DATASENSE-BENCH: THE FIRST STEP TOWARD AN AI SCIENTIST

Anonymous authors
Paper under double-blind review

ABSTRACT

As claims about recursive self-improvement (RSI) and artificial general intelligence (AGI) proliferate, we ask a simple question: do frontier AI models have a sense of data, i.e., can they reliably select the right data for training? We introduce DataSense-Bench to study this capability through the fundamental problem of data selection and performance forecasting in machine learning. We ask AI agents to select and rank candidate training subsets that can be used to fine-tune a small LLM model. Agents are allowed to inspect the data, write and execute analysis code, and run model forward passes, but can not train the model or access the actual evaluation tasks. We then fine-tune the base model on each selected subset and evaluate its post-training performance under a standardized protocol. We instantiate the benchmark in terminal problem solving and tool use, selecting trajectories from OpenThoughts-Agent and EnvScaler and evaluating on TBLite and BFCL, respectively. We then evaluate the agents along two complementary dimensions: the ***post-training performance of the top-ranked subset***, reflecting the ability to identify high-value training data, and ***ranking accuracy***, reflecting the ability to predict the relative performance of the selected subsets. In our experiments, selection gains over random selection are limited; agents do not reliably rank their selected groups, and ranking ability does not hold consistently across tasks: Astra identifies the best group in all three tool-use runs but in only one of three terminal runs. Analysis of execution traces on both tasks shows that agents often use similar data signals while interpreting their training value differently.

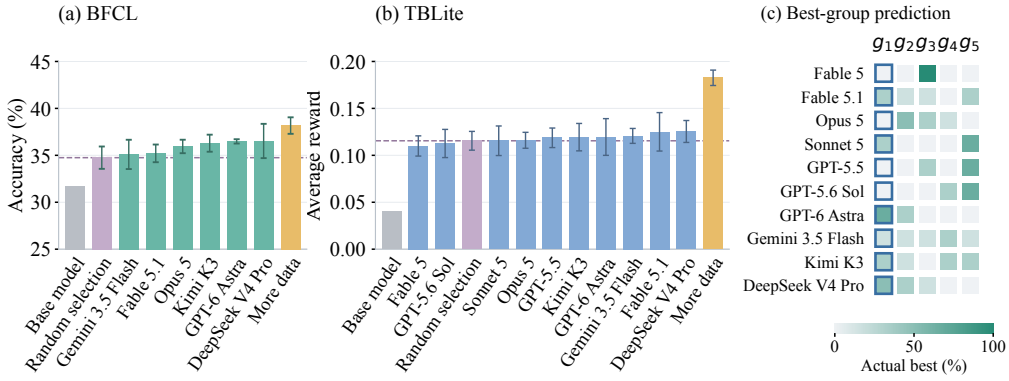

Figure 1: **Can CLI agents select useful training data and predict its value? (a,b)** Qwen3-4B performance after training on the predicted top-ranked group selected by each agent. Panel (a) reports BFCL accuracy; panel (b) reports TBLite reward. Dashed lines denote random selection. Scores are averaged across evaluation seeds within checkpoints, then across three agent runs; error bars show standard deviation across agent runs (across evaluation seeds for random selection). **(c)** Frequency of each predicted rank yielding the best group, pooling evaluated runs from both tasks with equal weight per run: six runs for agents evaluated on both tasks and three for TBLite-only agents. Tied best groups each count as correct; blue outlines mark g_1.

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

1 INTRODUCTION

Large language models (LLMs) are increasingly used for tasks such as question answering (Brown et al., 2020), code generation (Chen et al., 2021), and data analysis (Majumder et al., 2025). Recent advances in reasoning and tool use have made frontier models capable of tackling complex scientific problems, supporting AI scientists who generate research ideas, conduct experiments, analyze results, and write complete research manuscripts (Lu et al., 2026). Benchmarks increasingly assess scientific programming, discovery, and machine learning experimentation (Chen et al., 2025; Majumder et al., 2025; Huang et al., 2024; Rank et al., 2026). These settings test several abilities together. They leave a more focused question open: can an AI system anticipate the value of an experimental choice before observing its outcome?

Training-data selection provides a concrete setting for this question. Before committing to training, researchers must choose which examples will help a particular model learn (Zhou et al., 2023; Xia et al., 2024). This is a consequential and difficult decision: data selection determines the supervision a model receives, yet observable quality does not directly establish training value. For example, low loss alone need not imply high training value: an example may already be well learned (Mindermann et al., 2022). Likewise, a successful trajectory does not by itself establish how much a particular model will learn from it. Reliable judgment requires connecting properties of the data to what a particular model will learn under a fixed training procedure (Li et al., 2024; Xia et al., 2024).

Our Benchmark. Therefore, we introduce DATASENSE-BENCH to evaluate this judgment capability of the frontier models. A CLI agent receives a candidate training data pool, inference-only access to a base model, and a fixed fine-tuning protocol. It selects disjoint, equal-sized subsets and ranks them by expected value for the post-training performance. The agent may inspect data, execute analysis code, and run forward passes, but cannot train models or access evaluation tasks. Once the submission is fixed, the benchmark fine-tunes a fresh copy of the base model on each subset and measures the resulting performance. This separation makes the submitted ranking a prediction that can be checked against subsequent experimental evidence. As a consequence, the benchmark asks about both ***Selection quality*** and ***Ranking accuracy***, which measure whether the first group in the LLM's predicted ordering produces a useful model and whether the predicted ordering of the subsets agrees with their observed performance, separately. A strong top choice need not imply an accurate ordering; conversely, correctly ordering weak subsets need not produce a useful recommendation. Both are needed to understand whether an agent can turn data analysis into reliable decisions.

Our Findings. We instantiate this protocol in terminal problem solving and multi-turn tool use, using OpenThoughts-Agent and EnvScaler trajectories for training and TBLite and BFCL for downstream evaluation, respectively. Our experiments reveal two limitations: gains over random selection are modest, and agents do not reliably rank their selected groups. Ranking ability also does not hold consistently across tasks. **(1) The frontier agents produce plausible, data-specific recipes but do not reliably predict post-training utility.** On TBLite, two LLMs select first groups that underperform random selection, with differences across LLMs ranging from -0.55 to +0.99 points, as shown in Table 1. On BFCL, all six agents' first groups exceed random selection by 0.35–1.78 % points. Astra identifies the best group in all three BFCL runs but in only one of three TBLite runs. None of the six agents evaluated on both tasks identifies the best group in all three runs on each task. **(2) Selection recipes vary substantially across three independent runs of the same agent.** Astra's TBLite runs change their quality-assessment procedures, while Kimi's BFCL runs move from loss-outlier rejection to intermediate-loss and then lower-loss preferences. Repetition therefore changes the selection rule itself, not merely the chosen subset. **(3) Agents draw different conclusions from similar data signals.** For example, agents use base-model loss in substantially different ways across runs and tasks. Section 4.3 examines the concrete recipes in detail.

2 RELATED WORK

We review benchmarks for CLI agents, training-data selection, and performance forecasting.

Benchmarks for CLI Agents. Command-line interface (CLI) agents use shell commands and scripts to complete tasks in OS environments. Terminal-Bench evaluates realistic terminal tasks with executable verification, while SWE-bench focuses on resolving real GitHub issues through repository modifications (Merrill et al., 2026; Jimenez et al., 2024). Research-oriented benchmarks

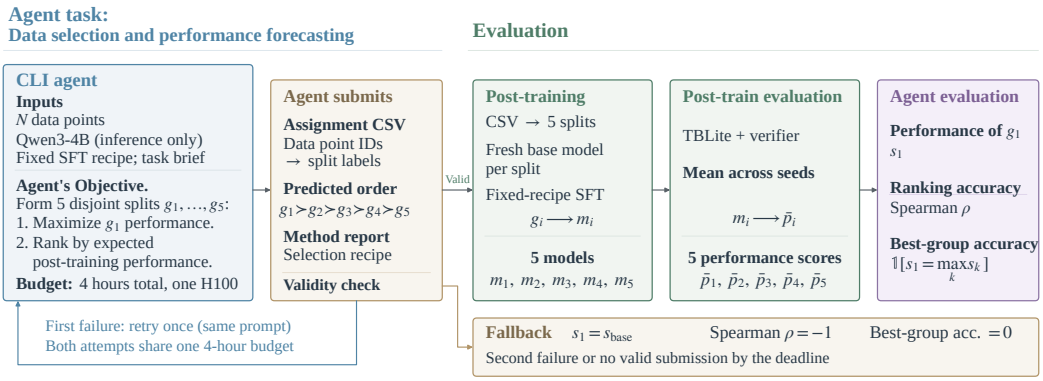

Figure 2: **From data-selection predictions to post-hoc verification.** Illustrated for terminal problem solving, the CLI agent analyzes candidate trajectories with inference-only access to the base model. It delivers a row-to-group assignment CSV and a method report specifying its selection recipe. We validate five disjoint groups of the specified size and freeze their membership and predicted order. Only then does evaluation perform SFT and downstream scoring, testing selection quality, best-group accuracy, and agreement with the full predicted ordering.

address scientific programming and reproducibility (Chen et al., 2025; Siegel et al., 2024), as well as machine learning experimentation and research engineering (Huang et al., 2024; Chan et al., 2025; Wijk et al., 2025). Most closely related, PostTrainBench evaluates agents' choices of data, methods, and hyperparameters for LLM post-training under a compute budget (Rank et al., 2026). These benchmarks measure end-to-end outcomes that depend on both decision quality and execution. DATASENSE-BENCH focuses on data-selection judgment: agents inspect candidate data, construct selection procedures, and submit ranked subsets for a fixed base model and training protocol before receiving any training feedback.

Training Data Selection. Existing selection methods differ in the signals they use to identify useful training data. Curation and scoring approaches emphasize the value of small, high-quality instruction sets (Zhou et al., 2023) and use model-based assessments of quality or instruction-following difficulty to select examples (Chen et al., 2024; Li et al., 2024). Other approaches seek to use importance resampling (Xie et al., 2023b) or improve coverage through deduplication and embedding-based data selection (Tirumala et al., 2023).RHO-LOSS uses a holdout-trained model to prioritize examples expected to reduce generalization loss rather than relying on raw loss (Mindermann et al., 2022). More recently, FEEDER assesses the sufficiency and necessity of demonstrations for in-context learning, with an extension to fine-tuning (Jin et al., 2025). Together, these methods motivate the quality, loss, and coverage signals examined in our procedure analysis.

Performance Forecasting. A complementary line estimates how training examples affect model behavior through derivative-based influence approximations (Koh & Liang, 2017; Pruthi et al., 2020; Kwon et al., 2024), marginal contributions to predictive performance (Ghorbani & Zou, 2019), or gradient similarity for targeted instruction tuning (Xia et al., 2024). At the subset or mixture level, datamodels predict model outputs from training-subset membership (Ilyas et al., 2022), while DoReMi learns domain weights with a proxy model and RegMix predicts mixture performance from small training runs (Xie et al., 2023a; Liu et al., 2025). These approaches differ in their use of target examples, gradients, and training feedback. Rather than proposing another selection method, DATASENSE-BENCH provides a standardized protocol for evaluating whether frontier models can select useful subsets and forecast their relative downstream performance. Subsequent training and evaluation measure both top-subset performance and ranking accuracy.

3 SETUP OF DATASENSE-BENCH

DATASENSE-BENCH aims to evaluate whether a CLI agent can judge the training value of data *before observing training outcomes*. We explain how we set up our benchmark in detail as follows.

3.1 TASK FOR CLI AGENTS: SELECT USEFUL DATA AND PREDICT ITS VALUE

Let D be a pool of N candidate trajectories, M a base model, and R the fine-tuning procedure used to test a submission. The agent selects K subsets $G = (g_1, \ldots, g_K)$ with $g_k \subseteq D$, $|g_k| = n$, and $g_i \cap g_j = \varnothing$ for $i \neq j$. The group labels encode its prediction: training on g_1 should give the highest downstream score, followed by g_2, through g_K. Figure 3 provides a condensed prompt example used in our benchmark, and the full prompt appears in Appendix C.

Post-training Data. We study two tasks that require learning from multi-step interactions: terminal problem solving and multi-turn tool use. Both expose intermediate actions and feedback, allowing agents to assess which trajectories may provide useful supervision. For each task, agents select five disjoint groups of complete trajectories, and we independently fine-tune Qwen3-4B [1] on each group using supervised fine-tuning (SFT). Whole trajectories are selected before the training procedure converts them into training samples.

Tool usage. The training data come from the EnvScaler SFT pool (Song et al., 2026), with validation trajectories excluded. These trajectories involve selecting functions, supplying arguments, and responding to tool feedback. Each selected group contains 50 complete trajectories. We evaluate the fine-tuned models on BFCL V3[2], measuring multi-turn tool-use performance.

Terminal problem solving. The training pool is OpenThoughts-Agent-SFT-10K (Raoof et al., 2026)[3], which contains trajectories pairing task instructions with terminal interactions, including commands, their outputs, and error recovery. Each selected group contains 1,000 trajectories. We evaluate the fine-tuned models on TBLite (Raoof et al., 2026), where automated task verifiers measure success on terminal tasks.

We will explain BFCL V3 and TBLite later, and the agents are not allowed to access these downstream evaluation tasks.

Agent's Objectives. We ask agents to select five groups and specify two objectives, as shown in Figure 3:

- to make g_1 as strong as possible, ideally outperforming random selection at the same size.
- to order the groups by expected post-training performance.

These objectives assess complementary abilities: ranking evaluates the ordering of weak groups, whereas selection evaluates the ability to identify a useful subset.

Resources that the agent can access. The agent receives the task prompt, candidate data, base model, task configuration, and a supervised fine-tuning script. The prompt specifies the downstream task types without revealing evaluation instances, while leaving the choice and combination of selection signals to the agent. Each selection run has inference-only access to the base model and is allocated one H100 80 GB GPU and four hours to analyze the data and submit a selection. Subsequent training and evaluation are performed outside this time budget. Within each task, all agents receive the same fixed training recipe, which is used to fine-tune every selected group. We evaluate ten CLI agents on TBLite and six on BFCL, with three independent selection runs per agent and task. Please refer to Table 3 in Appendix A for more experimental configuration.

Submission Protocol. Each agent is asked to submit a group-assignment CSV and a short method report explaining the selection recipe and its rationale. The CSV assigns valid candidate-pool indices to five disjoint groups: 50 for BFCL and 1,000 complete trajectories per group for TBLite, omitting unselected rows. Group labels g_1 through g_5 indicate the predicted ranking by expected post-training performance, from highest to lowest. The agent must write and verify both files within the selection time budget, finalizing group composition and ordering before observing any training or evaluation outcomes; no trained model is submitted.

[1] `https://huggingface.co/Qwen/Qwen3-4B`

[2] `https://gorilla.cs.berkeley.edu/blogs/13_bfcl_v3_multi_turn.html`

[3] Training data: `open-thoughts/OpenThoughts-Agent-SFT-10K`

Example selection prompt (condensed)

Task. Select training data for Qwen3-4B from a pool of 10,000 terminal trajectories. Read `config.yaml` for the data and model paths, sequence-length limit, and group sizes. Inspect the candidate conversations and `recipe/sft.py` to understand the SFT procedure, including formatting, truncation, and assistant-token supervision.

Objectives. Produce five disjoint groups of 1,000 complete trajectories each. Make g_1 as strong as possible, ideally better than a random subset of the same size, and rank g_1–g_5 by expected post-training performance. Each selected subset will be used to fine-tune a fresh model, whose terminal-task performance will be checked by automated tests.

Resources and limits. Choose your own selection method. You may inspect data, execute analysis code, and run base-model forward passes. You have one H100 80 GB GPU and four hours. Do not train models, update weights, modify or execute the SFT script, or access evaluation tasks and tests.

Deliverables. Submit `outputs/group_assignment.csv` with columns `index,group_id` and `outputs/method.md` explaining your recipe. Check valid row indices, exact group sizes, and no overlap before finishing. Your groups and predicted order must be fixed before seeing training or evaluation outcomes.

Figure 3: Condensed terminal-task prompt specifying the selection and ranking objectives, permitted tools, computation budget, and required deliverables. The prompt for BFCL is described in Appendix C.3.

3.2 Post-check and Post-training Evaluation

After the agent submits its selection, the benchmark validates the submission and tests its predicted ordering, shown in Figure 2.

Submission validation. We check the assignment against the submission protocol in Section 3.1 and verify that the method report is present. For BFCL, we also cross-check row positions against source trajectory identifiers and exclude validation trajectories. CLI completion and valid deliverables are checked separately. If the first attempt fails to produce a valid submission, the agent may retry once using the same prompt within the remaining four-hour selection budget. The two attempts share this budget. If the second attempt also fails, or no valid submission is produced before the budget expires, we apply the failure scores defined in Section 3.3.

Post-training evaluation. Each group is used to fine-tune a fresh copy of the base model under its task-specific training procedure. Within each task, all resulting models use the same evaluation suite and scoring protocol. Let s_k denote the downstream score (TBLite average reward or BFCL accuracy) for the model trained on g_k, averaged across evaluation tasks and evaluation seeds. These outcomes are used only to assess the submitted selection and ordering, not to revise them. Section 4.1 describes both downstream evaluation protocols.

3.3 Evaluation Metrics

We want to evaluate two complementary capabilities of the data selection agents: whether the agent selects useful training data, and whether it can anticipate which data choices will be more useful. We therefore evaluate the performance of its first-ranked group and the agreement between its predicted and observed ordering. **Selection quality.** The score s_1 measures the utility of the data the agent recommends most strongly. To assess whether this choice improves on an uninformed selection, we also report its gain over random selection, $\Delta_{\text{random}} = s_1 - s_{\text{random}}$, using the same group size and training procedure. A positive gain supports the selection objective, but does not establish that the agent can compare its five groups. **Ranking accuracy.** We test that comparison at two levels. Best-group accuracy asks whether the agent's first choice is actually the best among its selected groups: $A_{\text{best}} = \mathbb{1}[s_1 = \max_{k=1,\dots,5} s_k]$. Spearman correlation (Spearman, 1904) tests the full predicted ordering: $S_{\text{rank}} = \rho_{\text{Spearman}}\big((4,3,2,1,0),(s_1,\dots,s_5)\big)$. These measures distinguish identifying the best group from correctly ordering all alternatives. We reverse the group-index scale so that perfect rank agreement gives $+1$, whereas a reversed ordering gives -1.

For valid submissions, we compute both ranking metrics within each selection run after evaluating all five groups. Best-group accuracy counts a tie for the highest unrounded score as correct. Spearman correlation uses average ranks for ties and is undefined when all five outcomes are equal. Such runs are excluded only from the mean Spearman correlation; they remain in the averages of selection scores and best-group accuracy. For failed submissions under Section 3.2, we set $s_1 = s_{\text{base}}$, $S_{\text{rank}} = -1$, and $A_{\text{best}} = 0$. These are failure penalties, not correlations computed from tied base-model scores. Failed runs remain in the averages across selection runs; Δ_{random} uses the fallback s_1.

4 EXPERIMENTS

Our experiments ask whether agents select data that improves downstream performance, whether their predicted group order is reliable, and how they construct these judgments. We report outcomes first, then analyze the selection procedures and their resource requirements.

4.1 DOWNSTREAM EVALUATION

We evaluate the fine-tuned models on the two downstream tasks introduced in Section 3.1. The data-selection agents do not perform these evaluations or access their test instances during the data selection.

TBLite. OpenThoughts-TBLite contains 100 terminal tasks across nine categories, including software engineering, debugging, data processing, and security (Raoof et al., 2026). We use Harbor's[4] Terminus-2 scaffold to execute model-generated commands and return terminal outputs. Each attempt starts in a fresh Daytona[5] sandbox and ends when the model finishes or reaches the task's time limit. Task-supplied verifiers check the resulting environment or artifacts and assign rewards, including fractional credit where supported. We report average reward over 100 tasks with three attempts per task, giving 300 equally weighted scores per checkpoint.

BFCL. For models trained on EnvScaler trajectories, we use BFCL V3: 200 cases each for base interactions, missing functions, missing parameters, and long context. The evaluator checks tool-use outcomes against the required state and execution behavior. Each evaluation seed covers all 800 cases; its overall accuracy averages the four equally sized categories. We use three evaluation seeds per checkpoint.

4.2 SELECTION QUALITY AND RANKING ACCURACY

Table 1 separates two questions: whether agents choose useful training data and whether they correctly rank the groups they select.

Selection yields limited gains over random selection on both tasks. On BFCL, all six agents' average g_1 accuracies exceed random selection's 34.75%, but gains range from only 0.35 to 1.78 % points. DeepSeek V4 Pro has the highest average at 36.53%. Its run-3 g_1 achieves the highest accuracy among agent-selected checkpoints, 38.62%. On TBLite, average g_1 rewards range from 11.00% to 12.54%, compared with 11.55% for random selection. Two of ten agents underperform this reference, and DeepSeek V4 Pro's leading score improves on it by only 0.99 points. Thus, elaborate selection recipes yield modest gains on both tasks, with less consistent improvements on TBLite.

Selecting useful data does not imply reliably ranking the selected groups. BFCL reveals a gap between selection quality and ranking accuracy: all six agents improve over Random with g_1, yet their average Spearman correlations range from -0.27 to 0.83. GPT-6 Astra identifies the best group in all three runs and achieves a correlation of 0.83. In contrast, Kimi K3 achieves 36.29% average g_1 accuracy but a correlation of only 0.02, while Gemini's negative correlation indicates disagreement with the predicted ordering. Conversely, Opus has a correlation of 0.77 but never identifies the best group, illustrating that agreement over all five positions can coexist with an incorrect first choice. The difficulty is more widespread on TBLite, where correlations range from -0.50 to 0.30. Even

[4] https://www.harborframework.com/

[5] https://www.daytona.io/docs/en/

Table 1: Selection performance, best-group accuracy, and rank performance on BFCL and TBLite. Agent-group scores (s_1-s_5) report the post-training performance, accuracy (%) for BFCL, and reward for TBLite (scale from 0-1 to 0-100). We report the average performance across three evaluation seeds and three data selection runs, standard deviation (subscript), and maximum score among different data selection runs. Shading compares group averages within a row; red outlines mark each task's largest maximum among agent-selected groups. g_1 gain Δ_{random} is relative to random selection. Best-group accuracy A_{best} measures how often the agent identifies its best group; rank performance S_{rank} measures Spearman correlation between predicted and observed group orderings (Section 3.3). More detailed experimental results are in Appendix A.3.

Rank	Agent / reference	s_1	g_1 gain Δ_{random}	s_2	s_3	s_4	s_5	S_{rank}	A_{best} (%)
	BFCL accuracy (50 trajectories per group) Base model: 31.67%								
	More data (9,022)	$38.17_{\pm 0.88}$	–		–			–	–
1	DeepSeek V4 Pro	$36.53_{\pm 1.83}$ Max: 38.62	1.78	$\mathbf{36.61}_{\pm 0.72}$ Max: 37.42	$35.74_{\pm 1.12}$ Max: 37.00	$36.08_{\pm 1.44}$ Max: 37.75	$34.33_{\pm 1.16}$ Max: 35.67	0.65	33.33
2	GPT-6 Astra	$\mathbf{36.49}_{\pm 0.23}$ Max: 36.75	1.74	$35.31_{\pm 0.94}$ Max: 36.38	$34.74_{\pm 1.50}$ Max: 36.21	$34.93_{\pm 0.42}$ Max: 35.42	$34.58_{\pm 0.07}$ Max: 34.62	0.83	100.00
3	Kimi K3	$36.29_{\pm 0.91}$ Max: 36.92	1.54	$36.21_{\pm 0.65}$ Max: 36.92	$36.18_{\pm 1.22}$ Max: 37.58	$35.85_{\pm 1.14}$ Max: 36.62	$\mathbf{36.46}_{\pm 1.27}$ Max: 37.75	0.02	33.33
4	Opus 5	$35.94_{\pm 0.72}$ Max: 36.75	1.19	$\mathbf{36.35}_{\pm 1.09}$ Max: 37.38	$35.35_{\pm 0.53}$ Max: 35.96	$34.29_{\pm 2.29}$ Max: 36.54	$31.08_{\pm 3.45}$ Max: 35.04	0.77	0.00
5	Fable 5.1	$35.21_{\pm 0.94}$ Max: 36.00	0.46	$34.96_{\pm 1.67}$ Max: 36.58	$34.87_{\pm 0.07}$ Max: 34.96	$34.10_{\pm 0.24}$ Max: 34.38	$\mathbf{35.57}_{\pm 1.53}$ Max: 37.17	0.23	33.33
6	Gemini 3.5 Flash	$35.10_{\pm 1.56}$ Max: 36.42	0.35	$34.18_{\pm 1.10}$ Max: 35.29	$33.57_{\pm 0.49}$ Max: 34.12	$33.07_{\pm 3.35}$ Max: 35.21	$\mathbf{35.60}_{\pm 0.94}$ Max: 36.33	-0.27	33.33
	Random selection	$34.75_{\pm 1.19}$	0.00		–			–	–
	TBLite average reward (1,000 trajectories per group) Base model: 4.05%								
	More data (31.6K)	$18.26_{\pm 0.82}$	–		–			–	–
1	DeepSeek V4 Pro	$\mathbf{12.54}_{\pm 1.17}$ Max: 13.46	0.99	$11.75_{\pm 1.76}$ Max: 13.26	$11.65_{\pm 1.78}$ Max: 13.69	$12.19_{\pm 0.25}$ Max: 12.47	$12.11_{\pm 1.88}$ Max: 13.21	0.13	66.67
2	Fable 5.1	$\mathbf{12.50}_{\pm 2.05}$ Max: 14.83	0.95	$11.20_{\pm 0.98}$ Max: 12.24	$12.01_{\pm 0.92}$ Max: 12.87	$10.22_{\pm 0.84}$ Max: 11.18	$11.62_{\pm 1.57}$ Max: 13.22	0.30	33.33
3	Gemini 3.5 Flash	$12.07_{\pm 0.79}$ Max: 12.98	0.52	$\mathbf{13.42}_{\pm 0.66}$ Max: 13.93	$11.82_{\pm 1.56}$ Max: 13.25	$12.58_{\pm 0.98}$ Max: 13.70	$10.96_{\pm 2.69}$ Max: 13.55	0.30	0.00
4	GPT-6 Astra	$11.95_{\pm 1.96}$ Max: 14.18	0.40	$\mathbf{13.35}_{\pm 1.86}$ Max: 14.52	$10.60_{\pm 0.81}$ Max: 11.31	$11.60_{\pm 1.88}$ Max: 13.56	$10.44_{\pm 2.29}$ Max: 12.11	0.17	33.33
5	Kimi K3	$11.94_{\pm 1.46}$ Max: 13.41	0.38	$11.02_{\pm 2.00}$ Max: 13.09	$11.34_{\pm 0.63}$ Max: 12.04	$\mathbf{13.05}_{\pm 2.09}$ Max: 15.16	$10.81_{\pm 0.57}$ Max: 11.42	0.07	33.33
6	GPT-5.5	$11.87_{\pm 1.04}$ Max: 12.80	0.31	$9.81_{\pm 1.10}$ Max: 10.56	$11.70_{\pm 1.39}$ Max: 13.31	$11.54_{\pm 1.49}$ Max: 12.74	$\mathbf{12.45}_{\pm 0.25}$ Max: 12.64	-0.23	0.00
7	Opus 5	$11.60_{\pm 0.85}$ Max: 12.51	0.05	$\mathbf{12.50}_{\pm 1.33}$ Max: 13.73	$11.94_{\pm 2.05}$ Max: 13.70	$10.94_{\pm 2.26}$ Max: 13.53	$10.35_{\pm 1.99}$ Max: 11.71	0.30	0.00
8	Sonnet 5	$11.55_{\pm 1.57}$ Max: 13.00	0.00	$11.95_{\pm 0.78}$ Max: 12.48	$10.67_{\pm 0.79}$ Max: 11.57	$11.82_{\pm 0.57}$ Max: 12.43	$\mathbf{12.94}_{\pm 1.77}$ Max: 14.92	-0.20	33.33
	Random selection	$11.55_{\pm 1.04}$	0.00		–			–	–
9	GPT-5.6 Sol	$11.26_{\pm 1.50}$ Max: 12.96	-0.29	$11.22_{\pm 0.50}$ Max: 11.77	$11.39_{\pm 1.12}$ Max: 12.63	$\mathbf{12.90}_{\pm 0.50}$ Max: 13.29	$12.75_{\pm 2.50}$ Max: 14.78	-0.50	0.00
10	Fable 5	$11.00_{\pm 1.08}$ Max: 12.14	-0.55	$10.53_{\pm 1.86}$ Max: 12.08	$\mathbf{13.40}_{\pm 1.20}$ Max: 14.52	$11.50_{\pm 1.74}$ Max: 13.44	$10.92_{\pm 1.14}$ Max: 12.14	-0.03	0.00

DeepSeek V4 Pro, the strongest agent by average g_1 reward, achieves only 0.13. Kimi K3 places the highest-performing agent-selected subset fourth: its run-3 g_4 reaches 15.16% average reward. Together, the two tasks show that gains over Random do not establish an ability to compare alternative data choices; no agent identifies the best group in all three runs on both tasks. Astra, for example, does so in three of three BFCL runs but only one of three TBLite runs. Appendix Tables 7 and 8 report the individual selection runs.

4.3 How Agents Make Selection Decisions

We analyze 30 TBLite and 18 BFCL selection procedures and identify three patterns. **Common signals, different value judgments.** Across TBLite recipes, completion cues appear in 30 procedures, length features in 29, format checks in 28, base-model loss in 23, and diversity controls in 21. Yet agents interpret these signals differently. One DeepSeek V4 Pro recipe rewards higher first-assistant-

Time	Executed commands and resulting decisions
00:00:02	`cat config.yaml`; `cat recipe/sft.py`; `nvidia-smi`. Read the selection configuration and SFT recipe; inspect available compute.
00:00:49–00:03:13	`python - <<'EOF'`: read `data/train.parquet`, print example turns and task descriptions, and count completion markers, parser warnings, and response formats.
00:04:04	Write and run `python work/features.py`. Extract structural features and token counts into `features.parquet`; save task descriptions in `tasks.parquet`.
00:06:44–00:08:43	Write `work/score_gpu.py`; pilot with `timeout 900 python work/score_test.py`. Revise scoring priority, then launch `python work/score_gpu.py > work/gpu_log.txt 2>&1`: frozen-model assistant-token loss, up to 32,768 tokens.
00:10:24	Inline Python computes task duplicates and TF-IDF similarity, fits `MiniBatchKMeans(n_clusters=50)`, and saves `work/features2.parquet`.
00:12:01–00:14:35	Write and repeatedly run `python work/select.py`. Inspect the provisional groups; replace fixed cluster caps with pool-relative caps, add an 8% Tezos cap, then add source caps. These changes precede downstream evaluation.
00:40:43	Inline Python reads `work/gpu_scores.pkl` and checks loss distributions, source-level statistics, correlations, and extreme-loss examples while GPU scoring continues.
02:04:37	Rerun `python work/select.py`; inspect the CSV with `head -3` and `wc -l`. Inline assertions check column names, unique in-range indices, group IDs, and exactly 1,000 rows per group.
02:05:18	`cat >> outputs/method.md <<'EOF'`: append final group statistics and compute details; list the delivered files in `outputs/`.

Final recipe. *Score.* Combine completion (+2), usable assistant-token supervision, command substance, and writing/testing cues with soft penalties for format faults, context handoff, truncation, empty or repetitive actions, and narrow domain coverage. Add loss-outlier penalties for both unusually high and low average loss, high-loss token fractions, and extreme last/max-turn loss; loss is not a monotonic preference for harder or easier examples. *Construct groups.* Sort by composite score. For g_1, greedily take 1,000 rows with one trajectory per task and cap cluster c at $\max(12, \lceil 0.20 n_c \rceil)$, where n_c is its pool size. Repeat on unassigned rows for g_2, using $0.25 n_c$. Both groups cap software-engineering, synthetic, and other sources at 40%, 45%, and 35%, respectively, and Tezos at 8%. Assign the next two highest-scoring blocks to g_3 and g_4 without these diversity caps; assign the lowest-scoring remaining 1,000 rows to g_5. Deliver five disjoint groups in `group_assignment.csv` and explain the procedure in `method.md`.

Figure 4: Fable 5.1's data-selection workflow, from inspection to final recipe.

turn loss after penalizing formatting defects: $q_{\text{DS}} = \ell(1 - 0.10e - 0.05w - 0.50j)(0.5 + 0.5c)$, where e, w, j are extra-text, warning, and invalid-JSON rates, and c indicates completion. Kimi K3 instead favors lower length-normalized loss: $q_{\text{Kimi}} = -z_{\text{length}}(\ell) + 1.2c - 2t$, where loss is standardized within length deciles and t indicates truncation. Fable 5.1 uses several loss statistics as anomaly penalties on a structural score, with task coverage and source caps. These are different judgments of training value, although their loss measurements also differ in coverage and normalization. **Recipes change across independent runs.** Repeated selections by the same agent can change the rule used to judge data, rather than merely produce different subsets under a fixed rule. For example, Astra's three TBLite runs assess quality through generated correctness audits, separate prompted judgments of demonstration quality, task accomplishment and complexity, and a five-level rating with execution-evidence checks, respectively. The variation also occurs in BFCL: Kimi moves from rejecting loss outliers to preferring intermediate loss and then lower loss. Thus, a single run's recipe does not fully characterize an agent's selection strategy. **Agents construct the predicted ordering in different ways.** The ordering is part of the selection procedure: agents can partition a scored candidate list or apply different selection criteria to different groups. Astra's third TBLite run combines these approaches, constructing g_1 with deduplication and skill coverage before drawing the remaining groups from descending score bands. In BFCL, Fable 5.1's first run places its lowest-scoring candidates in g_5, whereas Gemini's second run explicitly assigns looping and error-heavy trajectories to that group. These procedures encode the agents' predictions of relative training value; Section 4.2 evaluates how well those predictions match post-training performance.

4.4 Case Study

Figure 4 shows Fable 5.1's data selection for the terminal task, including data inspection and the final recipe. The agent first builds structural features and base-model loss statistics, then revises its grouping rules after inspecting the resulting subsets. Its final recipe uses loss to penalize outliers rather than uniformly favoring high- or low-loss examples. It also applies task, cluster, and source constraints to g_1 and g_2, while constructing later groups from score bands without those diversity caps. This case shows that the submitted ordering combines example-level scores with group-level coverage choices, rather than following a single score throughout. In this run, g_1 achieves 14.83% reward, but g_2 (11.08%) falls below g_3 (12.12%). The best first choice therefore coexists with an error in the remaining order. Appendix B.1 and Table 6 provide the remaining recipes.

4.5 Resource Requirements

The benchmark separates the cost of making a data recommendation from the cost of testing it through SFT and downstream evaluation.

CLI agent cost and time budget. Table 2 summarizes the agent API cost for data selection. Selection estimates use cumulative token ledgers and exclude local inference, training, and evaluation charges. Figure 5 shows time cost against the four-hour time budget, including tool execution and waits. Measurement details and pricing assumptions are in Appendix A.5.

Table 2: Mean selection cost (USD; ranges reflect billing assumptions) and training/evaluation hours per checkpoint (medians; TBLite evaluation estimated).

Agent	TBLite	BFCL	Agent	TBLite	BFCL
Fable 5.1	5.98	5.35	Opus 5	3.69	4.91
GPT-6 Astra	11.11	$\geq$9.35	Kimi K3	2.41	1.38
DeepSeek	0.28–0.57	0.38–0.78	Gemini	1.11–1.34	0.84–1.01
Fable 5	5.10	–	Sonnet 5	3.63	–
GPT-5.5	2.54	–	GPT-5.6 Sol	0.70	–

Training / evaluation (h): TBLite 2.04 / $\approx$3.93; BFCL 0.29 / 3.30.

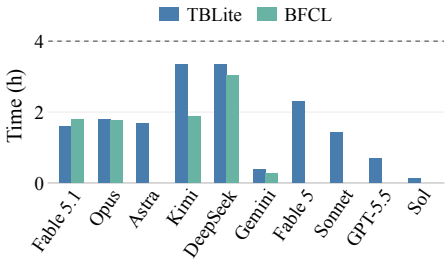

Figure 5: **Time cost in data selection.** The dashed line marks the time budget.

Training and evaluation time. Evaluation is the larger time component on both tasks (Table 2). Each selection requires five trained checkpoints. TBLite evaluation time is a scheduling estimate for 300 attempts with 32 concurrent trials on two H100 GPUs, including model loading; it is not a measured end-to-end runtime. BFCL medians cover 90 training runs and 35 complete checkpoint evaluations, with all three evaluation seeds sharing one model server on one GPU.

5 Conclusion

Anticipating which changes will improve a model can help recursive self-improvement allocate training resources more effectively. As these systems take on research roles, data sense becomes a concrete part of that challenge: connecting properties of training examples to what a model will learn from them. We introduce DataSense-Bench to evaluate this data sense through selecting useful subsets and predicting their relative post-training performance. Across terminal problem solving and multi-turn tool use, agents construct plausible, data-specific recipes, but selection gains over random selection are limited. Agents do not reliably rank their selected groups, and this ability does not hold consistently across tasks. These results highlight a gap between producing a convincing analysis of data and making reliable decisions about its training value, motivating data sense as a capability to evaluate when assessing progress toward AI scientists and their recursive self-improvement.

Limitations and future work. Our benchmark studies data sense through selecting and ranking existing data for SFT. Future work should extend this scope to reinforcement-learning post-training and the generation of new training data, testing whether agents can anticipate which experiences to collect or create and how those choices will improve learning.

AI USE STATEMENT

Generative AI tools, including OpenAI Codex, assist in conducting experiments, including drafting data-selection prompts, code snapshots, monitoring experiments, and also assist in polishing the manuscript. Frontier models also act as experimental participants, as described in Section 3.1. Numerical results are computed from the recorded selections and downstream evaluations. The authors are responsible for the final text, methods, results, and claims.

BROADER IMPACT

As AI systems increasingly influence how models are trained, their data choices can shape whose needs and experiences those models serve. DATASENSE-BENCH makes these choices testable, supporting closer scrutiny of claims about autonomous AI research and recursive self-improvement. Better data selection could reduce wasted training compute, but optimizing benchmark performance alone may favor narrow capabilities or amplify biases in the candidate data. Strong results should therefore be considered alongside data provenance, privacy, and representativeness. Repeatedly selecting data for aggregate performance could exclude rare but important cases. Human review of these choices remains necessary, especially when the resulting models are used in settings where errors affect people. We hope this work supports evidence-based oversight of AI research agents.

REFERENCES

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Aleksander Madry, and Lilian Weng. Mle-bench: Evaluating machine learning agents on machine learning engineering. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 50466–50494, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/7e3767db483c942b883eb4f8cfb74e31-Paper-Conference.pdf.

Lichang Chen, Shiyang Li, Jun Yan, Hai Wang, Kalpa Gunaratna, Vikas Yadav, Zheng Tang, Vijay Srinivasan, Tianyi Zhou, Heng Huang, and Hongxia Jin. Alpagasus: Training a better alpaca with fewer data. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 34767–34797, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/9543942c237ded1b39b1fd37259ff88e-Paper-Conference.pdf.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé, Jared Kaplan, Harrison Edwards, Yura Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mo Bavarian, Clemens S. Winter, Phil Tillet, Felipe Petroski Such, David W. Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Igor Babuschkin, Suchir Balaji, Shantanu Jain, Andrew Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew M. Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *ArXiv*, abs/2107.03374, 2021. URL https://api.semanticscholar.org/CorpusID:235755472.

Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, Vishal Dey, Mingyi Xue, Frazier N. Baker, Benjamin Burns, Daniel Adu-Ampratwum, Xuhui Huang, Xia Ning, Song Gao, Yu Su, and Huan Sun. Scienceagentbench: Toward rigorous assessment of language agents for data-driven scientific discovery. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 96934–96990, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/f12b4df26344f3be803c06b555252efe-Paper-Conference.pdf.

Amirata Ghorbani and James Zou. Data shapley: Equitable valuation of data for machine learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2242–2251. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/ghorbani19c.html.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. MLAgentBench: Evaluating language agents on machine learning experimentation. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 20271–20309. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/huang24y.html.

Andrew Ilyas, Sung Min Park, Logan Engstrom, Guillaume Leclerc, and Aleksander Madry. Datamodels: Understanding predictions with data and data with predictions. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 9525–9587. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/ilyas22a.html.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/edac78c3e300629acfe6cbe9ca88fb84-Paper-Conference.pdf.

Jiarui Jin, Yuwei Wu, Haoxuan Li, Xiaoting He, Weinan Zhang, Yiming Yang, Yong Yu, Jun Wang, and Mengyue Yang. Large language models are demonstration pre-selectors for themselves. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 28157–28186. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/jin25i.html.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1885–1894. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/koh17a.html.

Yongchan Kwon, Eric Wu, Kevin Wu, and James Y Zou. Datainf: Efficiently estimating data influence in lora-tuned llms and diffusion models. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 21921–21942, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/5e84a0f233611a1dc8fb794dc52415a3-Paper-Conference.pdf.

Ming Li, Yong Zhang, Zhitao Li, Jiuhai Chen, Lichang Chen, Ning Cheng, Jianzong Wang, Tianyi Zhou, and Jing Xiao. From quantity to quality: Boosting LLM performance with self-guided data selection for instruction tuning. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 7602–7635,

Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.421. URL https://aclanthology.org/2024.naacl-long.421/.

Qian Liu, Xiaosen Zheng, Niklas Muennighoff, Guangtao Zeng, Longxu Dou, Tianyu Pang, Jing Jiang, and Min Lin. Regmix: Data mixture as regression for language model pre-training. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=5BjQOUXq7i.

Chris Lu, Cong Lu, Robert Tjarko Lange, Yutaro Yamada, Shengran Hu, Jakob Foerster, David Ha, and Jeff Clune. Towards end-to-end automation of AI research. *Nature*, 651:914–919, 2026. doi: 10.1038/s41586-026-10265-5. URL https://www.nature.com/articles/s41586-026-10265-5.

Bodhisattwa Prasad Majumder, Harshit Surana, Dhruv Agarwal, Bhavana Dalvi Mishra, Abhijeetsingh Meena, Aryan Prakhar, Tirth Vora, Tushar Khot, Ashish Sabharwal, and Peter Clark. Discoverybench: Towards data-driven discovery with large language models. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 4556–4579, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/0d70af566e69f1dfb687791ecf955e28-Paper-Conference.pdf.

Mike Merrill, Alexander Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, 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 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 (Ryan) 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, Christopher Rytting, Ryan Marten, Yixin Wang, Jenia Jitsev, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 40903–40986, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/444a3737adaee10d86ad2ef5f74468e6-Paper-Conference.pdf.

Sören Mindermann, Jan M Brauner, Muhammed T Razzak, Mrinank Sharma, Andreas Kirsch, Winnie Xu, Benedikt Höltgen, Aidan N Gomez, Adrien Morisot, Sebastian Farquhar, and Yarin Gal. Prioritized training on points that are learnable, worth learning, and not yet learnt. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 15630–15649. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/mindermann22a.html.

Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 19920–19930. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/e6385d39ec9394f2f3a354d9d2b88eec-Paper.pdf.

Ben Rank, Hardik Bhatnagar, Ameya Prabhu, Shira Eisenberg, Karina Nguyen, Matthias Bethge, and Maksym Andriushchenko. Posttrainbench: Can LLM agents automate LLM post-training? In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=UnjxMTe57e.

Negin Raoof, Richard Zhuang, Marianna Nezhurina, Etash Guha, Atula Tejaswi, Ryan Marten, Charlie F. Ruan, Tyler Griggs, Alexander Glenn Shaw, Hritik Bansal, E. Kelly Buchanan, Artem

Gazizov, Reinhard Heckel, Chinmay Hegde, Sankalp Jajee, Daanish Khazi, Emmanouil Koukoumidis, Xiangyi Li, Hange Liu, Shlok Natarajan, Harsh Raj, Nicholas Roberts, Ethan Shen, Nishad Singhi, Michael Siu, Ashima Suvarna, Hanwen Xing, Patrick Yubeaton, Robert Zhang, Leon Liangyu Chen, Xiaokun Chen, Steven Dillmann, Saadia Gabriel, Xunyi Jiang, Anurag Kashyap, Boxuan Li, Yein Park, Minh Pham, Sujay Sanghavi, Lin Shi, Ke Sun, Yixin Wang, Zhiwei Xu, Erica Zhang, Siyan Zhao, Wanjia Zhao, Jenia Jitsev, Alex Dimakis, Benjamin Feuer, and Ludwig Schmidt. Openthoughts-agent: Data recipes for agentic models, 2026. URL https://arxiv.org/abs/2606.24855.

Zachary S Siegel, Sayash Kapoor, Nitya Nadgir, Benedikt Stroebl, and Arvind Narayanan. CORE-bench: Fostering the credibility of published research through a computational reproducibility agent benchmark. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=BsMMc4MEGS.

Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. EnvScaler: Scaling tool-interactive environments for LLM agent via programmatic synthesis. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 8326–8357, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.407. URL https://aclanthology.org/2026.findings-acl.407/.

C. Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101, 1904. doi: 10.2307/1412159. URL https://www.jstor.org/stable/1412159.

Kushal Tirumala, Daniel Simig, Armen Aghajanyan, and Ari Morcos. D4: Improving llm pretraining via document de-duplication and diversification. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 53983–53995. Curran Associates, Inc., 2023. doi: 10.52202/075280-2348. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/a8f8cbd7f7a5fb2c837e578c75e5b615-Paper-Datasets_and_Benchmarks.pdf.

Hjalmar Wijk, Tao Roa Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Joshua M Clymer, Jai Dhyani, Elena Ericheva, Katharyn Garcia, Brian Goodrich, Nikola Jurkovic, Megan Kinniment, Aron Lajko, Seraphina Nix, Lucas Jun Koba Sato, William Saunders, Maksym Taran, Ben West, and Elizabeth Barnes. RE-bench: Evaluating frontier AI r&d capabilities of language model agents against human experts. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 66772–66832. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/wijk25a.html.

Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. LESS: Selecting influential data for targeted instruction tuning. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 54104–54132. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/xia24c.html.

Sang Michael Xie, Hieu Pham, Xuanyi Dong, Nan Du, Hanxiao Liu, Yifeng Lu, Percy S Liang, Quoc V Le, Tengyu Ma, and Adams Wei Yu. Doremi: Optimizing data mixtures speeds up language model pretraining. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 69798–69818. Curran Associates, Inc., 2023a. doi: 10.52202/075280-3059. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/dcba6be91359358c2355cd920da3fcbd-Paper-Conference.pdf.

Sang Michael Xie, Shibani Santurkar, Tengyu Ma, and Percy S Liang. Data selection for language models via importance resampling. In A. Oh, T. Naumann, A. Globerson,

K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 34201–34227. Curran Associates, Inc., 2023b. doi: 10.52202/075280-1482. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/6b9aa8f418bde2840d5f4ab7a02f663b-Paper-Conference.pdf.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, LILI YU, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. Lima: Less is more for alignment. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 55006–55021. Curran Associates, Inc., 2023. doi: 10.52202/075280-2400. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/ac662d74829e4407ce1d126477f4a03a-Paper-Conference.pdf.

A Experimental Details

A.1 Training Configuration

Table 3 lists the configurations for both tasks. We use full-parameter SFT with assistant-token supervision; TBLite supervision includes thinking, and its gradient-clipping threshold is 10^{-3}. Selected BFCL trajectories are converted into assistant-supervised examples before training. Within each task, all agent-selected groups share the realized training configuration.

Table 3: Configuration of the two selection-and-ranking tasks. Group sizes count complete trajectories.

	Terminal problem solving	Multi-turn tool use
Training pool	OpenThoughts-Agent 10K	EnvScaler SFT
Groups × group size	$5 \times 1,000$	5×50
Base model	Qwen3-4B	Qwen3-4B
Selectors / runs per selector	10 / 3	6 / 3
Selection GPU / time budget	H100 80GB / 4 h	H100 80GB / 4 h
Learning rate / epochs	4×10^{-5} / 7	5×10^{-6} / 2
Effective batch / training seed	96 / 42	32 / 42
Training token cutoff	32,768	16,384
Downstream evaluation	TBLite: 100 tasks	BFCL V3: 800 cases
Main outcome	Average reward	Overall accuracy
Evaluation seeds	3	3

More-data references. The More data baselines use the respective official training recipes: the OpenThoughts-Agent recipe for terminal data and the published EnvScaler SFT checkpoint for tool use.

A.2 Selection Models and Inference Configuration

Table 4 maps the names in the results to the model identifiers used by the selection agents. Claude Code session headers record version 2.1.270; the retained deployment packages are Codex CLI 0.146.1 and Gemini CLI 0.60.0.

Table 4: Selection-agent model identifiers.

Agent	Model identifier
Fable 5.1	`claude-fable-5-1`
Opus 5	`claude-opus-5`
GPT-6 Astra	`gpt-6-astra`
Kimi K3	`kimi-k3`
DeepSeek V4 Pro	`deepseek-v4-pro`
Gemini 3.5 Flash	`gemini-3.5-flash`
Fable 5	`claude-fable-5`
Sonnet 5	`claude-sonnet-5`
GPT-5.5	`gpt-5.5`
GPT-5.6 Sol	`gpt-5.6-sol`

All selection CLIs run non-interactively with automatic tool approval. Claude Code uses streaming JSON output and summarized thinking display; Kimi and DeepSeek use their Anthropic-compatible endpoints through the same CLI. Codex enables web search and detailed reasoning summaries. The wrappers do not set a selection temperature, sampling seed, or reasoning-effort override; these use the CLI/provider defaults. Summary-display options affect the recorded trace and do not specify a reasoning-token budget. Gemini uses streaming JSON output; its TBLite token ledgers identify `gemini-3.5-flash` as the served model and `gemini-3-flash-preview` as a helper, although the original request used the alias `gemini-3.8-flash`.

A.3 Downstream Evaluation and Result Aggregation

For TBLite, the model proposes terminal commands at each step; Terminus-2 executes them and returns the terminal output for the model's next step. Harbor manages the task environment and runs the verifier after the attempt. The conversation retains the model's reasoning between command executions. A vLLM server runs the evaluated checkpoint. The configured temperature is 0.6, top-p is 0.95, and top-k is 20. The harness model limits are 32,768 input tokens, 40,960 total tokens, and 7,168 output tokens. These are model-call limits, not a total token budget for an entire task. The task-specific agent time limits range from 300 to 3,600 seconds; the agent timeout multiplier is one. Deployments use tensor parallelism 2.

BFCL evaluation uses a 65,536-token context and temperature 0.7. Each of the three evaluation seeds covers the full 800-case BFCL V3 multi-turn suite.

We average evaluation scores within each checkpoint before computing run-level metrics, then average these metrics across independent selection runs. Group-performance standard deviations measure variation across selection runs. TBLite checkpoint scores include all 300 task attempts. Ranking metrics compare all five groups within a valid selection run. Failed submissions receive the fallback scores defined in Section 3.3 and remain in the run-level averages. Random selection uses one fixed subset per task, so its repeated evaluations measure evaluation variability rather than variation across data selections. Full selection-level results appear in Appendix B.3.

A.4 Selection-Time Checks and Recipe Information

A Python guard disables common training entry points, including `Tensor.backward` and `Trainer.train`, while allowing forward passes. Model-file metadata and training-related trace terms support further review. Metadata are not weight-content hashes, and keyword matches can arise from reading recipes; these checks alone do not certify that every access or training restriction was enforced. We retain available CLI/tool traces, analysis scripts, intermediate summaries, final assignments, method reports, and validation outputs. Timestamps, commands, outputs, and exit status establish the execution sequence against which agent-written claims are checked. Appendix B.1 summarizes selection procedures and differences in record coverage.

Within each task, agents receive the same fixed training recipe, and all selected groups are fine-tuned using that recipe. The training procedure does not vary across agents or selection runs.

A.5 Cost and Runtime Measurement

Selection costs in Table 2 are averaged across three runs per agent and task. The BFCL entries use the recorded model-token ledgers and invocation durations for the 18 selections. Multiple recorded invocations belonging to the same selection are combined before averaging the three runs. Durations include tool execution and waits within those invocations, but exclude gaps between invocations and scheduler queueing. Astra lacks complete duration and usage coverage; its cost is therefore reported as a lower bound and its duration is omitted. Token estimates use the same fixed rate schedule as the TBLite comparison, rather than subscription payments or invoices. GPU inference, SFT, and evaluation costs are excluded. DeepSeek ranges vary peak/off-peak rates; Gemini ranges vary the billing treatment of unclassified output tokens. For reproducibility, the fixed USD-per-million-token rates for fresh input, cache reads, cache writes, and output are $(10, 0.25, 12.5, 50)$ for Fable 5.1, $(5, 0.5, 6.25, 25)$ for Opus 5, $(10, 1, 12.5, 50)$ for Astra, and $(3, 0.3, 3, 15)$ for Kimi K3. One-hour Claude cache writes use twice the fresh-input rate. DeepSeek uses $(0.66, 0.022, 0, 1.98)$ to twice those rates. Gemini Flash uses $(1.5, 0.15, 0, 9)$, and its separately logged helper model uses $(0.5, 0.05, 0, 3)$. These are comparison assumptions, not measured charges.

Training times in Table 2 exclude queueing and model loading. The TBLite sample comprises 16 completed four-GPU jobs, with a median of 2.04 h and a range of 1.38–3.30 h. The BFCL sample comprises 90 completed 50-trajectory runs, with a median of 0.29 h and an interquartile range of 0.25–0.38 h. BFCL evaluation timing covers 35 completed single-GPU jobs, each evaluating one checkpoint across all three evaluation seeds through a shared model server. The median is 3.30 h, including model startup and excluding queueing.

B Detailed Experimental Results

B.1 Selection Procedures

The TBLite summaries below distinguish signals used in the final method from the rule that allocates groups. Runs 1–3 identify each agent's three reported selection attempts in order; they are not training seeds. NLL denotes base-model negative log-likelihood on assistant tokens. Completion and verification features are proxies extracted from trajectories, not access to downstream evaluation outcomes. "Consecutive blocks" means partitioning the selected score order into five equal groups; "bands" means selecting separated portions of a ranking. Each row describes one selection attempt, with its signals and final grouping rule. Full execution records and scripts are retained separately from these summaries.

Figure 4 illustrates one Terminal procedure in detail. Each group contains 1,000 trajectories. The descriptions below follow executed selection code rather than unimplemented claims in method reports.

Table 5: TBLite selection methods, with one row per attempt.

Agent	Run	Signals used	Final grouping rule
Fable 5	1	Structure, NLL tails and 64 task clusters.	Cluster-balanced g_1; spaced lower score bands.
	2	Structure and partial NLL coverage.	Task deduplication and cluster caps in g_1.
	3	Structure and 8K-prefix NLL.	Task-deduplicated g_1 without clustering; lower score windows.
Fable 5.1	1	Structural quality with a small JSON-NLL term.	Task deduplication and cluster/style caps in g_1/g_2; lower score bands.
	2	Structural quality, model loss and a prompted concreteness score.	Task deduplication and cluster/style caps in g_1/g_2; lower score bands.
	3	Structural quality with soft loss-anomaly penalties.	Cluster/source caps and an 8% Tezos cap in g_1/g_2; next-ranked g_3/g_4 and bottom g_5.
Opus 5	1	Completion, protocol and verification features plus model statistics.	g_1 eligibility checks, task deduplication and cluster caps; lower score bands.
	2	Quality, NLL and embeddings.	Diversity-aware greedy g_1; lower score bands.
	3	Execution evidence and length-adjusted action loss.	g_1 eligibility with soft similarity, cluster and source penalties; lower score bands.
Sonnet 5	1	Completion, format and 4K-prefix loss.	Top 5,000 split into consecutive groups; no hard task deduplication.
	2	Structural quality and preference for median loss.	Retains one example per task hash, then splits the top 5,000 into consecutive groups.
	3	Format, loss and turn efficiency.	Heap follows global score order; top 5,000 split consecutively without hard task deduplication.
GPT-5.5	1	Structure, candidate NLL and task-term rarity.	Top 5,000 split consecutively; soft novelty adjustments without hard task deduplication.
	2	Structure and 2K-prefix NLL; domain counts do not affect selection.	Top 5,000 split consecutively; no hard task deduplication.
	3	Structural scoring, then NLL reranking of the top 3,000.	Top 5,000 split consecutively; no hard task deduplication.

Continued on next page

Table 5 continued

Agent	Run	Signals used	Final grouping rule
GPT-5.6 Sol	1	Completion, action, test/build and supervision features; tokenized lengths, no model loss.	Top 5,000 split consecutively without hard task deduplication.
	2	Completion, action, test/build and supervision features; estimated truncation, no model loss.	Top 5,000 split consecutively without hard task deduplication.
	3	Completion, action, test/build and supervision features; tokenized lengths, no model loss.	Top 5,000 split consecutively; ties use test/coding evidence, length and row index. No hard task deduplication.
GPT-6 Astra	1	Prompted quality judgments, execution evidence, sampled NLL and task representations; staged review.	Diverse g_1 with task deduplication and soft redundancy/coverage penalties; separated lower bands.
	2	Prompted quality judgments, execution evidence, sampled NLL and task representations; three assessments.	Diverse g_1 with task deduplication and soft redundancy/coverage penalties; separated lower bands.
	3	Prompted quality judgments, execution evidence, sampled NLL and task representations; five-level rubric.	g_1 completion/protocol/context eligibility, task deduplication and soft redundancy/coverage penalties; lower bands.
DeepSeek V4 Pro	1	Structural value and embeddings; NLL discarded.	Similarity-penalized greedy order.
	2	Loss weighted by cleanliness and completion.	One example per task hash, then top-5,000 consecutive blocks.
	3	Structural activity and recovery; no NLL in the final score.	Completed candidates, TF-IDF similarity filter at 0.92, then consecutive blocks.
Kimi K3	1	Completion, supervision and a small NLL term.	Task deduplication and g_1 cluster rebalancing.
	2	Preference for moderate length-residual loss.	Similarity threshold and up to three examples per task in g_1.
	3	Negative length-normalized NLL plus completion/truncation terms.	Cosine filter at 0.97 with fallback, then score-sorted blocks.
Gemini 3.5 Flash	1	Multiplicative structural score.	Top 5,000 completed examples, split into consecutive blocks.
	2	Additive completion/error/length score.	Top 4,000 for g_1–g_4; bottom 1,000 for g_5.
	3	NLL and thinking density on a premium candidate pool.	Premium g_1/g_2, other completed g_3, and stronger/weaker incomplete g_4/g_5.

B.2 BFCL Selection Procedures

We inspect the final method reports together with selection scripts or command traces for the 18 runs using 50-trajectory groups. The descriptions below summarize the rules used to form the submitted groups. Quality, correctness, and difficulty refer to the agents' selection signals, not independent labels of training utility.

B.3 Results by Selection

These tables retain the individual selections underlying the averages in the main text. Scores across evaluation seeds are averaged within each checkpoint; ranking metrics are computed within each selection before aggregation.

Table 6: BFCL selection recipes and differences across the three independent runs. Each run produces five disjoint groups of 50 complete trajectories.

Agent	Recipe and cross-run differences
Fable 5.1	Run 1 combines schema checks, errors, reasoning length, and length-adjusted loss; it samples descending score bands, ending with low-scoring rows. Run 2 emphasizes reasoning–action consistency and loss outliers. Run 3 separately scores reasoning, actions given reasoning, and actions without reasoning, then combines quality gates with environment coverage. Later groups include increasingly defect-heavy trajectories.
Opus 5	Run 1 prioritizes structural quality and uses a small residualized-loss term, with fixed interaction-type and environment coverage across groups. Run 2 adds verbosity-normalized reasoning loss and recovered-error bonuses, placing noncanonical argument formats in the lower groups. Run 3 probes action difficulty without the demonstrated reasoning and selects widely separated score bands with distinct environments.
GPT-6 Astra	All runs combine schema checks, argument grounding, state-changing actions, clarification, and coverage. Run 1 adds sampled-turn loss over the pool and inspection of top candidates. Run 2 uses a bounded preference for moderate reasoning loss and a fixed 70/30 interaction-type mix. Run 3 scores a quality-filtered shortlist and adds semantic checks on unsupported actions. Groups occupy separated predicted-value bands.
Kimi K3	Run 1 uses loss to penalize outliers, normalizes quality within interaction types, and selects successive groups under diversity caps. Run 2 favors intermediate total and tool-call loss. Run 3 instead rewards lower loss alongside successful responses and tool coverage, with environment caps relaxed from one to three across the groups.
DeepSeek V4 Pro	Run 1 prioritizes high sampled-turn loss, trajectory length, and final-action loss. Run 2 combines loss with structural richness, including a positive failed-call term, and uses diversity-penalized greedy selection. Run 3 emphasizes tool breadth, penalizes failed-call rate, and adds within-environment loss as a secondary signal.
Gemini 3.5 Flash	Run 1 separates groups by observed tool success, length, and turn count. Run 2 uses a heuristic score penalizing errors and loops while rewarding reasoning length and intermediate turn counts. Run 3 adds base-model loss to choose long, error-free trajectories for the top groups; short clean trajectories and error-heavy trajectories occupy lower groups.

Table 7: Terminal group-level results on a 0–1 reward scale. Runs use the same 1–3 numbering as the procedure summaries. Each reward averages 100 tasks with three attempts per task. Within each row, shading runs from darkest (best) to lightest (worst), and bold marks the best group. Colors use unrounded means; equal means receive the same shade. ρ compares the predicted g_1–g_5 order with observed performance; g_1 rank is one plus the number of groups scoring strictly higher.

Agent	Run	g_1	g_2	g_3	g_4	g_5	ρ	g_1 rank
Fable 5	r1	0.109	0.085	**0.145**	0.101	0.107	0.10	2
	r2	0.100	0.110	**0.121**	0.110	0.099	0.30	4
	r3	0.121	0.121	**0.135**	0.134	0.121	-0.50	4
Fable 5.1	r1	0.117	0.122	**0.129**	0.098	0.116	0.50	3
	r2	0.110	0.103	0.110	0.097	**0.132**	-0.30	3
	r3	**0.148**	0.111	0.121	0.112	0.101	0.70	1
Opus 5	r1	0.125	**0.137**	0.137	0.094	0.113	0.60	3
	r2	0.115	0.127	0.097	**0.135**	0.117	-0.30	4
	r3	0.108	0.111	**0.124**	0.099	0.081	0.60	3
Sonnet 5	r1	0.099	0.110	0.116	0.124	**0.149**	-1.00	5
	r2	0.118	0.123	0.101	0.117	**0.124**	-0.20	3
	r3	**0.130**	0.125	0.103	0.113	0.115	0.60	1
GPT-5.5	r1	0.107	0.086	0.109	0.099	**0.126**	-0.50	3
	r2	0.128	0.103	**0.133**	0.127	0.122	0.20	2
	r3	0.120	0.106	0.109	0.120	**0.125**	-0.40	2
GPT-5.6 Sol	r1	0.130	0.111	0.126	0.133	**0.148**	-0.70	3
	r2	0.101	0.108	0.105	**0.123**	0.100	0.10	4
	r3	0.107	0.118	0.111	0.131	**0.135**	-0.90	5
GPT-6 Astra	r1	0.112	**0.143**	0.113	0.114	0.121	-0.40	5
	r2	0.105	**0.145**	0.097	0.136	0.114	-0.10	4
	r3	**0.142**	0.112	0.108	0.098	0.078	1.00	1
DeepSeek V4 Pro	r1	**0.129**	0.098	0.109	0.125	0.099	0.30	1
	r2	**0.135**	0.133	0.104	0.120	0.132	0.60	1
	r3	0.112	0.122	**0.137**	0.121	0.132	-0.50	5
Kimi K3	r1	**0.134**	0.131	0.112	0.130	0.114	0.70	1
	r2	0.105	0.091	0.108	**0.110**	0.107	-0.60	4
	r3	0.119	0.109	0.120	**0.152**	0.103	0.10	3
Gemini 3.5 Flash	r1	0.115	0.137	0.102	**0.137**	0.135	-0.30	4
	r2	0.117	0.127	**0.132**	0.121	0.112	0.30	4
	r3	0.130	**0.139**	0.120	0.119	0.082	0.90	2

Table 8: BFCL accuracy (%) for 50-trajectory groups in each independent selection run. Scores average the 800-case evaluations across evaluation seeds. Shading compares groups within a run; ρ measures full-order agreement; g_1 rank is one plus the number of groups scoring strictly higher.

Agent	Run	g_1	g_2	g_3	g_4	g_5	ρ	g_1 rank
Fable 5.1	r1	36.00	33.25	34.83	34.38	**37.17**	-0.30	2
	r2	34.17	**36.58**	34.96	33.96	34.12	0.60	3
	r3	**35.46**	35.04	34.83	33.96	35.42	0.40	1
Opus 5	r1	35.71	**36.46**	35.08	31.96	28.75	0.90	2
	r2	36.75	**37.38**	35.00	36.54	29.46	0.80	2
	r3	35.38	35.21	**35.96**	34.38	35.04	0.60	2
GPT-6 Astra	r1	**36.38**	34.58	33.21	34.71	34.50	0.50	1
	r2	**36.75**	36.38	36.21	35.42	34.62	1.00	1
	r3	**36.33**	34.96	34.79	34.67	34.62	1.00	1
DeepSeek V4 Pro	r1	35.29	**37.42**	35.33	35.29	33.58	0.56	3
	r2	35.67	**36.04**	34.88	35.21	33.75	0.80	2
	r3	**38.62**	36.38	37.00	37.75	35.67	0.60	1
Kimi K3	r1	35.25	36.08	35.42	34.54	**36.42**	-0.30	4
	r2	36.71	35.62	37.58	36.38	**37.75**	-0.50	3
	r3	**36.92**	**36.92**	35.54	36.62	35.21	0.87	1
Gemini 3.5 Flash	r1	33.38	34.17	34.12	**35.21**	34.54	-0.80	5
	r2	35.50	33.08	33.21	34.79	**35.92**	-0.40	2
	r3	**36.42**	35.29	33.38	29.21	36.33	0.40	1

C SELECTION PROMPTS

C.1 SELECTION CONFIGURATION EXAMPLES

Each prompt reads a local `config.yaml`. The examples below show the selection-time settings for both tasks, with dataset paths written relative to the workspace. The BFCL pool contains 8,962 trajectories after excluding the 60-trajectory validation split. The `max_len` field specifies the length limit supplied during selection; realized training settings appear in Appendix A.1.

Terminal: `config.yaml`

```
base_model: ./model
dataset_path:
  ./data/train.parquet
n_groups: 5
group_size: 1000
max_len: 32768
```

BFCL: `config.yaml`

```
base_model: Qwen/Qwen3-4B
dataset_path:
  ./data/pool.parquet
n_groups: 5
group_size: 50
max_len: 16384
```

C.2 TERMINAL DATA SELECTION

All 30 reported terminal selections use the prompt below and the fixed training recipe described in Appendix A.1.

Read `config.yaml` in the current directory first. It specifies: `base_model` (HF id / path), `dataset_path` (the full SFT candidate dataset, parquet), `n_groups` (=5), `group_size` (=1000), and `max_len`. Use exactly these — do not hardcode other values.

We have that base model and the full SFT candidate dataset. A FIXED training recipe (provided as code at `recipe/sft.py`, do not modify or run it) will later fine-tune the base model separately on each of `n_groups` data groups, with identical hyper-parameters; the only variable across runs is which data subset goes in.

Downstream benchmark (what the trained models will be scored on)

Each trained model will act as a TERMINAL AGENT: dropped into a fresh Unix environment with a natural-language task (e.g. build/fix code, process data, configure the system), it interacts over many turns — issuing shell commands and reading their output — and is scored by whether the task's automated tests pass at the end. You never see the benchmark tasks; you only know the evaluation is of this kind.

Your goal

This is a SELECTION-AND-RANKING task. Predict, for every sample in the dataset, its expected value for SFT (how much it would improve the downstream benchmark if trained on), and use that prediction to SELECT `n_groups` groups of exactly `group_size` rows each, ordered so that: after training each group with the fixed recipe, benchmark score goes from high to low as the group index goes from 1 to n_groups (group 1 best, group n_groups worst).

You are FREE to design the valuation however you think works best. You may use a SINGLE signal or COMBINE MULTIPLE complementary signals (e.g. loss/perplexity, trajectory quality, task difficulty, efficiency, diversity/coverage, length effects, ...) into a composite score — your choice. Don't feel obliged to use just one textbook metric; use whatever you believe will best achieve the objective below.

Objective (what you are optimizing)

1. **PRIMARY — make group 1 as strong as possible.** Group 1 should be the single most valuable `group_size`-row subset you can assemble from the whole pool: after training on it alone, its benchmark score should be as HIGH as possible — ideally beating a random subset of the same size and approaching what training on the whole pool would give. Put real effort into getting the very top group right; that is the headline result.
2. **ORDERING — rank the groups.** Score should decrease from group 1 to group n_groups (Spearman between group index and post-trained score -> -1). Place the most valuable samples in the lowest-numbered groups and the least valuable of your selection in the highest.

Deliverable (the ONLY thing you produce)

Write `outputs/group_assignment.csv` with columns: `index,group_id`

- `index` = the row index into the dataset (config.dataset_path) (0-based)

- `group_id` = an integer in 1..n_groups
- The file contains ONLY the selected rows: exactly n_groups x group_size lines (plus header), each selected index exactly once, exactly `group_size` rows per group_id. Unselected rows must NOT appear in the file.

Also write `outputs/method.md` briefly explaining your method and why. Do NOT produce a trained model. Do NOT run the training recipe.

Hard constraints

1. INFERENCE-ONLY. You may only run forward passes of the base model (hidden states / logits are accessible). NO gradients, NO optimizer steps, NO training of any model (not even a small proxy). Gradient/optimizer calls are disabled in this sandbox and will raise.
2. The base model (config.base_model) is READ-ONLY. Use it as-is; do not fine-tune or overwrite it.
3. Do NOT access any benchmark/test set. Only use the dataset at `config.dataset_path`.
4. The n_groups groups must be mutually disjoint and each contain exactly `group_size` rows. They need NOT cover the dataset: rows outside your selection are left unused.
5. Work only in the current directory. Internet is available for packages if needed (transformers/datasets/torch/pyarrow are pre-installed).

Data format

The dataset at `config.dataset_path` has a `messages` column: a Python list of role/content dicts in chat format; each row is one multi-turn terminal-agent trajectory. The recipe trains on this `messages` column via the model chat template (tokenizer.apply_chat_template), truncating at `max_len`, with the non-assistant parts masked so loss is on assistant tokens only. To match it, build inputs with the chat template rather than concatenating raw strings.

Environment

- A single H100 GPU (80 GB) is available. The dataset (config.dataset_path), the base model (config.base_model), and `recipe/sft.py` (read-only reference) are in the current directory.
- TIME BUDGET: complete the task within 4 hours. Plan your computation to fit — the deliverable must be written and verified before the budget runs out.
- You operate fully autonomously; never ask for feedback. Just produce `outputs/group_assignment.csv`.

Execution model (IMPORTANT — single headless turn)

You run as ONE non-interactive, single-shot invocation. There is NO async re-invocation: the moment you end your turn, the process exits and ANY background work you started is KILLED. Therefore:

- Run every computation SYNCHRONOUSLY in the FOREGROUND and BLOCK until it finishes (e.g. run `python score.py` directly and wait for it to return — do NOT use `nohup ... &`, `&` backgrounding, `disown`, or detached processes).
- Do NOT use ScheduleWakeup, completion-waiters, sleep-then-yield, or any "I'll resume when notified" pattern — you will NOT be resumed.
- Long foreground commands are fine within the time budget. Only end your turn AFTER `outputs/group_assignment.csv` is written and verified on disk.

C.3 Multi-turn Tool-use Selection

Read `config.yaml` in the current directory first. It specifies: `base_model` (HF id / path), `dataset_path` (the full SFT candidate dataset, parquet), `n_groups` (=5), `group_size` (=50), and `max_len`. Use exactly these — do not hardcode other values.

We have that base model and the full SFT candidate dataset. A FIXED training recipe (documented in `recipe/README.md`, `recipe/sft.yaml` and `recipe/expand.py`; read them, do not modify or run them) will later fine-tune the base model separately on each of `n_groups` data groups, with identical hyper-parameters; the only variable across runs is which data subset goes in.

Downstream benchmark (what the trained models will be scored on)

Each trained model will act as a TOOL-USE AGENT in a stateful environment: given a set of function definitions and a user request, it must decide over several turns which functions to call and with what arguments, read the returned results, and continue until the request is fulfilled — including cases where a needed function is missing, a required argument was never given, or the context is very long. It is scored

by whether the environment ends in the correct state. You never see the benchmark tasks; you only know the evaluation is of this kind.

Your goal

This is a SELECTION-AND-RANKING task. Predict, for every trajectory in the dataset, its expected value for SFT (how much it would improve the downstream benchmark if trained on), and use that prediction to SELECT `n_groups` groups of exactly `group_size` rows each, ordered so that: after training each group with the fixed recipe, benchmark score goes from high to low as the group index goes from 1 to n_groups (group 1 best, group n_groups worst).

You are FREE to design the valuation however you think works best. You may use a SINGLE signal or COMBINE MULTIPLE complementary signals (e.g. loss/perplexity, trajectory quality, task difficulty, tool-call correctness, efficiency, diversity/coverage of environments and tools, length effects, ...) into a composite score — your choice. Don't feel obliged to use just one textbook metric; use whatever you believe will best achieve the objective below.

Objective (what you are optimizing)

1. **PRIMARY — make group 1 as strong as possible.** Group 1 should be the single most valuable `group_size`-row subset you can assemble from the whole pool: after training on it alone, its benchmark score should be as HIGH as possible — ideally beating a random subset of the same size. Put real effort into getting the very top group right; that is the headline result.
2. **ORDERING — rank the groups.** Score should decrease from group 1 to group n_groups (Spearman between group index and post-trained score -> -1). Place the most valuable trajectories in the lowest-numbered groups and the least valuable of your selection in the highest.

Deliverable (the ONLY thing you produce)

Write `outputs/group_assignment.csv` with columns: `index,meta_index,group_id`

- `index` = the 0-based ROW POSITION of the trajectory in the dataset file (config.dataset_path): 0 for the first row, n_rows-1 for the last.
- `meta_index` = the value of the `meta_index` COLUMN of that same row. It is an identifier, NOT the row position, and differs from `index` for most rows. Write both: they are checked against each other and the file is rejected if any pair disagrees, so a mix-up surfaces immediately instead of silently selecting other trajectories.
- `group_id` = an integer in 1..n_groups
- The file contains ONLY the selected rows: exactly n_groups x group_size lines (plus header), each selected index exactly once, exactly `group_size` rows per group_id. Unselected rows must NOT appear in the file.

Also write `outputs/method.md` briefly explaining your method and why. Do NOT produce a trained model. Do NOT run the training recipe.

`index` and `meta_index` are DIFFERENT numbers — read this before writing the file

`index` is WHERE a trajectory sits in the file. `meta_index` is an ID stored INSIDE the row, carried over from the corpus this pool was filtered out of. Because some trajectories were removed when the pool was built, the two drift apart: they agree for the first rows and then differ for about 96% of the file, by up to 60. Concretely, the row at position 379 carries `meta_index` 380.

In pandas terms: `index` is the position you would pass to `.iloc[]`; `meta_index` is the column `df["meta_index"]`. Reading the column when you need the position (or the reverse) selects DIFFERENT trajectories than the ones you scored, and most of those wrong values are themselves in range, so nothing would look broken.

For each selected trajectory write the pair exactly as the dataset has it: `index` = its row position, `meta_index` = `df["meta_index"].iloc[index]`.

Hard constraints

1. INFERENCE-ONLY. You may only run forward passes of the base model (hidden states / logits are accessible). NO gradients, NO optimizer steps, NO training of any model (not even a small proxy). Gradient/optimizer calls are disabled in this sandbox and will raise.
2. The base model (config.base_model) is READ-ONLY. Use it as-is; do not fine-tune or overwrite it.
3. Do NOT access any benchmark/test set. Only use the dataset at `config.dataset_path`.
4. The n_groups groups must be mutually disjoint and each contain exactly `group_size` rows. They need NOT cover the dataset: rows outside your selection are left unused.

5. Work only in the current directory. Internet is available for packages if needed (transformers/datasets/torch/pyarrow are pre-installed).

Data format

Each row of the dataset is ONE multi-turn tool-use trajectory. Columns: `meta_index` (an identifier carried over from the source corpus – NOT the row position), `env_id` (the simulated environment the trajectory was collected in), `task_id`, `traj_type` ("conversation" or "non_conversation"), `steps` (number of assistant turns) and `messages`, a list of role/content dicts:

- `messages[0]` is the system message; it contains the environment instructions AND the function definitions available in that trajectory (Qwen3 `<tools>` format).
- Then user and assistant messages alternate. A user message is either the user's request or one or more `<tool_response>...</tool_response>` blocks returning what the previously called functions produced.
- EVERY assistant message starts with its reasoning in `<think>...</think>`, followed by text and/or `<tool_call>{"name": ..., "arguments": {...}}</tool_call>` blocks.

The recipe does NOT train on a row as one sequence: it expands a row with `steps` assistant turns into `steps` samples, each supervising one assistant turn (with its `<think>`) given the preceding turns with their `<think>` blocks removed — see `recipe/expand.py`, and use it (or match it exactly) when you compute anything per sample with the model chat template (tokenizer.apply_chat_template), truncating at `max_len`.

Environment

- A single H100 GPU (80 GB) is available. The dataset (config.dataset_path), the base model (config.base_model), and `recipe/` (read-only reference) are in the current directory.
- Scale: about 9k trajectories averaging ˜18 turns and ˜6.5k tokens each when rendered; a full forward pass over the whole pool is feasible on this GPU but not free — plan it.
- TIME BUDGET: complete the task within 4 hours. Plan your computation to fit — the deliverable must be written and verified before the budget runs out.
- You operate fully autonomously; never ask for feedback. Just produce `outputs/group_assignment.csv`.

Execution model (IMPORTANT — single headless turn)

You run as ONE non-interactive, single-shot invocation. There is NO async re-invocation: the moment you end your turn, the process exits and ANY background work you started is KILLED. Therefore:

- Run every computation SYNCHRONOUSLY in the FOREGROUND and BLOCK until it finishes (e.g. run `python score.py` directly and wait for it to return — do NOT use `nohup ... &`, `&` backgrounding, `disown`, or detached processes).
- Do NOT use ScheduleWakeup, completion-waiters, sleep-then-yield, or any "I'll resume when notified" pattern — you will NOT be resumed.
- Long foreground commands are fine within the time budget. Only end your turn AFTER `outputs/group_assignment.csv` is written and verified on disk.