1.2}$	13.3
	Qwen3-8B [45]	5.9$_{0.8}$	13.7$^{\P}_{1.4}$	13.2$^{\bullet}_{0.6}$	2.2$^{\diamond}_{0.5}$	11.7$^{\bullet}_{0.9}$	34.4$_{1.6}$	3.8$_{0.6}$	5.0$_{0.6}$	1.3$_{0.7}$	10.2
	SETA-RL-Qwen3-8B [39]	6.4$_{1.0}$	0.7$_{0.5}$	1.4$_{0.3}$	1.5$_{0.7}$	2.1$_{0.4}$	25.2$_{1.7}$	3.2$_{0.6}$	6.8$_{0.7}$	4.7$_{1.5}$	6.4
	SWE-agent-LM-7B [47]	1.7$^{\dagger}_{0.3}$	15.3$^{\dagger}_{1.3}$	16.2$^{\dagger}_{0.6}$	0.7$^{\dagger}_{0.4}$	3.1$^{\dagger}_{0.4}$	0.0$^{\dagger}_{0.0}$	0.8$^{\dagger}_{0.3}$	0.5$_{0.4}$	0.0$_{0.0}$	3.0
	Llama-3.1-Nemotron-Nano-8B-v1 [5]	1.0$_{0.5}$	0.0$_{0.0}$	0.4$_{0.0}$	0.0$_{0.0}$	2.5$_{0.1}$	2.2$_{0.6}$	0.2$_{0.2}$	0.0$_{0.0}$	0.0$_{0.0}$	0.8
	OpenThinker3-7B [16]	1.0$_{0.0}$	0.0$_{0.0}$	0.4$_{0.0}$	0.7$_{0.5}$	2.7$_{0.0}$	0.3$_{0.3}$	0.0$_{0.0}$	0.0$_{0.0}$	0.0$_{0.0}$	0.6
	DeepSeek-R1-Distill-Qwen-7B [17]	1.0$_{0.0}$	0.0$_{0.0}$	0.4$_{0.0}$	0.4$_{0.4}$	2.5$_{0.1}$	0.0$_{0.0}$	0.0$_{0.0}$	0.3$_{0.3}$	0.0$_{0.0}$	0.5
32B Scale	OpenThinkerAgent-32B	41.3$_{2.1}$	**55.7**$_{1.3}$	54.0$_{0.7}$	**26.2**$_{1.6}$	**32.4**$_{1.2}$	**85.9**$_{1.1}$	47.8$_{1.0}$	**23.6**$_{1.5}$	**44.0**$_{2.8}$	44.8
	Nemotron-Terminal-32B [35]	**48.6**$_{1.9}$	45.0$_{1.9}$	41.9$_{0.8}$	**25.1**$_{1.5}$	24.9$_{1.3}$	69.1$_{1.1}$	**62.6**$_{0.9}$	**22.3**$_{1.2}$	40.7$_{3.5}$	40.9
	GLM-4.7-Flash [41]	24.5$_{1.9}$	46.7$_{1.6}$	45.0$_{0.8}$	15.4$_{1.5}$	20.1$_{1.1}$	77.8$_{1.3}$	40.9$_{1.1}$	**22.8**$_{1.7}$	**42.0**$_{3.0}$	37.7
	Nemotron-Terminal-14B [35]	34.7$_{1.9}$	35.3$_{1.8}$	31.5$_{0.8}$	21.0$_{1.6}$	18.4$_{1.2}$	75.9$_{1.5}$	56.9$_{1.1}$	16.3$_{1.2}$	30.7$_{3.3}$	35.8
	SWE-Lego-Qwen3-32B [40]	25.6$^{\dagger}_{1.9}$	51.0$^{*}_{1.5}$	51.0$^{*}_{0.7}$	16.1$^{*}_{1.5}$	30.1$^{*}_{1.1}$	81.0$^{*}_{0.8}$	36.2$_{1.1}$	12.9$^{*}_{1.2}$	15.3$_{2.4}$	34.7
	daVinci-Dev-32B [49]	22.2$_{1.9}$	**55.3**$^{\dagger}_{1.3}$	54.5$^{\dagger}_{0.8}$	12.0$_{1.6}$	21.3$^{\dagger}_{0.9}$	64.8$_{1.8}$	38.0$_{1.2}$	11.5$_{1.4}$	21.3$_{3.0}$	31.9
	Qwen3-Coder-30B-A3B-Instruct [45]	23.3$^{*}_{2.0}$	47.7$^{\dagger}_{1.5}$	<u>51.6</u>$^{*}_{0.0}$	13.9$^{*}_{1.4}$	27.6$^{\dagger}_{1.1}$	67.8$_{1.3}$	20.9$_{1.0}$	11.3$^{*}_{1.0}$	12.7$_{2.0}$	29.4
	SERA-32B [38]	20.9$_{1.7}$	48.7$^{\P}_{1.6}$	49.4$^{\bullet}_{0.7}$	9.7$^{\P}_{1.2}$	26.7$^{\P}_{1.1}$	69.1$_{1.6}$	15.6$_{1.0}$	8.7$_{1.2}$	17.3$_{2.6}$	28.1
	SA-SWE-32B [8]	15.3$_{1.8}$	36.0$^{\P}_{0.0}$	<u>39.4</u>$^{\P}_{0.0}$	<u>16.2</u>$^{*}_{0.0}$	17.3$^{*}_{1.0}$	74.8$_{1.6}$	15.8$^{*}_{1.0}$	11.5$_{0.9}$	13.3$_{2.5}$	26.9
	DeepSWE-Preview [27]	13.3$_{1.6}$	38.7$^{\S}_{0.0}$	<u>42.2</u>$^{\S}_{0.0}$	4.9$_{1.0}$	27.3$^{\dagger}_{1.3}$	77.2$_{1.4}$	8.7$_{0.8}$	16.5$_{1.4}$	10.0$_{2.2}$	26.7
	Nemotron-Nano-30B-A3B [31]	22.4$_{2.1}$	33.3$^{*}_{1.7}$	30.6$^{*}_{0.7}$	6.4$^{\dagger}_{1.2}$	21.5$^{*}_{1.1}$	81.6$^{*}_{1.5}$	16.2$_{1.0}$	14.7$^{*}_{1.1}$	11.3$_{2.6}$	26.0
	SWE-agent-LM-32B [47]	22.3$_{1.8}$	41.7$^{\dagger}_{1.2}$	40.7$^{\dagger}_{0.6}$	13.9$_{1.4}$	20.0$^{\dagger}_{0.9}$	40.9$_{2.1}$	32.9$_{1.2}$	8.7$_{1.0}$	11.3$_{2.6}$	24.1
	Qwen3-32B [45]	13.7$_{1.8}$	26.7$^{\P}_{1.3}$	29.1$^{\bullet}_{0.7}$	7.5$^{\diamond}_{0.9}$	28.9$^{\P}_{1.2}$	68.3$_{1.4}$	6.8$_{0.7}$	9.7$_{1.0}$	9.3$_{1.8}$	22.8
	LiteCoder-Terminal-30B-A3B [34]	29.7$_{2.2}$	19.3$_{1.6}$	16.7$_{0.7}$	13.5$_{1.5}$	22.2$_{1.1}$	55.0$_{1.6}$	23.1$_{1.1}$	11.5$_{1.1}$	16.0$_{2.7}$	22.6
	Qwen2.5-Coder-32B-Instruct [19]	10.1$^{\dagger}_{1.4}$	14.7$^{\dagger}_{1.4}$	10.0$^{\diamond}_{0.5}$	7.1$^{\dagger}_{1.0}$	13.3$^{\diamond}_{0.8}$	60.4$^{*}_{0.9}$	37.4$_{1.1}$	6.8$_{1.0}$	14.7$_{2.3}$	21.4
	R2EGym-32B-Agent [21]	14.9$_{1.7}$	33.0$^{\S}_{0.0}$	<u>34.4</u>$^{\S}_{0.0}$	9.7$^{\diamond}_{0.9}$	15.9$^{\diamond}_{0.7}$	49.1$_{1.9}$	25.2$_{1.2}$	6.0$_{0.9}$	2.7$_{1.3}$	20.4
	Qwen3-30B-A3B-Instruct-2507 [45]	15.6$_{1.6}$	5.7$_{1.1}$	6.7$_{0.5}$	6.7$_{1.2}$	19.9$_{1.1}$	58.5$_{1.4}$	12.1$_{0.8}$	8.1$_{0.9}$	6.7$_{1.8}$	17.0
	OpenSWE-32B [11]	4.7$^{\dagger}_{0.9}$	44.0$^{\dagger}_{1.9}$	**62.4**$^{\dagger}_{0.0}$	3.4$^{\dagger}_{0.8}$	10.1$^{\dagger}_{0.9}$	0.5$_{0.4}$	3.0$^{\dagger}_{0.5}$	0.0$^{\dagger}_{0.0}$	0.0$_{0.0}$	11.3
Frontier	Kimi-K2.5 [42]	**72.1**$_{1.9}$	**70.3**$_{1.3}$	70.7$_{0.6}$	40.1$_{1.8}$	**71.9**$_{1.1}$	**86.4**$_{0.9}$	49.4$_{0.9}$	**58.3**$_{1.7}$	65.3$_{3.1}$	63.2
	GLM-5 [14]	**70.6**$_{2.0}$	**70.0**$_{1.4}$	<u>72.8</u>$^{\S}_{0.0}$	**46.1**$_{1.7}$	56.3$_{1.1}$	**86.2**$_{0.7}$	45.3$_{0.7}$	**58.3**$_{1.7}$	**72.0**$_{2.6}$	62.4
	Qwen3.5-27B [36]	62.2$_{2.0}$	60.3$_{1.3}$	62.7$_{0.6}$	40.1$_{1.8}$	50.5$_{1.0}$	**86.4**$_{0.8}$	**64.7**$_{0.7}$	44.9$_{1.6}$	55.3$_{2.8}$	57.8
	GLM-4.7-FP8 [41]	61.2$_{2.1}$	64.7$^{\dagger}_{1.7}$	63.9$^{\diamond}_{0.7}$	32.7$_{1.4}$	46.5$_{1.1}$	**86.2**$_{0.6}$	55.4$_{0.8}$	45.4$_{1.7}$	60.0$_{3.1}$	55.7
	Qwen3.5-9B [36]	32.7$_{2.0}$	48.7$_{1.8}$	49.0$_{0.8}$	19.9$_{1.7}$	30.1$_{1.1}$	79.4$_{1.1}$	47.6$_{1.2}$	24.9$_{1.6}$	48.0$_{2.8}$	42.7
	Qwen3-Coder-480B-A35B-Instruct-FP8 [45]	41.1$_{2.2}$	44.0$_{1.8}$	40.2$_{0.8}$	21.0$_{1.7}$	44.3$_{1.2}$	73.4$_{1.1}$	36.2$_{0.8}$	22.6$_{1.5}$	38.7$_{2.6}$	39.5
	Qwen3-235B-A22B-Instruct [45]	40.5$_{2.2}$	15.7$_{1.6}$	15.5$_{0.7}$	13.1$_{1.7}$	34.8$_{1.2}$	81.6$_{1.0}$	46.2$_{0.9}$	15.5$_{1.4}$	27.3$_{2.8}$	33.4

Table 23: **OpenThinkerAgent-32B is the strongest 32B-Scale model by average accuracy across the seven benchmarks shared with main Table 1.** Each cell reports the model's best accuracy (%) across evaluated agent harnesses; the subscript is that cell's per-run standard error. The Avg column averages each model's accuracy over the same seven benchmarks used in main Table 1 (SWE-Bench-Verified, Terminal-Bench-2, Aider-Polyglot, BFCL-Parity, MedAgentBench, GAIA-127, FinanceAgent-Terminal); OpenThoughts-TBLite and SWE-Bench-Verified-100 are shown as columns but excluded from the average. Models are grouped into scale bands (8B, 32B, Frontier). Within each band, cells whose accuracy lies within 1 Standard Error of the group's maximum on that benchmark are bolded.

Harness symbols. Cells without a superscript use the default harness terminus-2. Non-default symbols: $*$ = OpenHands [44], † = regular SWE-Agent [46], • = SERA-SWE-Agent (the SERA team's SWE-Agent config), § = R2EGym-Edit-Agent (a custom ReAct scaffold implemented in R2EGym evaluation [21]), ¶ = SkyRL (a simple ReAct [48] scaffold used in SkyRL-Agent evaluation [7]), ⋆ = Mini-SWE-Agent [46]. Note that R2EGym-Edit-Agent and SkyRL-Agent are not directly available within Harbor's agent suite.

Color coding. Green model names also appear in main Table 1. Orange flags models whose published results we have not been able to fully reproduce (see section G for more detail). Their SWE-Bench-Verified and SWE-Bench-Verified-100 cells are also rendered in orange to flag the gap. <u>Underlined</u> numbers are filled in from the model's reported paper number rather than our reproduction. Note: we do not include in this table open-data releases that did not release a model, such as CoderForge-Preview [3].

Table 24: Qwen 3.5-27B improves slightly on Terminal-Bench 2.1; however, the gap between 2.1 and 2.0 is far smaller than the observed gap for frontier agents. This helps motivate our decision to report Terminal-Bench 2.0 results in the paper.

Benchmark	Accuracy $\pm$ se (%)
Terminal-Bench 2.0	40.1 ± 1.6
Terminal-Bench 2.1	43.1 ± 1.4