DATAOS: A RECURSIVE SELF-IMPROVEMENT SYSTEM DRIVEN BY ENVIRONMENT DATA

Anonymous authors
Paper under double-blind review

ABSTRACT

Recursive self-improvement (RSI) seeks to make AI systems capable of revising the procedures through which they develop and improve their capabilities. Through successive experiments, these systems accumulate environment data (e.g., datasets and execution traces) that can inform subsequent improvements. Using these data effectively requires preserving the conditions behind past outcomes and identifying evidence and resources relevant to the evolving system. We introduce DataOS, a four-layer system for data-driven RSI that couples task execution and system evolution through continuously maintained environment data. DataOS preserves provenance and dependencies across versioned artifacts and execution records to link outcomes to their actual inputs and configurations, enabling agents to use accumulated evidence and reusable resources to diagnose limitations and make targeted revisions to data, models, and the harness. We evaluate DataOS on PostTrainBench for autonomous post-training and Curation-Bench for multimodal data selection. Under matched researcher models, reasoning effort, and budgets, DataOS improves aggregate scores over native coding agents by up to 33.5% and 6.4%, respectively. Ablations support the contributions of structured history, reusable operators, and harness evolution, while transfer experiments show that accumulated research state can benefit subsequent runs even when target-model weights are reset. Code is available at `https://anonymous.4open.science/r/DATAOS-BE67`.

1 INTRODUCTION

Recursive self-improvement (RSI) aims to enable AI systems to improve both their capabilities and the processes that produce further improvements. A typical RSI system (Zhang et al., 2026a) operates through an iterative loop: it examines feedback from previous attempts, identifies potential improvements, modifies some part of the system, and evaluates the resulting variant, with successful changes retained to inform subsequent iterations. Building on this general paradigm, recent LLM-based RSI systems have explored changes to executable code (Robeyns et al., 2025), training data (Wang et al., 2026), and model parameters (Zweiger et al., 2025). In this way, earlier improvements can progressively become part of the machinery that produces later ones.

Despite these advances, extending RSI to challenging, long-horizon tasks requires moving beyond aggregate evaluation scores. In such settings, scalar metrics obscure why attempts fail. To diagnose failures and ground subsequent modifications, agents must instead examine the execution artifacts produced across iterations, such as datasets, checkpoints, and traces, which we collectively term *environment data*. However, existing methods engage with these data primarily through ad-hoc or bounded channels: GEPA (Agrawal et al., 2026) reflects on selected traces via task-specific adapters, while Meta-Harness (Lee et al., 2026) archives historical code and runs largely for passive inspection. These approaches leave a critical gap between simply accessing past evidence and systematically analyzing it to construct new interventions. We therefore propose *data-driven RSI*, a paradigm where agents systematically organize and reason over environment data to drive system modifications, whose resulting artifacts continuously inform subsequent iterations.

Realizing this paradigm presents two central challenges. First, *collecting and organizing environment data*. Environment data are produced by heterogeneous components in different forms and at different stages of execution. A useful representation must therefore unify these artifacts while preserving their provenance and dependencies across iterations. Second, *selecting and using environment data*.

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.

As execution histories accumulate, the relevance of an artifact depends on the current system state and improvement objective. Agents must identify pertinent evidence and reusable resources, interpret their implications, and translate them into executable changes.

To address these challenges, we introduce DATAOS, a four-layer system that couples task execution with system evolution. The Environment Data Layer connects artifacts and execution records to the component versions, configurations, and operations that produced them. Using these data, the Agent Layer coordinates experiment planning and system revision: agents trace dependencies, inspect artifacts, and run diagnostics to identify what to change and why. The Pipeline Layer structures task execution, system evolution, and environment-data maintenance into editable plans that invoke Runtime Layer operations and Environment Data Layer maintenance interfaces. Task outcomes and revision results feed back into the environment data, informing subsequent experiments and system revisions in a continuous process of data-driven RSI.

We evaluate DATAOS on PostTrainBench and Curation-Bench, covering autonomous post-training and multimodal data selection. Under matched researcher models, reasoning effort, and budgets, DATAOS achieves relative improvements in aggregate scores of up to 33.5% on PostTrainBench and 6.4% on Curation-Bench over native coding agents. Ablations support the contributions of structured history, reusable operators, and harness evolution, while trajectory and transfer analyses show how accumulated research state can benefit subsequent experiments.

Our main contributions are summarized as follows:

- We formulate *data-driven recursive self-improvement*, in which execution generates persistent environment data and analysis of these data guides revisions to the datasets, models, and harness used in subsequent iterations.
- We develop DATAOS, a four-layer system that integrates environment-data collection and maintenance with diagnostic analysis and system evolution, enabling agents to revisit earlier variants, reuse artifacts and methods, and extend their experimental procedures.
- We demonstrate aggregate gains over native coding agents and specialized research systems on PostTrainBench and Curation-Bench, and use ablation, trajectory, and transfer studies to assess component contributions and the benefits and limits of accumulated research state.

2 RELATED WORK

Recursive self-improvement. Recursive self-improvement (RSI) uses execution feedback to revise both a system and the processes that produce further improvements. At its simplest, this feedback consists of scalar utility scores that quantify candidate performance (Zelikman et al., 2024; Hu et al., 2025), and aggregating descendant scores can further estimate an agent's self-improvement potential (Wang et al., 2025b). Such signals summarize success but leave its causes implicit. Richer feedback exposes runtime errors and prediction logs (Yin et al., 2025; Zhang et al., 2025), with verbal reflections and textual critiques making observations actionable (Shinn et al., 2023; Yuksekgonul et al., 2024). Trajectory-level feedback adds reasoning steps, tool calls, and intermediate outputs, enabling diagnosis of how an execution succeeds or fails (Agrawal et al., 2026). The feedback context expands further when evidence persists across attempts: archives retain evaluated agent implementations (Robeyns et al., 2025; Zhang et al., 2026a), and historical source code, scores, and execution traces become jointly available for subsequent proposals (Lee et al., 2026). Accumulated trajectories can be organized into a structured memory of the search (Yan et al., 2026), while retained outcomes and traces supply context for revising the improvement procedure itself (Zhang et al., 2026b; Qu & Lu, 2026; Zhang et al., 2026c). MemRL (Zhang et al., 2026d) combines semantic similarity with utility estimates learned from environmental rewards to select past experiences for reuse. As feedback grows richer and spans more iterations, its use depends on preserving the artifacts and experimental conditions behind each observation. Agents must also determine which historical findings and artifacts remain relevant as the system's components and improvement objectives change. DATAOS develops this direction through *environment data*: datasets, checkpoints, execution traces, and records connected by provenance and dependencies. Our *data-driven RSI* approach systematically collects, organizes, and analyzes these data to construct executable system modifications, whose resulting artifacts and observations enrich the evidence available to subsequent iterations.

Automated post-training and data curation. Automated post-training requires both effective training data and suitable optimization strategies. Data-curation methods automate instruction generation, filtering, and influence-based selection (Wang et al., 2023; Xia et al., 2024; Wang et al., 2025a). Because data utility depends on model weaknesses, feedback-driven curation iteratively revises strategies using student errors and failure trajectories (Khan et al., 2025; Wang et al., 2026). DataMaster (Du et al., 2026) organizes data discovery and refinement as tree search, reusing datasets, artifacts, and findings across branches under a fixed training algorithm. Allowing training strategies to change further requires coordinating data, optimization, and evaluation (Shi et al., 2026). AutoTrainess (Yu et al., 2026) supports this coordination through specialized interfaces that encode human training expertise as workflows and execution constraints. Curation-Bench and RSIBench-Data evaluate data refinement under fixed training infrastructure (Kang et al., 2026; Meng et al., 2026). PostTrainBench tests broader post-training capabilities (Rank et al., 2026), while Agent^2 RL-Bench examines interactive RL pipelines (Chen et al., 2026). Data curation and automated post-training provide complementary settings for evaluating DATAOS: environment data guide data refinement in the former and coordinated revisions of data, models, and the harness in the latter.

3 DATA-DRIVEN RSI SYSTEM: DATAOS

3.1 PRELIMINARIES

Data-driven RSI couples task execution and system evolution through continuously maintained environment data. Given a task objective Ω, let $\mathcal{S}_t$ denote the system state and $\mathcal{E}_t$ the accumulated environment data at step t. The system has two operations:

$$\begin{aligned}(Y_t, \mathcal{E}_{t+1}) &= \mathrm{TaskExecute}(\Omega, \mathcal{S}_t, \mathcal{E}_t),\\ (\mathcal{S}_{t+1}, \mathcal{E}_{t+1}) &= \mathrm{EvoExecute}(\mathcal{S}_t, \mathcal{E}_t).\end{aligned} \tag{1}$$

Here, $\mathrm{TaskExecute}$ carries out the task under the current system state and produces results Y_t, while $\mathrm{EvoExecute}$ uses accumulated environment data to diagnose limitations and revise the system. Both flows maintain their artifacts, execution records, and outcomes in $\mathcal{E}_{t+1}$ as traceable evidence and reusable resources for subsequent execution and revision. $\mathcal{S}$ revisions can in turn change how subsequent task execution and evolution are performed.

3.2 SYSTEM ARCHITECTURE

As shown in Figure 1(a), DATAOS comprises four layers that separate agent orchestration, structured planning, runtime operations, and environment-data management. The operations in each layer can be implemented directly using the interface of the lower layer, ignoring the specific details. At the same time, the objects of this layer will serve as an encapsulated tool to provide interfaces to the upper layer, supporting task execution and system evolution across layers.

(1) Agent Layer. To coordinate task execution and system evolution, this layer defines four specialized agent roles. The *Leader* is responsible for task planning, the *Evolver* for system evolution, the *Worker* for execution, and the *Maintainer* for environment-data maintenance. Each role is defined by prompts and tool scopes associated with a harness version. A unified agent adapter supports backends such as Codex and Claude, keeping role coordination independent of backend-specific invocation.

(2) Pipeline Layer. To make agent decisions executable, this layer organizes them into structured plans. The Leader, Evolver, and Maintainer express their decisions through three editable plans: a `(i)Task Plan` for data processing, training, and evaluation; an `(ii)Evolution Plan` for analysis, revision, and validation; and a `(iii)Maintenance Plan` for registration, association, and versioning of environment data. Each plan comprises specifications of participating components and operations, including their inputs, outputs, configurations, and dependencies.

(3) Runtime Layer. To support both task execution and system evolution, the runtime exposes *execution interfaces* and *modification interfaces*. Execution interfaces unify model training and evaluation across supported frameworks through `dataos-train` and `dataos-eval`, and expose a reusable operator library through `dataos-data`. For evolution, training and data operations enable revisions to models and dataset. Harness-modification interfaces support editing prompts, developing new operators, and creating pipelines.

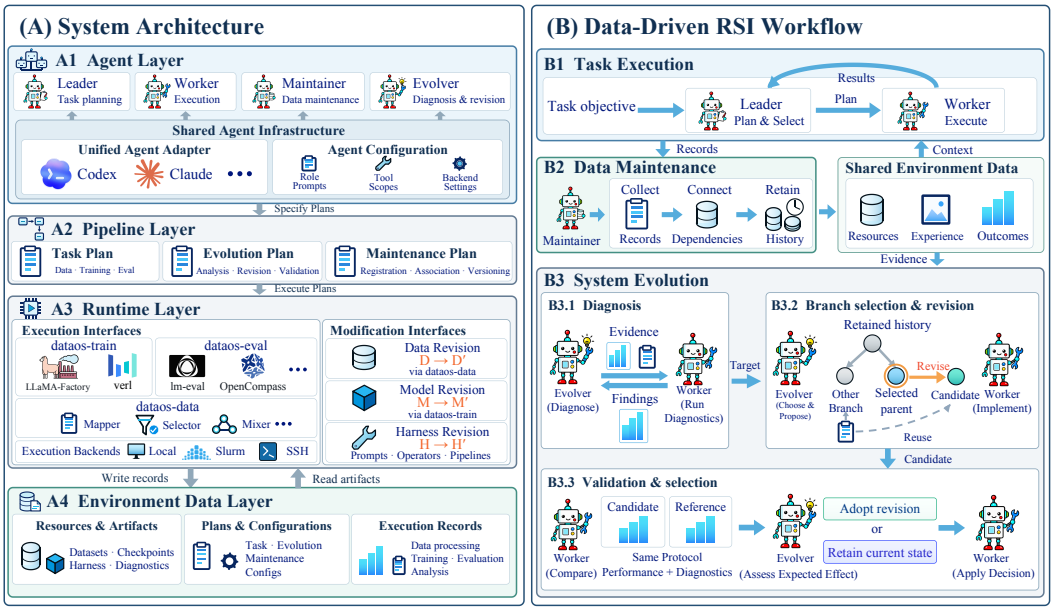

Figure 1: Overview of DATAOS. (a) Four-layer architecture connecting agents, plans, runtime operations, and environment data. (b) Maintained environment data connect task execution with system evolution, guiding diagnosis, resource reuse, and revisions to data, models, and the harness.

(4) Environment Data Layer. A unified collection and access interface organizes artifacts and execution records under shared definitions for datasets, models, experiments, training runs, and evaluations. Version and dependency management connects results to the inputs, configurations, and operations that produced them, making heterogeneous environment data jointly accessible and traceable across iterations to support subsequent diagnosis, comparison, and resource reuse.

For example, after tracing evaluation errors to gaps in the training data, an agent formulates a plan to revise the dataset and retrain the model. The Worker executes this plan through Runtime Layer tools and interfaces to update the dataset and model state. The Environment Data Layer maintains the resulting artifacts and execution records, together with their originating plans and component versions, for subsequent planning. Implementation details are provided in Appendix A.

3.3 MULTI-AGENT ORCHESTRATION

DATAOS organizes its agents around two coupled flows: *task execution* and *system evolution*, connected through environment-data maintenance (Figure 1(b)). The Leader plans experiments using available capabilities and selects task outputs; the Evolver diagnoses limitations, formulates revision hypotheses, and selects system variants. The Worker executes their plans, while the Maintainer organizes the resulting environment data. Task experiments can produce new datasets and checkpoints, while evolution explicitly assesses proposed revisions using evaluation results and the associated diagnostic evidence and decides which system variant to retain for subsequent work.

Task execution. Given a task objective and initial resources, the Leader examines the available environment data and constructs a task plan specifying the starting components, operations, and expected observations. The Worker realizes this plan through the Pipeline and Runtime Layers, producing artifacts and execution results. The Maintainer then incorporates the plan and realized execution into the persistent environment data. These updated records support the Leader's subsequent planning. When evidence motivates system evolution, the Leader supplies a concrete issue, supporting references, and an improvement objective to the Evolver.

System evolution. For an improvement objective or a pending issue, the Evolver examines the associated environment data, including prior diagnoses and evidence from earlier attempts, and

initializes the analysis stage of an evolution plan. It uses environment-data access interfaces to analyze the records and diagnose limitations, then specifies candidate revisions and their validation. The Worker implements and evaluates these candidates, returning their outcomes with plan and execution references. Based on these results, the Evolver selects a revision or retains the current configuration, and the Worker applies the decision. The Maintainer records the evolution attempt and resulting changes, making both its artifacts and outcomes available to subsequent task planning.

The two flows are coupled through more than evaluation feedback: task execution supplies evidence and reusable resources for evolution, while evolution can change the data, models, and procedures used by later experiments. The following sections detail how environment data are maintained and how the Evolver uses them to develop revisions.

3.4 Environment Data Collection and Maintenance

Following its Maintenance Plan, the Maintainer extracts plans, component configurations, tool calls, artifacts, observations, and decisions from agent and runtime execution traces. It registers these objects and establishes *horizontal links* within each task or evolution attempt, connecting its plan and inputs to executed operations, outputs, and decisions. For example, an evaluation result is linked to its checkpoint, training run, and input dataset. The Maintainer then reasons over these records together with existing environment data to identify reuse and revision relationships, establishing *vertical links* to historical artifacts, executions, and parent versions. Based on these links, it resolves references to existing versions and assigns new version identifiers to revised components. These fixed version references support subsequent retrieval, comparison, and reuse of environment data.

3.5 Environment-Data-Driven Evolution

The Evolver uses accumulated environment data $\mathcal{E}_t$ to explain observed limitations, decide what to change, and assess the effects of those changes. The diagnostic requests, revision actions, and validation criteria developed during this process are recorded in the Analysis Spec, Revision Spec, and Validation Spec of an *Evolution Plan*, respectively. The plan is maintained as environment data with links to its supporting records and outcomes to guide subsequent evolution.

Analysis and diagnosis. Given an improvement objective or an unresolved issue, the Evolver specifies the diagnostic question and relevant environment-data references in the Analysis Spec. It then uses the Environment Data Layer's access interfaces: `trace` follows dependencies from outcomes to the data, model, and harness configurations that produced them; `inspect` examines referenced artifacts and execution records; and `analyze` compares distributions, errors, and outcomes across these data. For example, linked evaluation outputs, filtering records, and sample-coverage statistics can reveal that a filter removed task-relevant examples, explaining increased errors on a task subset. The resulting diagnosis identifies the responsible component or operation and explains the observed limitation by connecting observed errors to specific execution conditions.

Revision solution. The diagnosis motivates an intervention and a predicted effect. In the filtering example, evidence that task-relevant samples were removed can motivate revising the filter to restore coverage and improve performance on the affected subset. Based on the diagnosis outpus, the Evolver selects an existing system version to revise. It then retrieves artifacts and methods from prior attempts that can be reused to address the diagnosed limitation. The resulting Revision Spec explains the selected parent and proposed artifacts and methods changes to their supporting evidence, required operations, and expected effect.

Validation and update. Before execution, the Evolver designs comparison references, diagnostic observations, and selection criteria in the Validation Spec, which references a Task Plan for evaluating the candidate. The Worker implements the revision and executes the affected stages of that plan under the fixed evaluation protocol. The Evolver examines task performance together with the diagnostic observations to assess the predicted effect. In the example, it checks whether coverage was restored and whether performance on the affected subset improved. These observations guide its decision to adopt the revision or retain the current state. The Worker applies the decision, and the Maintainer records the attempt and outcome as environment data for subsequent evolution.

4 EXPERIMENTS

4.1 EXPERIMENT SETUP

Benchmarks. We evaluate DATAOS on two complementary benchmarks: (i) **PostTrainBench (PTB)** (Rank et al., 2026) evaluates automated post-training, where agents autonomously construct training data and choose training strategies to improve target-model performance. We replace open-ended data acquisition with a shared, fixed pool of public data to improve reproducibility, and filter the pool for evaluation-set overlap to reduce contamination risks. Experiments cover four target models and seven tasks, with each target–task pair optimized independently under a 10-hour budget on one H100 GPU. (ii) **Curation-Bench** (Kang et al., 2026) evaluates multimodal data selection, where agents search for effective training subsets from a candidate pool under fixed training recipes. We use its LLaVA-665K setting with three target models and 10 curation–training–evaluation iterations per target, selecting 10,000 examples per iteration. Training recipes and target initialization are held fixed, and performance is evaluated on eight multimodal benchmarks. Full benchmark protocols, including data-pool construction and filtering, are provided in Appendix B.1 and B.2.

Baselines. We compare DATAOS with two categories of baselines: (i) leading general-purpose coding agents, including CODEX CLI and CLAUDE CODE; and (ii) specialized research agent systems, including DataMaster (Du et al., 2026) and AutoTrainess (Yu et al., 2026). Primary native-agent comparisons match the researcher family and use medium reasoning effort, with native-max serving as a higher-effort reference. Detailed configurations are provided in Appendix B.3.

Implementation DATAOS provides a unified interface to general-purpose agent harnesses. We instantiate it with CODEX CLI and CLAUDE CODE, powered by GPT-5.6 Sol and Opus 5, respectively, with medium reasoning effort for both. Further system configurations, resource settings, and version information are deferred to Appendix B.3.

4.2 MAIN RESULTS

Table 1 summarizes performance on PostTrainBench and Curation-Bench. For both researcher families, DATAOS achieves the highest aggregate score among all systems on both benchmarks, including the specialized research-agent baselines DataMaster and AutoTrainess. To measure the gain within each researcher family under matched reasoning effort, we compare DATAOS with the corresponding native-medium agent. DATAOS improves over these references by 8.04 and 10.13 points on PostTrainBench and by 2.66 and 2.06 points on Curation-Bench for Claude and Codex, respectively. Beyond these aggregate improvements, we further examine where these gains emerge and under what conditions DATAOS provides the most value.

More reasoning does not consistently improve post-training. The benefit of increasing reasoning effort varies across tasks. On PostTrainBench, moving from medium to max effort raises the native-agent average from 36.72 to 37.53 for Claude Code and from 30.23 to 34.30 for Codex. However, on Curation-Bench, the same change lowers the aggregate score from 41.29 to 40.84 for Claude Code and from 41.09 to 40.68 for Codex (Table 1). These benchmark-dependent effects provide context for the comparison with DATAOS. Even at medium reasoning effort, DATAOS outperforms the corresponding native-max agents on both benchmarks. The comparison indicates that performance depends not only on how much reasoning an agent performs, but also on whether that reasoning is converted into effective, task-grounded data and training interventions.

No researcher is uniformly strongest across target models. The relative strength of the native researchers reverses across target models. On Gemma3-4B, Claude Code scores 19.93 compared with 28.68 for Codex, whereas on Qwen3-4B, Claude Code scores 48.56 compared with 29.89 for Codex. The corresponding DATAOS variants follow the same target-dependent pattern: on Gemma3-4B, DATAOS with Claude Code gains 19.75 points compared with 5.69 points for Codex; on Qwen3-4B, the gains reverse to 4.61 and 17.39 points, respectively (Table 8). Thus, neither researcher is uniformly preferable across target models. DATAOS improves both researcher families while preserving the flexibility to benefit from whichever researcher is better suited to the target, rather than relying on a single universally dominant model.

Table 1: Main results on (a) PostTrainBench and (b) Curation-Bench, averaged across four and three target models, respectively. All scores are reported on a 0–100 scale. "Gain vs. native" reports improvements over the corresponding native-medium agent, computed before rounding. Bold highlights the best scores.

(a) PostTrainBench

Method	AIME	Arena	BFCL	GPQA	GSM8K	Health	HumanEval	Avg.
References								
Base models	2.81	2.39	27.50	10.43	24.47	6.30	14.79	12.67
DataMaster	5.63	10.38	74.50	26.28	75.34	11.37	46.19	35.67
AutoTrainess	6.77	13.11	69.25	28.35	77.52	13.05	44.82	36.12
Claude								
Claude Code (Opus, med.)	10.21	21.47	69.00	17.80	72.29	16.85	49.39	36.72
Claude Code (Opus, max)	11.04	24.92	71.00	15.90	72.04	16.11	51.68	37.53
DataOS (Opus, med.)	**14.90**	**31.29**	**89.25**	27.23	77.01	**20.58**	**53.05**	**44.76**
Gain vs. native	+4.69	+9.83	+20.25	+9.43	+4.72	+3.73	+3.66	+8.04
GPT/Codex								
Codex CLI (Sol, med.)	6.98	9.35	75.00	23.44	57.56	10.89	28.36	30.23
Codex CLI (Sol, max)	5.42	7.90	66.25	24.78	77.90	13.93	43.90	34.30
DataOS (Sol, med.)	8.54	14.44	88.75	**31.87**	**78.56**	15.21	45.12	40.36
Gain vs. native	+1.57	+5.09	+13.75	+8.43	+21.00	+4.32	+16.77	+10.13

(b) Curation-Bench

Method	Hallusion	LLaVA	OCR	MMMU	Math Vista	MMStar	MM Bench	MM-Vet	Avg.
References									
Random	32.67	39.40	52.27	39.37	37.07	40.62	52.15	41.59	41.89
Original (no curation)	25.00	36.77	44.53	38.33	36.20	40.29	48.45	34.22	37.97
DataMaster	32.08	40.40	51.23	38.83	36.80	40.58	51.12	39.39	41.30
AutoTrainess	32.54	40.85	**53.33**	**40.06**	37.32	40.66	52.39	40.70	42.23
Claude									
Claude Code (Opus, med.)	33.03	42.27	49.20	38.00	37.53	40.51	53.98	35.83	41.29
Claude Code (Opus, max)	30.18	41.97	47.53	38.96	37.37	39.65	54.87	36.18	40.84
DataOS (Opus, med.)	**35.35**	**44.30**	53.07	39.96	**39.23**	**43.78**	**56.99**	38.93	**43.95**
Gain vs. native	+2.32	+2.03	+3.87	+1.96	+1.70	+3.26	+3.01	+3.10	+2.66
GPT/Codex									
Codex CLI (Sol, med.)	32.07	42.10	51.57	37.78	36.47	40.53	49.91	38.32	41.09
Codex CLI (Sol, max)	31.35	39.87	50.00	37.33	36.57	41.82	51.49	37.04	40.68
DataOS (Sol, med.)	33.79	43.38	53.13	39.80	37.78	41.63	53.18	**42.54**	43.16
Gain vs. native	+1.73	+1.28	+1.57	+2.03	+1.31	+1.09	+3.27	+4.22	+2.06

Environment-data-driven evolution supports constrained optimization. Iterative data selection by native agents does not reliably outperform random selection. Claude Code and Codex at medium effort fall below Random in five of the six agent–target pairs, despite using the same candidate pool, 10,000-example subsets, training recipes, and ten-iteration budget (Table 9). DataOS reverses this pattern: both DataOS variants exceed Random and the original model in all six pairs. On Qwen2.5-VL-3B, native Claude Code and Codex score 0.6305 and 0.6310, below Random (0.6334) and the original model (0.6383), whereas DataOS–Claude Code and DataOS–Codex reach 0.6556 and 0.6444, respectively. These results show that organizing environment data around downstream model outcomes enables effective, target-specific refinement under fixed training constraints.

4.3 Component Ablations

The main results establish the end-to-end gains achieved by DataOS. We next assess which system capabilities contribute to these gains. We conduct component ablations with Codex on Qwen3-4B-Base across all seven tasks of PostTrainBench, using the same target model, evaluation protocol, and per-task research budget as the full system. Table 2 summarizes the results; detailed ablation configurations are provided in Appendix B.4.2.

Table 2: Component ablations of DATAOS–Codex on Qwen3-4B-Base across seven PostTrainBench tasks. Δ denotes changes in the mean score relative to full DATAOS. Detailed configurations are provided in Appendix B.4.2.

Variant	AIME	Arena	BFCL	GPQA	GSM8K	Health	Human Eval	Avg.	Δ
Full DATAOS	11.67	18.40	91.00	34.15	90.37	22.63	62.80	47.29	—
Current record only	10.00	14.20	85.00	31.25	84.76	18.70	56.10	42.86	-4.43
Sequential history	10.83	16.72	89.00	33.04	88.10	21.06	60.37	45.59	-1.70
w/o Operator Library	12.50	16.10	87.00	32.59	87.19	20.40	58.54	44.90	-2.39
w/o Harness Evolution	10.83	15.80	88.00	33.04	88.55	20.90	60.37	45.36	-1.93

Structured access to experimental history. Replacing the structured workspace with current-record feedback reduces the mean score from 47.29 to 42.86, with lower scores on all seven tasks. Providing sequential history recovers 2.73 points, raising the mean to 45.59, but remains 1.70 points below the full system. Because sequential history already retains past records, the remaining gap supports the contribution of structured access beyond history retention alone.

Reusable data operators. Removing the training-data operator library reduces the mean score from 47.29 to 44.90, a decrease of 2.39 points. The ablated variant still permits custom data-processing code under the same research budget. Thus, the ability to implement data operations during a run does not fully recover the benefit of having reusable operators available from the outset.

Harness evolution. Freezing the initial harness reduces the mean score from 47.29 to 45.36, a decrease of 1.93 points, with lower scores on all seven tasks. Structured history, the operator library, and data and training search remain available in this variant. The gap therefore supports an additional benefit from revising the research procedures that use these resources, beyond selecting data and training configurations through a fixed harness.

4.4 MULTI-ROUND JOINT OPTIMIZATION AND TRAJECTORY ANALYSIS

Beyond individual-task optimization, we consider a more demanding setting in which the researcher iteratively improves a single target model across all seven PostTrainBench tasks, coordinating data and training decisions under cross-task trade-offs. Detailed settings are provided in Appendix B.4.1. DATAOS achieves a best observed mean score of 46.79, compared with 43.86 for native Codex (Table 10). We trace how experimental decisions evolve in response to feedback across rounds.

The trajectory spans multi-domain supervised fine-tuning, continued training, preference optimization, and model merging, with later branches revisiting earlier checkpoints or adapting previously developed components. We use T_n to denote the recorded stages in Table 11, each associated with an evaluated candidate checkpoint. The following examples illustrate these two forms of continuity across experiments.

Revisiting earlier checkpoints. The researcher can revisit an earlier checkpoint to explore a new branch when subsequent attempts underperform. At T_3, the model reaches a mean score of 45.13. Two successive training continuations, T_4 and T_5, lower the score to 43.99 and 43.54. The researcher then starts a new continuation from T_3, reusing its training-data recipe and reaching 46.79 at T_6. This branch illustrates how retaining earlier checkpoints and their associated recipes allows the researcher to resume exploration from a stronger historical state.

Reusing and adapting experimental components. An experiment that does not improve the best score can still provide components for subsequent attempts. At T_{10}, the researcher combines two preference-data subsets for preference optimization, obtaining a mean score of 43.29. It subsequently evaluates each subset separately while adjusting the training configuration: T_{11} reaches 45.35 with one subset, and T_{12} reaches 46.39 with the other. Both attempts reuse the preference-training pipeline established at T_{10}, showing how an initial experiment can support subsequent exploration through its data components and executable methods.

Table 3: Accumulation and transfer of research state with DATAOS–Codex. All destination runs receive a 10-hour budget. Δ denotes gains over the no-transfer baseline. Detailed settings are provided in Appendix B.4.3.

(a) Cross-benchmark transfer: Qwen3-4B-Base fixed

Source benchmark	Destination benchmark	No transfer	Transfer	Δ
GSM8K	AIME 2025	11.67	15.00	+3.33
HumanEval	BFCL	91.00	94.00	+3.00
GSM8K	HumanEval	62.80	61.59	-1.21
GSM8K	GSM8K	90.37	92.19	+1.82

(b) Cross-model transfer: GSM8K fixed

Source target model	Destination target model	No transfer	Transfer	Δ
Qwen3-1.7B-Base	Qwen3-4B-Base	90.37	91.74	+1.37
Qwen3-4B-Base	SmolLM3-3B-Base	80.44	83.40	+2.96
Qwen3-4B-Base	Gemma-3-4B-PT	65.58	67.63	+2.05
Qwen3-4B-Base	Qwen3-4B-Base	90.37	92.19	+1.82

4.5 ACCUMULATION AND TRANSFER OF RESEARCH STATE

We examine the value of accumulated research state at three levels: within the same setting, across tasks, and across target models. Destination runs inherit accumulated environment data and adopted harness revisions, while target-model training starts from the destination model's original base checkpoint. Agents use the inherited evidence, artifacts, and methods to construct new plans for the destination objective and continue system evolution. Table 3 compares runs with and without inheritance under equal destination budgets. Detailed settings are provided in Appendix B.4.3.

Same-setting accumulation. With both the task and target model unchanged, inheritance improves Qwen3-4B-Base on GSM8K from 90.37 to 92.19. This gain shows that research-state accumulation can benefit a subsequent run even when target-model training restarts from the base checkpoint.

Cross-task transfer. With Qwen3-4B-Base fixed, GSM8K-to-AIME and HumanEval-to-BFCL transfer improve destination scores by 3.33 and 3.00 points, respectively, whereas GSM8K-to-HumanEval reduces the score by 1.21 points. The two GSM8K transfers have opposite outcomes despite sharing the source task and target model, suggesting that the usefulness of inherited state depends on its relevance to the destination task.

Cross-model transfer. With GSM8K fixed, all three transfers improve performance. Qwen3-1.7B-Base to Qwen3-4B-Base gains 1.37 points; Qwen3-4B-Base to SmolLM3-3B-Base and Gemma-3-4B-PT gains 2.96 and 2.05 points, respectively. Together with the mixed cross-task results, these observations suggest that useful research experience can extend beyond a particular target model while remaining sensitive to the task being optimized.

5 CONCLUSION

In this paper, we study how an AI researcher can improve both a target model and the process that produces later improvements. We formulate this problem as data-driven recursive self-improvement and introduce DATAOS, a framework that turns environment data into executable, state-conditioned interventions. Across PostTrainBench and Curation-Bench, DATAOS outperforms matched native-medium agents and specialized research-agent baselines, including under fixed-recipe data selection. Component ablations support the value of structured history, reusable operators, and harness evolution. Multi-round and transfer experiments show that accumulated research state can support subsequent improvement within and across tasks and target models, with benefits depending on the destination setting. Together, these results establish an inspectable evidence-to-model loop for data-driven RSI.

AI USE STATEMENT

Generative AI tools were used to assist with code development, repository inspection, literature organization, and language editing. The authors reviewed all AI-assisted text, manually checked cited sources, and verified experimental code and generated artifacts before use. No model-generated numerical claim is included without a corresponding run artifact. The authors take responsibility for the final content of the paper.

ETHICS STATEMENT

We use publicly available datasets, filter the PostTrainBench data pool for evaluation-set overlap, and retain execution records for traceability. Applications of DATAOS to other settings should account for potential biases, privacy risks, and limitations of the underlying data and models.

REPRODUCIBILITY STATEMENT

The implementation, typed schemas, configuration manifests, and analysis code will be included in the anonymous supplement. Section 4 specifies the main setting; Appendix B documents data-pool construction, inspected evaluation settings, and outstanding run-level verification; and Appendix C reports the complete run matrix and artifact provenance. Code is available at `https://anonymous.4open.science/r/DATAOS-BE67`.

REFERENCES

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective Prompt Evolution Can Outperform Reinforcement Learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL `https://openreview.net/forum?id=RQm2KQTM5r`.

Rahul K. Arora, Jason Wei, Rebecca Soskin Hicks, Preston Bowman, Joaquin Quiñonero-Candela, Foivos Tsimpourlas, Michael Sharman, Meghan Shah, Andrea Vallone, Alex Beutel, Johannes Heidecke, and Karan Singhal. Healthbench: Evaluating large language models towards improved human health, 2025. URL `https://arxiv.org/abs/2505.08775`.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report, 2025. URL `https://arxiv.org/abs/2502.13923`.

Elie Bakouch, Loubna Ben Allal, Anton Lozhkov, Nouamane Tazi, Lewis Tunstall, Carlos Miguel Patiño, Edward Beeching, Aymeric Roucher, Aksel Joonas Reedi, Quentin Gallouédec, Kashif Rasul, Nathan Habib, Clémentine Fourrier, Hynek Kydlicek, Guilherme Penedo, Hugo Larcher, Mathieu Morlon, Vaibhav Srivastav, Joshua Lochner, Xuan-Son Nguyen, Colin Raffel, Leandro von Werra, and Thomas Wolf. SmolLM3: smol, multilingual, long-context reasoner. `https://huggingface.co/blog/smollm3`, 2025.

Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Zehui Chen, Haodong Duan, Jiaqi Wang, Yu Qiao, Dahua Lin, and Feng Zhao. Are we on the right way for evaluating large vision-language models? In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 27056–27087. Curran Associates, Inc., 2024. doi: 10.52202/079017-0850. URL `https://proceedings.neurips.cc/paper_files/paper/2024/file/2f8ee6a3d766b426d2618e555b5aeb39-Paper-Conference.pdf`.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri 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, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew 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, 2021. URL https://arxiv.org/abs/2107.03374.

Wanyi Chen, Xiao Yang, Xu Yang, Tianming Sha, Qizheng Li, Zhuo Wang, Bowen Xian, Fang Kong, Weiqing Liu, and Jiang Bian. Agent^2 RL-Bench: Can LLM agents engineer agentic RL post-training? *arXiv preprint arXiv:2604.10547*, 2026. URL https://arxiv.org/abs/2604.10547.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Yaxin Du, Xiyuan Yang, Zhifan Zhou, Wanxu Liu, Zixing Lei, Zimeng Chen, Fenyi Liu, Haotian Wu, Yuzhu Cai, Zexi Liu, Xinyu Zhu, WenHao Wang, Linfeng Zhang, Chen Qian, and Siheng Chen. DataMaster: Data-centric autonomous AI research. *arXiv preprint arXiv:2605.10906*, 2026. URL https://arxiv.org/abs/2605.10906.

Haodong Duan, Junming Yang, Yuxuan Qiao, Xinyu Fang, Lin Chen, Yuan Liu, Xiaoyi Dong, Yuhang Zang, Pan Zhang, Jiaqi Wang, Dahua Lin, and Kai Chen. Vlmevalkit: An open-source toolkit for evaluating large multi-modality models. In *Proceedings of the 32nd ACM International Conference on Multimedia*, MM '24, pp. 11198–11201, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400706868. doi: 10.1145/3664647.3685520. URL https://doi.org/10.1145/3664647.3685520.

Tianrui Guan, Fuxiao Liu, Xiyang Wu, Ruiqi Xian, Zongxia Li, Xiaoyu Liu, Xijun Wang, Lichang Chen, Furong Huang, Yaser Yacoob, Dinesh Manocha, and Tianyi Zhou. Hallusionbench: An advanced diagnostic suite for entangled language hallucination and visual illusion in large vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14375–14385, June 2024.

Shengran Hu, Cong Lu, and Jeff Clune. Automated Design of Agentic Systems. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=t9U3LW7JVX.

Feiyang Kang, Hanze Li, Adam Nguyen, Mahavir Dabas, Jiaqi W. Ma, Frederic Sala, Dawn Song, and Ruoxi Jia. Can generalist agents automate data curation? *arXiv preprint arXiv:2606.04261*, 2026. URL https://arxiv.org/abs/2606.04261.

Zaid Khan, Elias Stengel-Eskin, Jaemin Cho, and Mohit Bansal. Dataenvgym: Data generation agents in teacher environments with student feedback. In *International Conference on Learning Representations*, volume 2025, pp. 53480–53507, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/846d3ef94d8c1a833a62cff05569c851-Abstract-Conference.html. Spotlight.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, pp. 611–626, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702297. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-Harness: End-to-End Optimization of Model Harnesses, 2026. URL https://arxiv.org/abs/2603.28052. arXiv preprint.

Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E. Gonzalez, and Ion Stoica. From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline, 2024. URL https://arxiv.org/abs/2406.11939.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning, 2023. URL https://arxiv.org/abs/2304.08485.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning, 2024a. URL https://arxiv.org/abs/2310.03744.

Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, Kai Chen, and Dahua Lin. Mmbench: Is your multi-modal model an all-around player? In *Computer Vision – ECCV 2024: 18th European Conference, Milan, Italy, September 29–October 4, 2024, Proceedings, Part VI*, pp. 216–233, Berlin, Heidelberg, 2024b. Springer-Verlag. ISBN 978-3-031-72657-6. doi: 10.1007/978-3-031-72658-3_13. URL https://doi.org/10.1007/978-3-031-72658-3_13.

Yuliang Liu, Zhang Li, Mingxin Huang, Biao Yang, Wenwen Yu, Chunyuan Li, Xu-Cheng Yin, Cheng-Lin Liu, Lianwen Jin, and Xiang Bai. Ocrbench: on the hidden mystery of ocr in large multimodal models. *Science China Information Sciences*, 67(12), December 2024c. ISSN 1869-1919. doi: 10.1007/s11432-024-4235-6. URL http://dx.doi.org/10.1007/s11432-024-4235-6.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. In *International Conference on Learning Representations (ICLR)*, 2024.

Andrés Marafioti, Orr Zohar, Miquel Farré, Merve Noyan, Elie Bakouch, Pedro Cuenca, Cyril Zakka, Loubna Ben Allal, Anton Lozhkov, Nouamane Tazi, Vaibhav Srivastav, Joshua Lochner, Hugo Larcher, Mathieu Morlon, Lewis Tunstall, Leandro von Werra, and Thomas Wolf. Smolvlm: Redefining small and efficient multimodal models, 2025. URL https://arxiv.org/abs/2504.05299.

Fanqing Meng, Lingxiao Du, Qiguang Chen, Ziqi Zhao, Haocheng Lu, Mengkang Hu, and Michael Qizhe Shieh. RSIBench-Data: Benchmarking data-centric research for recursive self-improvement. *arXiv preprint arXiv:2607.25886*, 2026. URL https://arxiv.org/abs/2607.25886.

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 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. 48371–48392. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

Yaonan Qu and Meng Lu. Bilevel Autoresearch: Meta-Autoresearching Itself, 2026. URL https://arxiv.org/abs/2603.23420. arXiv preprint.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Ben Rank, Hardik Bhatnagar, Ameya Prabhu, Shira Eisenberg, Karina Nguyen, Matthias Bethge, and Maksym Andriushchenko. PostTrainBench: Can LLM agents automate LLM post-training? *arXiv preprint arXiv:2603.08640*, 2026. URL https://arxiv.org/abs/2603.08640.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Maxime Robeyns, Martin Szummer, and Laurence Aitchison. A Self-Improving Coding Agent. *arXiv preprint arXiv:2504.15228*, 2025. URL https://arxiv.org/abs/2504.15228.

Zhan Shi, Bing He, Yisi Sang, Benoit Dumoulin, and Hanqing Lu. A-Evolve-Training: Autonomous Post-Training of a 30B Model, 2026. URL https://arxiv.org/abs/2606.20657. arXiv preprint.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xiaohai Zhai, Anton Tsitsulin, Robert Busa-Fekete, Alex Feng, Noveen Sachdeva, Benjamin Coleman, Yi Gao, Basil Mustafa, Iain Barr, Emilio Parisotto, David Tian, Matan Eyal, Colin Cherry, Jan-Thorsten Peter, Danila Sinopalnikov, Surya Bhupatiraju, Rishabh Agarwal, Mehran Kazemi, Dan Malkin, Ravin Kumar, David Vilar, Idan Brusilovsky, Jiaming Luo, Andreas Steiner, Abe Friesen, Abhanshu Sharma, Abheesht Sharma, Adi Mayrav Gilady, Adrian Goedeckemeyer, Alaa Saade, Alex Feng, Alexander Kolesnikov, Alexei Bendebury, Alvin Abdagic, Amit Vadi, András György, André Susano Pinto, Anil Das, Ankur Bapna, Antoine Miech, Antoine Yang, Antonia Paterson, Ashish Shenoy, Ayan Chakrabarti, Bilal Piot, Bo Wu, Bobak Shahriari, Bryce Petrini, Charlie Chen, Charline Le Lan, Christopher A. Choquette-Choo, CJ Carey, Cormac Brick, Daniel Deutsch, Danielle Eisenbud, Dee Cattle, Derek Cheng, Dimitris Paparas, Divyashree Shivakumar Sreepathihalli, Doug Reid, Dustin Tran, Dustin Zelle, Eric Noland, Erwin Huizenga, Eugene Kharitonov, Frederick Liu, Gagik Amirkhanyan, Glenn Cameron, Hadi Hashemi, Hanna Klimczak-Plucińska, Harman Singh, Harsh Mehta, Harshal Tushar Lehri, Hussein Hazimeh, Ian Ballantyne, Idan Szpektor, Ivan Nardini, Jean Pouget-Abadie, Jetha Chan, Joe Stanton, John Wieting, Jonathan Lai, Jordi Orbay, Joseph Fernandez, Josh Newlan, Ju yeong Ji, Jyotinder Singh, Kat Black, Kathy Yu, Kevin Hui, Kiran Vodrahalli, Klaus Greff, Linhai Qiu, Marcella Valentine, Marina Coelho, Marvin Ritter, Matt Hoffman, Matthew Watson, Mayank Chaturvedi, Michael Moynihan, Min Ma, Nabila Babar, Natasha Noy, Nathan Byrd, Nick Roy, Nikola Momchev, Nilay Chauhan, Noveen Sachdeva, Oskar Bunyan, Pankil Botarda, Paul Caron, Paul Kishan Rubenstein, Phil Culliton, Philipp Schmid, Pier Giuseppe Sessa, Pingmei Xu, Piotr Stanczyk, Pouya Tafti, Rakesh Shivanna, Renjie Wu, Renke Pan, Reza Rokni, Rob Willoughby, Rohith Vallu, Ryan Mullins, Sammy Jerome, Sara Smoot, Sertan Girgin, Shariq Iqbal, Shashir Reddy, Shruti Sheth, Siim Põder, Sijal Bhatnagar, Sindhu Raghuram Panyam, Sivan Eiger, Susan Zhang, Tianqi Liu, Trevor Yacovone, Tyler Liechty, Uday Kalra, Utku Evci, Vedant Misra, Vincent Roseberry, Vlad Feinberg, Vlad Kolesnikov, Woohyun Han, Woosuk Kwon, Xi Chen, Yinlam Chow, Yuvein Zhu, Zichuan Wei, Zoltan Egyed, Victor Cotruta, Minh Giang, Phoebe Kirk, Anand Rao, Kat Black, Nabila Babar, Jessica Lo, Erica Moreira, Luiz Gustavo Martins, Omar Sanseviero, Lucas Gonzalez, Zach Gleicher, Tris Warkentin, Vahab Mirrokni, Evan Senter, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, Yossi Matias, D. Sculley, Slav Petrov, Noah Fiedel, Noam Shazeer, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Jean-Baptiste Alayrac, Rohan Anil, Dmitry, Lepikhin, Sebastian Borgeaud, Olivier Bachem, Armand Joulin, Alek Andreev, Cassidy Hardin, Robert Dadashi, and Léonard Hussenot. Gemma 3 technical report, 2025. URL https://arxiv.org/abs/2503.19786.

Dingzirui Wang, Xuanliang Zhang, Keyan Xu, Qingfu Zhu, and Wanxiang Che. CurateEvo: Data-curation evolving for agentic post-training. *arXiv preprint arXiv:2607.06140*, 2026. URL https://arxiv.org/abs/2607.06140.

Jingtan Wang, Xiaoqiang Lin, Rui Qiao, Pang Wei Koh, Chuan-Sheng Foo, and Bryan Kian Hsiang Low. NICE data selection for instruction tuning in LLMs with non-differentiable evaluation metric. 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. 63662–63689.

PMLR, 13–19 Jul 2025a. URL https://proceedings.mlr.press/v267/wang25bm.html.

Wenyi Wang, Piotr Piekos, Li Nanbo, Firas Laakom, Yimeng Chen, Mateusz Ostaszewski, Mingchen Zhuge, and Jürgen Schmidhuber. Huxley-Godel Machine: Human-Level Coding Agent Development by an Approximation of the Optimal Self-Improving Machine. *arXiv preprint arXiv:2510.21614*, 2025b. URL https://arxiv.org/abs/2510.21614.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13484–13508, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.754. URL https://aclanthology.org/2023.acl-long.754/.

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.

Minghao Yan, Bo Peng, Benjamin Coleman, Ziqi Chen, Zhouhang Xie, Shuo Chen, Zhankui He, Noveen Sachdeva, Isabella Ye, Weili Wang, Chi Wang, Ed H. Chi, Fernando Pereira, Wang-Cheng Kang, Derek Zhiyuan Cheng, and Beidou Wang. PACEvolve: Enabling Progress-Aware Consistent Evolution, 2026. URL https://arxiv.org/abs/2601.10657. arXiv preprint.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. Gödel agent: A self-referential agent framework for recursively self-improvement. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27890–27913, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1354. URL https://aclanthology.org/2025.acl-long.1354/.

Weihao Yu, Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Zicheng Liu, Xinchao Wang, and Lijuan Wang. Mm-vet: evaluating large multimodal models for integrated capabilities. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Zhaojian Yu, Penghao Yin, Shuzheng Gao, Shilin He, Kai Cai, and Xiao-Ping Zhang. AutoTrainess: Teaching language models to improve language models autonomously. *arXiv preprint arXiv:2606.31551*, 2026. URL https://arxiv.org/abs/2606.31551.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, Cong Wei, Botao Yu, Ruibin Yuan, Renliang Sun, Ming Yin, Boyuan Zheng, Zhenzhu Yang, Yibo Liu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9556–9567, June 2024.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. TextGrad: Automatic Differentiation via Text. *arXiv preprint arXiv:2406.07496*, 2024. URL https://arxiv.org/abs/2406.07496.

Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. Self-Taught Optimizer (STOP): Recursively Self-Improving Code Generation. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=1gkePTsAWf.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Tjarko Lange, and Jeff Clune. Darwin Godel Machine: Open-Ended Evolution of Self-Improving Agents. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=pUpzQZTvGY.

Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. Hyperagents, 2026b. URL https://arxiv.org/abs/2603.19461. arXiv preprint.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating Agentic Workflow Generation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=z5uVAKwmjf.

Jiayi Zhang, Yongfeng Gu, Jianhao Ruan, Maojia Song, Yiran Peng, Zhiguang Han, Jinyu Xiang, Zhitao Wang, Caiyin Yang, Yixi Ouyang, Bang Liu, Chenglin Wu, and Yuyu Luo. Harnessing Agentic Evolution, 2026c. URL https://arxiv.org/abs/2605.13821. arXiv preprint.

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Zhuo Li, Yujie Zheng, Weinan Zhang, Ying Wen, Zhiyu Li, Feiyu Xiong, Yutao Qi, Bo Tang, and Muning Wen. MemRL: Self-Evolving Agents via Runtime Reinforcement Learning on Episodic Memory, 2026d. URL https://arxiv.org/abs/2601.03192. arXiv preprint.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2025, 2025.

Adam Zweiger, Jyothish Pari, Han Guo, Ekin Akyürek, Yoon Kim, and Pulkit Agrawal. Self-Adapting Language Models. *arXiv preprint arXiv:2506.10943*, 2025. URL https://arxiv.org/abs/2506.10943.

A Implementation Details

This appendix follows the four-layer architecture introduced in Section 3.2.

A.1 Agent Layer

A.1.1 Role Prompts and Tool Scopes

The following prompts define the responsibilities, expected inputs, operating rules, and handoff formats of the four roles introduced in Section 3.3. Task-specific objectives, plans, resource constraints, and environment-data references are supplied separately during execution. Plans are passed as structured artifacts. Each handoff identifies the plan version, relevant execution and artifact references, and the next requested action, accompanied by a concise explanation. When a plan is revised, completed executions retain references to the version under which they ran.

Leader Prompt

You are the DataOS Leader. You plan experiments using the available system capabilities and select task outputs. Delegate experiment execution to the Worker and diagnosis-driven system revision decisions to the Evolver.

Your input includes the task objective, permitted resources, evaluation protocol, remaining budget, current system configuration, and references to accumulated environment data.

Leader Prompt (continued)

Tool scope: Use `trace` and `inspect` to examine resources and execution history. Send Task Plans to the Worker, improvement requests to the Evolver, and plans and decisions to the Maintainer for recording.

Your work:

1. Review the available datasets, model checkpoints, operators, training and evaluation methods, and execution history. Identify what has been achieved and what remains unresolved.
2. Choose the next experiment using the relevant artifacts and recorded outcomes. Specify its objective and why it is useful.
3. Construct a Task Plan comprising Data, Training, and Eval Specs that identify the starting components, operations, dependencies, required resources, and expected observations.
4. Provide the structured Task Plan and its reference to the Worker. Use the returned execution records and artifact references to decide subsequent experiments.
5. When failures, missing capabilities, or other evidence motivate a system revision, send the Evolver a concrete issue, supporting references, and an improvement objective.
6. When the task concludes, select the output artifact according to the task-defined evaluation and selection rules.

Planning rules:

- Reference the component versions and records supporting each decision.
- Distinguish intended operations from completed executions.
- Use the available system capabilities when planning experiments; direct proposed capability changes to the Evolver.
- Task experiments may produce new datasets and checkpoints. Delegate revision hypotheses and system-variant adoption decisions to the Evolver.
- Keep plans within the task constraints and remaining budget.

Provide the structured plan or its accessible reference, followed by a concise handoff:

```
Task state:
Relevant environment data:
Task plan reference:
Expected observations:
Issue and improvement objective:
Handoff:
```

Under Handoff, specify the requested Worker execution, the issue for the Evolver, the selected task output, or the prerequisite preventing further progress. Mark sections that do not apply as not applicable.

Evolver Prompt

You are the DataOS Evolver. You are responsible for diagnosing limitations, planning revisions to datasets, models, and the harness, and selecting revisions based on their evaluated outcomes. Delegate candidate implementation and evaluation to the Worker.

Your input includes an improvement objective or unresolved issue, the retained system variants, associated environment-data references, task constraints, and any results from previous revision attempts.

Tool scope: Use the Environment Data Layer interfaces `trace`, `inspect`, and `analyze` to examine and analyze evidence directly. Delegate candidate implementation, evaluation, and application to the Worker; send plans and decisions to the Maintainer.

Your work:

1. Review available evidence and initialize an Evolution Plan with an Analysis Spec defining the question, source references, inspections or computations, and expected observations. Reuse existing diagnostic results when applicable.
2. Analyze the referenced environment data to produce a diagnosis identifying the responsible component or operation and explaining the observed limitation.
3. Select a parent from retained evolution-tree variants, revisiting earlier configurations when supported by evidence. Reference artifacts and methods reused from other branches.
4. Complete the Revision Spec with the parent, evidence, changes, required operations, resources, and expected effects, and the Validation Spec with a Task Plan reference, comparison reference, diagnostic observations, and selection criteria.
5. Send the updated plan and its reference to the Worker for implementation and evaluation. Compare candidates with their parent or the task-defined reference under the same evaluation protocol.

Evolver Prompt (continued)

6. Examine task performance and diagnostic observations. Select a revision or retain the current configuration. Send the decision to the Worker for application and to the Maintainer for recording.

Evidence rules:

- Distinguish observed differences from proposed explanations.
- Interpret outcomes using the versions and conditions that produced them.
- Extend the analysis of relevant environment data when a diagnosis remains unresolved.
- Refine the plan as evidence arrives, retaining the plan versions associated with completed diagnostics and candidate evaluations.
- Preserve evidence from unsuccessful attempts for subsequent exploration.

Provide the current structured plan or its accessible reference, followed by a concise handoff:

```
Improvement objective:
Current stage:
Evidence and diagnosis:
Parent variant:
Evolution plan reference:
Evaluation assessment:
Selection decision:
Handoff:
```

Mark unresolved plan fields and decisions as pending. Under Handoff, identify the relevant plan version and specify the implementation, evaluation, or application work required from the Worker.

Worker Prompt

You are the DataOS Worker. You execute Task Plans and implement, evaluate, and apply system revisions specified by Evolution Plans. Task-level decisions belong to the Leader; revision selection belongs to the Evolver.

Your input includes the assigned plan version, referenced resources and harness configuration, permitted changes, runtime interfaces, and execution constraints.

Tool scope: Use environment-data access interfaces to resolve inputs. Invoke `dataos-data`, `dataos-train`, and `dataos-eval`, together with Runtime Layer Data, Model, and Harness modification interfaces, within the assigned scope.

Your work:

1. Resolve the assigned plan version, input artifacts, and component versions. Report missing prerequisites or conflicting references before dependent execution.
2. Execute the specified data-processing, training, and evaluation operations in the designated local, Slurm, or remote SSH environment. Track execution or job references and report completion, failures, or unfinished work.
3. For evolution requests, implement the planned Data, Model, or Harness changes and execute the Validation Spec. Edit role prompts or implement operators and pipelines when required by the Revision Spec.
4. Record the originating plan version, actual inputs and component versions, configurations, operations, outputs, observations, status, and errors or deviations.
5. Return execution and artifact references to the requesting role and supply the same records to the Maintainer. Apply a selected revision when directed by the Evolver.

Execution rules:

- Keep execution within the assigned objective and permitted changes.
- Report deviations, failed operations, and partial results explicitly.
- Base reported measurements on completed executions.
- Distinguish candidate implementation and evaluation from application of the Evolver's selection decision.

Begin the handoff with:

```
Execution status:  completed | partial | failed | blocked
```

Then provide:

```
Plan reference and assigned request:
Executed operations:
Actual inputs and configurations:
```

Worker Prompt (continued)

```
Artifacts and observations:
Errors, deviations, and unresolved issues:
Execution and job references:
```

Maintainer Prompt

You are the DataOS Maintainer. You plan and perform updates to persistent environment data, connecting artifacts, executions, component versions, and decisions. Experiment planning belongs to the Leader, and revision selection belongs to the Evolver.

Your input includes initial resources, new artifacts and diagnostic results, plan versions and configurations, agent handoffs, execution records, selection decisions, and the existing environment data.

Tool scope: Use Environment Data Layer access interfaces to review existing objects and its registration, association, and versioning interfaces to apply a Maintenance Plan. Return persistent identifiers and relationships for subsequent access.

Your work:

1. Compare incoming resources, plans, artifacts, and records with existing environment data. Construct a Maintenance Plan comprising Registration, Association, and Versioning Specs for the required updates.
2. Apply the plan through Environment Data Layer interfaces. Register initial resources and incorporate plans, configurations, execution records, diagnostic outputs, handoffs, and decisions at task and evolution boundaries.
3. Link each execution to its originating plan version, actual inputs and component versions, configurations, outputs, observations, and status. Preserve agent-invocation references identifying the role and harness configuration used.
4. Preserve dependency chains connecting evaluations to checkpoints, training runs, datasets, and processing methods.
5. Register component revisions as new versions while retaining earlier artifacts and results. Keep large payloads in their native formats and make them accessible through references.
6. Record task-output selections and system-variant adoption decisions with their supporting evaluations. For each evolution attempt, retain the parent, intervention, resulting variant when available, and evaluated outcomes.
7. Retain unresolved failures and missing capabilities with their supporting records for subsequent planning and evolution.

Maintenance rules:

- Associate outcomes with the conditions actually used during execution.
- Keep planned operations distinct from realized operations.
- Preserve historical records when later versions become available.
- Mark missing provenance explicitly instead of inferring unsupported relationships.
- Record execution outcomes and selection decisions as separate facts.

Begin the handoff with:

```
Update status:  complete | partial | blocked
```

Then provide:

```
Maintenance plan reference:
Registered artifacts and records:
Execution and provenance links:
Version and dependency updates:
Evolution-tree updates:
Unresolved issues:
Missing information:
```

Provide the structured Maintenance Plan or its accessible reference, together with identifiers of the registered objects and updated relationships, so that the Leader and Evolver can retrieve the maintained environment data.

A.1.2 UNIFIED AGENT ADAPTER

DATAOS wraps agent backends behind a shared asynchronous interface. Each invocation receives the role prompt and tool settings resolved from its assigned harness version, an execution environment,

and a time budget, and returns a standardized execution result. Backend-specific adapters translate model, reasoning-effort, and tool settings into native CODEX CLI or CLAUDE CODE arguments. A shared runner stores each invocation prompt separately, launches the selected CLI in the designated workspace, and records its output and termination status under the specified timeout. Output parsers extract the agent response from the backend's structured event stream while retaining raw traces, elapsed time, and error information. Invocation records identify the role, harness configuration, and originating plan version when applicable, alongside the stored prompt and execution trace. After a harness revision is adopted and applied, subsequent invocations resolve prompts and tool settings from the updated version. This interface keeps role orchestration independent of backend invocation and supplies execution records for environment-data maintenance.

A.2 PIPELINE LAYER

The Pipeline Layer represents the decisions of the Leader, Evolver, and Maintainer through three structured, editable plans. Each plan specifies an objective and constraints, with constituent specifications describing inputs, operations, settings, dependencies, and expected outputs. Plans are retained as referenced artifacts, and handoffs identify the plan version to execute together with the requested operation. Revising a plan preserves earlier versions and their execution links. Realized executions are recorded separately as environment data.

Task Plan. The Leader specifies an experiment through **(1) Data Spec**, which identifies source datasets and versions, selected fields or subsets, processing operators and parameters, operation dependencies, and the output dataset format; **(2) Training Spec**, which identifies the starting checkpoint, training data, method, objective, hyperparameters, backend, compute requirements, and expected checkpoints and training records; and **(3) Eval Spec**, which identifies the evaluated artifact, evaluation datasets, evaluator, inference settings, metrics, aggregation rules, and outputs to retain under the task-defined protocol. Artifact references connect the three specifications, allowing an experiment to reuse existing datasets or checkpoints and execute only the remaining stages.

Evolution Plan. The Evolver specifies system improvement through **(1) Analysis Spec**, which defines the diagnostic question, environment-data references, inspections or computations, and expected measurements and examples supporting a diagnosis; **(2) Revision Spec**, which identifies the parent variant, supporting evidence, proposed changes to data, models, or the harness, required resources, and expected candidate artifacts; and **(3) Validation Spec**, which references a Task Plan for assessing the candidate and specifies the comparison reference, diagnostic observations, and selection criteria under a shared evaluation protocol. The Evolver first defines the Analysis Spec and uses its results to complete the Revision and Validation Specs, connecting the diagnosis to an intervention and an assessment of its predicted effect.

Maintenance Plan. The Maintainer compares incoming resources and execution outputs with existing environment data and specifies the necessary updates through **(1) Registration Spec**, which identifies resources, artifacts, plans, configurations, diagnostic results, handoffs, and execution and decision records to register, with their metadata and access references; **(2) Association Spec**, which links artifacts, executions, configurations, originating plan versions, and decisions to preserve provenance and dependencies; and **(3) Versioning Spec**, which identifies new component versions and their relationships to historical versions, parent variants, and modification operations while retaining earlier artifacts and results. The Maintainer applies these updates through Environment Data Layer interfaces, making the resulting records accessible to subsequent planning and analysis.

A.3 RUNTIME LAYER

Execution interfaces. The Runtime Layer exposes three tools for executing pipeline specifications: **(1) `dataos-data`** wraps the operator library to execute the processing operations and settings in a Data Spec, including filtering, sampling, transformation, and mixing; **(2) `dataos-train`** wraps training frameworks including LLaMA-Factory and `verl`, translating a Training Spec into the selected framework's configuration and invocation; and **(3) `dataos-eval`** wraps `lm-eval`, OpenCompass, and custom evaluators to execute an Eval Spec and return measurements and evaluation outputs under the task-defined protocol. These tools hide framework-specific operations and

support local execution, Slurm scheduling, and remote execution over SSH, with the selected backend handling process launch, job submission, or remote invocation. Runtime results provide execution references, job identifiers where applicable, status, artifact and log references, and error information, allowing the Worker to track jobs and report incomplete or failed work. Table 4 summarizes the built-in operator library exposed through `dataos-data`.

Table 4: Built-in training-data operator statistics by category. Counts exclude run-local plugins and abstract extension classes.

Category	Function	Count
Mappers	Schema and supervision formatting	10
Evaluators	Scoring, representations, and image metadata	10
Selectors	Sampling, filtering, and subset selection	27
Mixers	Combining data sources	4
Reweighters	Assigning sample weights	1
Total		**52**

Modification interfaces. The runtime also provides interfaces for modifying the three system components identified by a Revision Spec: **(1) Data** modification produces revised datasets by changing their contents or composition through the operator library and `dataos-data`; **(2) Model** modification updates model parameters from a referenced checkpoint through the training methods supported by `dataos-train`; and **(3) Harness** modification edits role prompts, adds or extends operators, and implements new pipelines. New operators are exposed through `dataos-data`, while new pipelines compose the available tools, allowing revised capabilities to reuse the framework wrappers and execution backends. Both execution and modification operations retain actual inputs, configurations, output artifacts, and status for environment-data collection and version maintenance.

A.4 Environment Data Layer

Artifact and record schemas. The layer organizes environment data into **(1) Resources and artifacts**, including datasets, model checkpoints, harness components such as prompts and operators, and diagnostic outputs; **(2) Plans and configurations**, describing intended task, evolution, and maintenance operations; and **(3) Execution and decision records**, covering agent invocations, experiments, data processing, training, evaluation, analysis, and selection outcomes. Artifact descriptions include persistent identifiers, types, versions, storage references, and links to their producing executions or initial resource registrations. Versioned plans identify intended inputs, operations, and dependencies, while execution records retain actual component versions, configurations, inputs, outputs, observations, and status. Agent-invocation records identify the role and harness configuration; handoffs reference the originating plan and requested next action. Selection decisions link chosen task artifacts or system variants to their supporting evaluation records. Datasets, checkpoints, and logs retain their native formats and are accessed through the references in these shared descriptions.

Collection and access interfaces. Runtime operations expose actual inputs, configurations, generated artifacts, measurements, logs, and status through the shared recording interface. The agent adapter supplies invocation traces and responses. At task and evolution boundaries, the Maintainer applies its Maintenance Plan to incorporate these records together with plans, handoffs, and decisions, associate the objects, and update version relationships, retaining failed and partial executions alongside successful ones. For access, `trace` takes a component or execution reference and follows recorded dependencies to retrieve related records; `inspect` reads a referenced artifact or selected execution-trace segment. The Evolver uses `analyze` to compare and summarize referenced environment data, including distributions, errors, and outcomes, and derives a diagnosis from these analyses. Analysis outputs and diagnoses are collected with their source references, making the results available to later planning.

Version and dependency management. Revisions to datasets, models, and harness components are registered as new versions while earlier artifacts and execution records are retained. Dependencies connect each result to the versions and operations that produced it: an evaluation references its

checkpoint, the checkpoint references its training run, and that run references its input data and associated processing records. Each evolution-tree node references a harness, model, and dataset configuration, and each revision edge identifies the parent and applied intervention. Evolution-attempt records associate these references with candidate artifacts, evaluation outcomes, and selection decisions. Artifacts and methods from other branches can be reused through explicit references that preserve their original provenance, supporting both historical comparison and subsequent execution.

B Experimental Details

In this section, we provide additional details on the benchmark settings, experimental protocols, and analysis experiments used to evaluate DataOS.

B.1 PostTrainBench

B.1.1 Tasks and Target Models

PostTrainBench (PTB) (Rank et al., 2026) evaluates autonomous post-training, where an agent constructs training data and selects training strategies to improve a target model on a specified benchmark. We evaluate four target models: `Qwen3-1.7B-Base`, `Qwen3-4B-Base` Yang et al. (2025), `SmolLM3-3B-Base` Bakouch et al. (2025), and `Gemma-3-4B-PT` Team et al. (2025). The evaluation covers seven benchmarks: GSM8K Cobbe et al. (2021), AIME 2025 Zhang & Math-AI (2025), HumanEval Chen et al. (2021), BFCL Patil et al. (2025), GPQA Rein et al. (2024), HealthBench Arora et al. (2025), and Arena-Hard Writing Li et al. (2024). In the main experiment, each target–benchmark pair is optimized in a separate run, yielding 28 task configurations per system.

B.1.2 Candidate Data Pool

Source selection. We construct a fixed candidate pool of publicly available datasets for the PTB experiments. Candidate datasets are identified on Hugging Face Hub using task-specific search terms, community popularity, and domain knowledge. We screen repository metadata and dataset descriptions for accessibility and potential derivation from the evaluation benchmarks. Row-level overlap filtering is described in Section B.1.3.

Pool composition. After filtering, the pool contains 103 data sources comprising 68,207,461 examples. It covers the seven target domains and includes additional datasets for translation, classification, summarization, sentiment analysis, and other tasks. Table 5 summarizes the source distribution. These domain groupings describe how datasets were collected; they do not restrict which sources can be used for a given task.

Data representation. Datasets retain their original columns, and no per-source row cap is imposed during pool construction. Each source is accompanied by metadata describing its upstream repository, domain, size, and available fields. Field mappings identify prompt, answer, option, and system-message columns where recognizable. The agent is responsible for selecting and transforming these heterogeneous records into training data.

B.1.3 Decontamination

To reduce the risk that overlap between training and evaluation data inflates benchmark scores, we decontaminate the PTB candidate pool against all seven evaluation benchmarks.

Reference construction. We construct a reference set from the evaluation datasets listed in Table 6. For each evaluation record, we concatenate all text fields, including questions, answers, options, and text within nested structures. We normalize the resulting text by lowercasing, replacing punctuation and other non-word characters with spaces, and collapsing repeated whitespace. Tokens are obtained by whitespace splitting.

Overlap detection. We extract all contiguous 13-token spans from the normalized evaluation text and hash each span using 64-bit BLAKE2b. For a nonempty record containing fewer than 13 tokens,

Table 5: Composition of the PTB candidate data pool after decontamination. Sources are grouped by the domain used during data collection; these groups do not impose task-specific restrictions on data use.

Domain	Sources	Examples
School mathematics	10	41,055,070
Competition mathematics	8	1,599,497
Code	12	2,397,973
Function calling	9	444,900
Science	13	3,351,542
Medicine	11	886,316
Writing	10	7,175,350
Other domains	30	11,296,813
Total	**103**	**68,207,461**

we hash its complete normalized text instead. Candidate data undergo the same text extraction, normalization, and hashing procedure. A candidate record is flagged if any of its hashes matches a reference hash from any of the seven benchmarks.

Filtering criteria. We remove flagged records and exclude an entire data source if more than 3% of its records are flagged or fewer than 1,000 records remain after filtering. For retained sources, the surviving records preserve their original fields. The resulting pool is summarized in Table 5.

Scope. This procedure detects lexical overlap between the candidate training data and the evaluation benchmarks. It may miss paraphrased content and may remove unrelated records that share common text. It does not assess contamination in the target models' pretraining data. This filtering procedure applies to the PTB candidate pool; Curation-Bench follows its own benchmark protocol.

B.1.4 Training and Evaluation Protocol

Training interventions. The agent constructs training data from the fixed candidate pool and selects training strategies to improve the target model. The evaluation procedure is fixed and cannot be modified by the agent. Each run must return a complete Hugging Face model checkpoint, with any trained adapters merged into the model. Resource budgets and endpoint selection are described in Sections B.3.2 and B.3.3.

Benchmark variants. Table 6 summarizes the benchmark variants, reference-set sizes, evaluation metrics, and generation settings. We use the main/test split of GSM8K, the test splits of AIME 2025 and HumanEval, the `exec_simple` subset of BFCL, and GPQA-main. The provider's `train` split of GPQA-main is used exclusively for evaluation. HealthBench uses a 245-example rubric-evaluation reference set, and Arena-Hard Writing uses the 250-example creative-writing subset of Arena-Hard-v2.0. Endpoint selection and final assessment coverage are addressed in Section B.3.3.

Generation and judging. For AIME 2025, we generate eight responses per problem with temperature 0.6, top-p 0.95, and a maximum output length of 16,000 tokens. We compute the mean exact-match accuracy across the eight responses for each problem, then average across the evaluated problems and report the result as a percentage. A full evaluation covers all 30 problems. HealthBench and Arena-Hard Writing are evaluated using `qwen3.5-plus` as the judge model.

Model formatting. We use model-family-specific templates for Qwen3, SmolLM3, and Gemma3, with matched training and evaluation formats. For BFCL, Gemma3 uses a dedicated tool-calling template.

Table 6: PTB benchmark variants and evaluation settings. Sizes refer to the reference sets used for evaluation and decontamination, rather than the coverage of every checkpoint evaluation. Token limits specify the maximum number of generated tokens per response.

Benchmark	Dataset / split	Size	Metric and generation settings
GSM8K	`openai/gsm8k`, main/test	1,319	Numeric-answer accuracy; zero-shot, one response per problem.
AIME 2025	`math-ai/aime25`, test	30	Mean exact-match accuracy over 8 responses per problem; temperature 0.6, top-p 0.95, 16,000-token limit.
HumanEval	`openai/openai_humaneval`, test	164	Executable-test pass@1; zero-shot, one response per problem, 4,000-token limit.
BFCL	`exec_simple` subset	100	Tool-call execution evaluation; temperature 0.6, top-p 0.95, 16,000-token limit.
GPQA	`Idavidrein/gpqa`, `gpqa_main/train`	448	Multiple-choice accuracy; zero-shot, temperature 1.0, 16,000-token limit.
HealthBench	Rubric-evaluation reference set	245	Rubric-based judging; 16,384-token limit.
Arena-Hard Writing	Arena-Hard-v2.0 creative-writing subset	250	Judge-based writing evaluation; 16,384-token limit.

B.2 CURATION-BENCH

Curation-Bench (Kang et al., 2026) evaluates autonomous data selection for vision-language model fine-tuning. An agent selects a fixed-size training subset from a fixed candidate pool and uses evaluation feedback to improve its selection strategy over successive iterations.

B.2.1 TASKS AND TARGET MODELS

We evaluate three target models: LLaVA-1.5-7B Liu et al. (2024a), Qwen2.5-VL-3B-Instruct Bai et al. (2025), and the 2.2B SmolVLM-Base Marafioti et al. (2025) model. Each target model is evaluated in a separate curation session. The agent's objective is to select 10,000 training examples that improve the target model's performance across eight multimodal benchmarks. The training recipe and initial target-model checkpoint are fixed within each session, so the agent searches over data subsets.

B.2.2 CANDIDATE DATA POOL

We use LLaVA-665K Liu et al. (2024a) as the fixed candidate pool, containing approximately 665,000 multimodal instruction-tuning examples. At each iteration, the agent selects exactly 10,000 examples from this pool to form the training dataset. The candidate pool and the required subset size remain unchanged throughout the session, while the selection strategy can be revised using feedback from earlier iterations. Each selected subset is submitted as a loadable dataset with `images` and `texts` columns.

B.2.3 TRAINING AND EVALUATION PROTOCOL

Iterative curation. Following Kang et al. (2026), each session consists of 10 iterations of data selection, training, and evaluation. At each iteration, the selected 10,000-example subset is used to fine-tune a fresh copy of the target's original pre-curation checkpoint. Evaluation scores are then returned to the agent to guide the next selection. Target-model weights are reset between iterations; the selection strategy is refined using accumulated feedback. The reported endpoint is selected according to Section B.3.3.

Training recipe. We follow the model-specific recipes in Appendix B.4 of Kang et al. (2026). Each selected subset is used for one epoch of fine-tuning on one A100 GPU. LLaVA-1.5-7B uses a learning rate of 2×10^{-5}, a per-device batch size of 1, and 16 gradient accumulation steps, with the vision tower frozen. Qwen2.5-VL-3B-Instruct fine-tunes the entire model with a learning rate of 10^{-5}, a per-device batch size of 1, and 8 gradient accumulation steps. SmolVLM-Base trains the language model and multimodal projector with a learning rate of 10^{-5}, a per-device batch size of 16, and 1

Table 7: Evaluation metrics for Curation-Bench.

Benchmark	Reported metric
HallusionBench	Mean of aAcc, fAcc, and qAcc
LLaVABench	Relative score
OCRBench	Answer-match score divided by 10
MMMU	Overall score on `MMMU_DEV_VAL`
MathVista	Score on `MathVista_MINI`
MMStar	Overall multiple-choice score
MMBench	Overall score on the English development set
MMVet	Judge-based overall score

gradient accumulation step, while freezing the vision tower. All three recipes use AdamW, cosine learning-rate scheduling, a warmup ratio of 0.03, and bfloat16 precision. These hyperparameters remain fixed across selection iterations.

Evaluation. Each fine-tuned checkpoint is evaluated on HallusionBench Guan et al. (2024), LLaVABench Liu et al. (2023), OCRBench Liu et al. (2024c), MMMU Yue et al. (2024), MathVista Lu et al. (2024), MMStar Chen et al. (2024), MMBench Liu et al. (2024b), and MMVet Yu et al. (2024). We use VLMEvalKit Duan et al. (2024) with vLLM Kwon et al. (2023) enabled and four API workers; `Qwen3.5-27B` Qwen Team (2026) serves as the judge model for judge-based scoring. Table 7 summarizes the reported metrics.

B.3 Execution and Reporting Protocols

B.3.1 Researcher and System Configurations

Research agents. We instantiate DataOS with two agent harnesses: Codex CLI powered by GPT-5.6 Sol and Claude Code powered by Opus 5. Both configurations use medium reasoning effort. For each researcher model, we compare DataOS with the corresponding native agent at medium reasoning effort (native-medium). We also evaluate native-max as a higher-reasoning-effort reference. We additionally compare against DataMaster (Du et al., 2026) and AutoTrainess (Yu et al., 2026), two specialized systems for automated model improvement.

Reference baselines. For both benchmarks, we evaluate the initial target-model checkpoints before task-specific fine-tuning. For PTB, we additionally include the corresponding official instruction-tuned models as performance references; their training data and compute are not matched to our experimental budgets. For Curation-Bench, we include a random-selection baseline that samples 10,000 examples from the same candidate pool using random seed 123 and follows the same model-specific training and evaluation protocol.

B.3.2 Resource Budgets and Time Accounting

Each main PTB run is allocated one H100-80GB GPU and eight CPU cores, with a 10-hour time budget. The budget includes time spent on agent execution, target-model training, and evaluation, while excluding scheduler queue time.

Curation-Bench uses a budget of 10 curation–training–evaluation iterations per target model. Each iteration follows the fixed training recipe described in Section B.2.3.

Resource allocations and budgets for the analysis experiments are specified in Section B.4.

B.3.3 Endpoint Selection

We determine the reported checkpoint for each run according to the benchmark-specific rules below.

PostTrainBench. At the end of each run, the agent is required to explicitly report the path to its selected model checkpoint. We load the checkpoint from the submitted path for evaluation. If

the agent run fails or the submitted path does not exist, we mark the run as failed and report the corresponding base model's score as its outcome.

Curation-Bench. We select the checkpoint with the highest normalized mean score across the eight evaluation benchmarks among the 10 curation iterations. All eight reported scores are taken from this checkpoint.

B.3.4 Reporting Conventions

We conduct three runs for each experimental configuration and report the mean scores across the three runs.

Score differences. In the main results tables, Δ denotes the score difference between DataOS and native-medium using the same researcher model. Differences are reported on the corresponding score scale, rather than as relative percentage improvements. Reference conditions for ablation and transfer comparisons are specified in their respective tables. Aggregates and differences are computed before rounding for display.

Failed runs and missing results. Failed PTB runs contribute the corresponding base-model scores to the aggregates, following Section B.3.3. Unavailable results are distinguished from failed runs and are not assigned zero scores. We report an aggregate only when all required component scores are available: seven benchmarks and four target models for PTB, and eight benchmarks and three target models for Curation-Bench.

B.4 Analysis Experiment Settings

We conduct additional experiments on PTB to examine the effects of increased computational budgets, the contributions of individual system components, and the transfer of accumulated research state across tasks and target models. We also define trajectory-level metrics to characterize the search process. Unless otherwise specified, these experiments follow the benchmark protocols in Section B.1 and the execution and reporting rules in Section B.3.

B.4.1 Multi-round Joint Optimization

We examine whether DataOS can improve a single target model across multiple tasks through successive training interventions. This experiment jointly optimizes performance on all seven PTB benchmarks under an increased computational budget.

Experimental setup. We use Qwen3-4B-Base as the target model and compare DataOS–Codex with native Codex CLI. Both systems use medium reasoning effort. Each run is allocated four H100 GPUs and a 20-hour budget, including agent execution, target-model training, and evaluation, while excluding scheduler queue time.

Optimization objective. The objective is the equally weighted mean score across the seven PTB benchmarks:

$$J(\theta) = \frac{1}{7} \sum_{b=1}^{7} x_b(\theta), \tag{2}$$

where $x_b(\theta)$ denotes the score of checkpoint θ on benchmark b, expressed on the 0–100 scale. All seven scores are obtained from the same checkpoint. The agent can perform successive training interventions to improve this joint objective.

B.4.2 Mechanism Ablations

We conduct ablation experiments to examine the contributions of structured access to experimental history, reusable data-processing operators, and harness evolution.

Experimental setup. All ablations use DATAOS–Codex with Qwen3-4B-Base. Each of the seven PTB benchmarks is optimized in a separate run under the standard 10-hour budget. The full-system reference is the corresponding DATAOS–Codex configuration in Table 8. We compare this reference with four ablated variants, holding the remaining system components and per-task resource budgets fixed.

Ablation variants.

Current-record feedback. We replace the System Data layer with direct feedback containing only the current iteration's execution trajectory and evaluation scores. Records from earlier iterations are not included in this feedback.

Sequential-history feedback. We replace the System Data layer with an unstructured chronological sequence of accumulated experimental records. This variant retains historical feedback but removes explicit relations between recorded objects and the structured query interfaces.

Without the training-data operator library. We remove the provided data-processing operators. The agent can still implement and execute its own data-processing code under the same data-access rules.

Without harness evolution. We freeze the initial harness throughout the run. The agent can continue to select training data and training strategies through the available interfaces, but cannot revise the harness itself.

Reporting. We conduct three runs per configuration and report the mean score for each benchmark. The overall score is the unweighted mean across the seven benchmarks, each optimized independently. Results are presented in Table 2.

B.4.3 INHERITANCE AND TRANSFER

We examine whether research state accumulated in an earlier run can improve subsequent optimization when the target model is initialized from its original base checkpoint. We consider inheritance within the same setting, transfer across benchmarks, and transfer across target models.

Experimental protocol. All experiments use DATAOS–Codex. A source run first optimizes a target model on a source benchmark. A subsequent destination run inherits accumulated environment data and adopted harness revisions, while target-model weights are initialized from the destination model's original base checkpoint. When a source system is reused for multiple destinations, each destination starts from an independent copy of the same source endpoint.

The inherited environment data include plans and configurations, execution and decision records, diagnostic and evaluation results, and reusable datasets and recipes, together with their provenance, dependency, and version links. Records of both successful and unsuccessful attempts are retained. The destination harness incorporates adopted revisions to role prompts, operators, and pipelines. Source-model configurations and evaluation records remain available as historical evidence; source-model weights are excluded from the transferred resources. Each destination starts fresh agent sessions and uses the inherited evidence and resources to construct new Task and Evolution Plans for its own objective, with system evolution continuing during the run.

Same-setting inheritance. We use Qwen3-4B-Base on GSM8K for both the source and destination runs. The destination inherits the source research state but resets the target-model weights. This setting examines the value of accumulated research state when the model and task remain unchanged.

Cross-benchmark transfer. We fix the target model to Qwen3-4B-Base and change the benchmark between the source and destination runs. We evaluate three directed transfers:

- GSM8K $\rightarrow$ AIME 2025;
- HumanEval $\rightarrow$ BFCL;
- GSM8K $\rightarrow$ HumanEval.

Cross-model transfer. We fix the benchmark to GSM8K and change the target model between the source and destination runs. We evaluate three directed transfers:

- Qwen3-1.7B-Base → Qwen3-4B-Base;
- Qwen3-4B-Base → SmolLM3-3B-Base;
- Qwen3-4B-Base → Gemma-3-4B-PT.

Budgets and comparisons. Source and destination runs each receive a 10-hour budget. The destination budget begins after the inherited system state has been loaded. For each destination setting, the no-inheritance reference is the corresponding 10-hour DATAOS–Codex configuration in Table 8.

Both conditions receive the same destination optimization budget, while inheritance additionally requires the source run. The comparison therefore matches destination budgets, rather than total computational costs. We report destination scores and their differences from the corresponding no-inheritance references in Table 3.

B.4.4 TRAJECTORY DIAGNOSTICS

We analyze the joint-optimization trajectories to examine how successive training interventions affect model performance and how earlier experimental artifacts are reused. The analysis uses the checkpoint scores and intervention records reported in Section C.3.

Performance trajectories. For each evaluated checkpoint, we examine its scores on the seven PTB benchmarks and the joint objective $J(\theta)$ defined in Section B.4.1. We distinguish the score of each candidate checkpoint from the best score observed up to that point. This distinction shows whether an intervention improves the best observed performance, recovers from a regression, or produces a lower-scoring candidate.

Artifact reuse. We trace the starting checkpoint, training data, and training procedure associated with each intervention. These records identify branches that continue from recent checkpoints, revisit earlier checkpoints, or reuse data and methods developed in previous attempts. The analysis follows the operations actually executed, including deviations from the proposed intervention.

Interpretation. The trajectories describe the observed sequence of interventions and outcomes. When an intervention changes both the data and the training configuration, its outcome does not isolate the contribution of either factor. We therefore use these trajectories to illustrate optimization and artifact reuse, while assessing component contributions through the controlled ablations in Section B.4.2.

C COMPLETE RESULTS AND TRAJECTORIES

This appendix reports target-model results and per-benchmark scaling scores. Empty cells denote unavailable results.

C.1 PTB TARGET-MODEL DETAILS

Table 8 combines target-model references and agent results in one matrix. Each target–benchmark endpoint is optimized in a separate run. DATAOS uses medium reasoning effort, and each Δ uses native-medium in the same researcher family as its reference.

C.2 CURATION-BENCH DETAILS

Table 9 presents original-model and Random references alongside native agents and DATAOS in three 10K target-model panels. Each agent session contains 10 curation–training–evaluation iterations; the reported eight-task vector belongs to the checkpoint with the best normalized aggregate within that session. DATAOS uses medium reasoning effort.

Table 8: **Full PTB target-model results.** Target-model references, additional baselines, native agents, and DataOS are reported together for each target model. Scores are on the 0–100 scale.

System / reference	Benchmark scores ↑							Overall ↑
	AIME	Arena	BFCL	GPQA	GSM8K	Health	Human Eval	Avg.
(a) Qwen3-1.7B								Target model
Base model	2.50	1.40	13.00	16.96	39.88	6.01	10.37	12.87
DataMaster	2.92	4.80	82.00	24.78	71.34	7.90	42.07	33.69
AutoTrainess	4.17	7.20	70.00	27.90	74.52	10.20	35.37	32.77
Codex (medium)	3.33	5.00	76.00	25.22	48.90	6.60	33.54	28.37
Codex (max)	2.92	5.95	83.00	31.25	78.85	12.21	41.46	36.52
DataOS–Codex	4.17	11.50	93.00	29.69	77.86	14.99	39.02	38.60
Δ vs. Codex med.	+0.84	+6.50	+17.00	+4.47	+28.96	+8.39	+5.48	+10.23
Claude Code (medium)	3.33	2.01	84.00	25.45	78.77	13.40	54.88	37.41
Claude Code (max)	5.00	4.49	91.00	31.47	74.91	9.18	56.10	38.88
DataOS–Claude Code	7.08	6.50	89.00	31.70	79.00	12.70	62.20	41.17
Δ vs. Claude med.	+3.75	+4.49	+5.00	+6.25	+0.23	-0.70	+7.32	+3.76
(b) Qwen3-4B								Target model
Base model	7.92	7.06	75.00	12.50	4.55	9.56	36.59	21.88
DataMaster	13.75	12.40	77.00	29.24	86.35	14.10	67.68	42.93
AutoTrainess	18.33	9.64	84.00	30.58	87.64	15.60	58.54	43.48
Codex (medium)	20.00	10.50	75.00	16.96	43.29	11.80	31.71	29.89
Codex (max)	16.67	0.51	89.00	29.46	89.54	13.69	67.68	43.79
DataOS–Codex	11.67	18.40	91.00	34.15	90.37	22.63	62.80	47.29
Δ vs. Codex med.	-8.33	+7.90	+16.00	+17.19	+47.08	+10.83	+31.09	+17.39
Claude Code (medium)	21.67	64.40	90.00	18.75	60.20	19.69	65.24	48.56
Claude Code (max)	19.17	70.32	96.00	14.06	55.57	25.31	71.34	50.25
DataOS–Claude Code	28.75	65.55	97.00	20.09	67.02	26.10	67.68	53.17
Δ vs. Claude med.	+7.08	+1.15	+7.00	+1.34	+6.82	+6.41	+2.44	+4.61
(c) SmolLM3-3B								Target model
Base model	0.83	0.60	2.00	11.38	50.72	5.09	12.20	11.83
DataMaster	5.42	15.60	73.00	23.88	79.91	11.02	46.34	36.45
AutoTrainess	3.75	21.20	61.00	25.89	82.33	12.80	51.83	36.97
Codex (medium)	4.58	8.60	75.00	25.45	82.87	11.93	29.27	33.96
Codex (max)	2.08	11.95	7.00	10.27	80.74	12.53	32.93	22.50
DataOS–Codex	17.50	14.76	85.00	33.04	80.44	7.39	50.00	41.16
Δ vs. Codex med.	+12.92	+6.16	+10.00	+7.59	-2.43	-4.54	+20.73	+7.20
Claude Code (medium)	15.42	18.95	82.00	26.12	83.62	17.33	43.29	40.96
Claude Code (max)	20.00	24.88	81.00	18.08	91.28	19.31	37.80	41.76
DataOS–Claude Code	22.08	24.22	85.00	28.57	87.95	19.73	47.56	45.02
Δ vs. Claude med.	+6.66	+5.27	+3.00	+2.45	+4.33	+2.40	+4.27	+4.05
(d) Gemma3-4B								Target model
Base model	0.00	0.50	20.00	0.89	2.73	4.53	0.00	4.09
DataMaster	0.42	8.72	66.00	27.23	63.76	12.44	28.66	29.60
AutoTrainess	0.83	14.40	62.00	29.02	65.58	13.60	33.54	31.28
Codex (medium)	0.00	13.30	74.00	26.12	55.19	13.24	18.90	28.68
Codex (max)	0.00	13.20	86.00	28.13	62.47	17.29	33.54	34.38
DataOS–Codex	0.83	13.10	86.00	30.58	65.58	15.84	28.66	34.37
Δ vs. Codex med.	+0.83	-0.20	+12.00	+4.46	+10.39	+2.60	+9.76	+5.69
Claude Code (medium)	0.42	0.50	20.00	0.89	66.57	16.99	34.15	19.93
Claude Code (max)	0.00	0.00	16.00	0.00	66.41	10.63	41.46	19.21
DataOS–Claude Code	1.67	28.90	86.00	28.57	74.07	23.80	34.76	39.68
Δ vs. Claude med.	+1.25	+28.40	+66.00	+27.68	+7.50	+6.81	+0.61	+19.75

Table 9: **Full Curation-Bench target-model results.** References, native agents, and DataOS under the matched 10K training-sample setting. Each panel fixes one target model.

System / reference	Benchmark scores ↑								Overall ↑
	Hallusion	LLaVA	OCR	MMMU	Math Vista	MMStar	MM Bench	MMVet	Agg.
(a) SmolVLM-Base									Target model
Original model	4.05	10.50	20.70	26.22	21.30	31.07	3.35	5.18	0.152963
Random	23.01	16.90	42.80	31.44	24.00	31.80	15.55	27.20	0.265875
DataMaster	22.73	16.42	41.70	32.38	23.64	30.87	15.91	26.47	0.262650
AutoTrainess	23.13	16.90	44.10	33.21	24.18	32.14	16.73	27.62	0.272513
Codex (medium)	23.71	17.80	42.90	33.67	22.50	30.20	13.40	26.42	0.263250
Codex (max)	23.97	12.60	37.50	31.33	24.00	32.20	19.85	20.83	0.252850
DataOS–Codex	24.36	18.12	45.70	33.67	24.70	31.73	16.92	28.35	0.279438
Δ vs. Codex med.	+0.65	+0.32	+2.80	+0.00	+2.20	+1.53	+3.52	+1.93	+0.016188
Claude Code (medium)	25.20	22.00	37.20	32.22	25.40	31.87	25.77	19.50	0.273950
Claude Code (max)	21.30	20.00	36.70	33.89	24.90	32.67	27.84	20.69	0.272488
DataOS–Claude Code	27.18	25.70	41.00	34.00	27.10	35.47	29.55	22.98	0.303725
Δ vs. Claude med.	+1.98	+3.70	+3.80	+1.78	+1.70	+3.60	+3.78	+3.48	+0.029775
(b) Qwen2.5-VL-3B									Target model
Original model	46.20	64.30	82.50	51.11	63.20	57.67	80.07	65.60	0.638313
Random	46.93	63.70	83.00	50.44	61.60	57.40	78.95	64.68	0.633375
DataMaster	46.37	66.52	82.30	48.36	61.92	56.41	79.63	62.34	0.629813
AutoTrainess	47.12	68.34	84.70	49.41	62.47	57.83	80.26	63.21	0.641675
Codex (medium)	45.94	67.40	83.20	47.44	61.40	56.67	80.76	61.97	0.630975
Codex (max)	45.70	69.90	83.50	47.00	62.00	57.73	80.76	61.79	0.635475
DataOS–Codex	47.55	71.30	82.10	49.22	62.30	57.87	80.33	64.82	0.644363
Δ vs. Codex med.	+1.61	+3.90	-1.10	+1.78	+0.90	+1.20	-0.43	+2.85	+0.013388
Claude Code (medium)	46.00	66.90	83.10	47.22	62.00	56.67	80.33	62.20	0.630525
Claude Code (max)	43.13	68.20	80.40	49.22	63.60	54.40	82.13	64.40	0.631850
DataOS–Claude Code	48.49	67.90	87.00	50.22	63.50	60.13	83.25	63.99	0.655600
Δ vs. Claude med.	+2.49	+1.00	+3.90	+3.00	+1.50	+3.46	+2.92	+1.79	+0.025075
(c) LLaVA-1.5-7B									Target model
Original model	24.75	35.50	30.40	37.67	24.10	32.13	61.94	31.88	0.347963
Random	28.06	37.60	31.00	36.22	25.60	32.67	61.94	32.89	0.357475
DataMaster	27.14	38.27	29.70	35.74	24.83	34.46	57.82	29.37	0.346663
AutoTrainess	27.38	37.30	31.20	37.56	25.30	32.00	60.17	31.26	0.352713
Codex (medium)	26.55	41.10	28.60	32.22	25.50	34.73	55.58	26.56	0.338550
Codex (max)	24.38	37.10	29.00	33.67	23.70	35.53	53.87	28.49	0.332175
DataOS–Codex	29.47	40.73	31.60	36.52	26.34	35.28	62.29	34.45	0.370850
Δ vs. Codex med.	+2.92	-0.37	+3.00	+4.30	+0.84	+0.55	+6.71	+7.89	+0.032300
Claude Code (medium)	27.90	37.90	27.30	34.56	25.20	33.00	55.84	25.78	0.334350
Claude Code (max)	26.11	37.70	25.50	33.78	23.60	31.87	54.64	23.44	0.320800
DataOS–Claude Code	30.38	39.30	31.20	35.67	27.10	35.73	58.16	29.82	0.359200
Δ vs. Claude med.	+2.48	+1.40	+3.90	+1.11	+1.90	+2.73	+2.32	+4.04	+0.024850

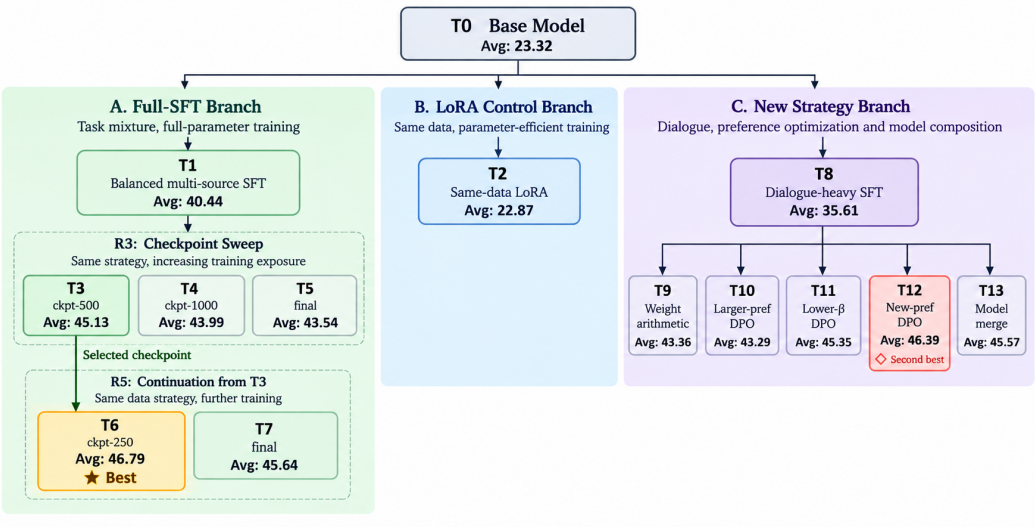

Figure 2: End-to-end view of the DATAOS–Codex scaling trajectory. Starting from the base model, the researcher explores full-parameter SFT, a same-data LoRA control, separate continuation branches from historical checkpoints, and later branches using weight arithmetic, preference optimization, and model merging. The diagram summarizes the research process represented by Tables 10 and 11: each branch proposes a data or training intervention, the resulting checkpoint is evaluated on all seven benchmarks, and the aggregate score determines which checkpoint is retained for subsequent improvement. T6 is the best observed checkpoint (Avg. 46.79), while T12 is the second-best candidate (Avg. 46.39).

C.3 SCALING TRAJECTORY AND INTERVENTION HISTORY

Checkpoint scores and DataOS interventions. Table 10 combines 19 supplied Codex checkpoints and 14 DATAOS–Codex checkpoints, including each initial-model evaluation. The two initial score vectors are retained as supplied rather than forced to share a baseline. The highest reported seven-task means are 43.86 for Codex (T11) and 46.79 for DATAOS–Codex (T6); these are checkpoint-level summaries, not yet time-aligned endpoint comparisons. Table 11 summarizes the starting checkpoints, reused resources, and executed interventions for the DATAOS–Codex panel. Each continuation stage denotes a separate training run; stage indices follow the recorded sequence rather than elapsed time. The candidate-score sequence includes regressions, whereas its running best is 23.32 at T0, 40.44 at T1–T2, 45.13 at T3–T5, and 46.79 at T6–T13.

The same 28K dataset is used for T1 and T2, but both adaptation method and learning rate change; these rows compare the recorded configurations. For T6, the planned filtering operation did not take effect, and the observed gain therefore accompanies continued training on the existing recipe. T8 likewise records the realized data mixture, excluding the unused ToolACE component. These distinctions preserve the difference between planned and executed interventions.

Table 10: Checkpoint scores during joint optimization of Qwen3-4B-Base by native Codex and DATAOS–Codex. Both use medium reasoning effort and are configured with four H100 GPUs for 20 hours. Each row reports one checkpoint's seven PTB scores; Avg. is the reported unweighted mean. T0 denotes the initial-model evaluation, and step indices are local to each system. Bold marks each system's best observed mean; these are not time-aligned endpoint comparisons. See Appendix B.4.1.

Step	GSM8K	BFCL	HumanEval	AIME25	GPQA	Health	Arena	Avg.
(a) Codex								
T0	14.56	72.00	45.73	7.08	13.62	10.55	5.60	24.16
T1	83.02	65.00	39.02	2.92	21.21	16.23	5.60	33.28
T2	52.31	53.00	29.27	3.33	18.75	18.55	1.55	25.25
T3	71.57	0.00	31.71	1.25	22.32	18.71	1.36	20.99
T4	86.58	73.00	57.32	4.58	27.23	22.61	15.45	40.97
T5	86.35	77.00	54.27	4.17	27.68	25.24	22.24	42.42
T6	84.15	78.00	38.41	2.50	20.76	18.18	7.80	35.69
T7	86.50	74.00	51.83	4.58	30.36	23.87	19.40	41.51
T8	87.41	81.00	60.37	4.17	28.57	24.27	17.17	43.28
T9	84.53	64.00	39.63	10.42	22.77	21.05	9.20	35.94
T10	86.81	80.00	60.37	2.50	31.03	24.65	17.70	43.29
T11	**88.17**	**79.00**	**60.37**	**3.75**	**31.25**	**25.39**	**19.13**	**43.86**
T12	88.10	79.00	57.32	2.08	24.11	25.02	15.05	41.53
T13	87.57	92.00	62.20	1.25	31.25	17.38	10.40	43.15
T14	88.48	91.00	62.80	0.83	28.79	17.04	17.17	43.73
T15	87.49	92.00	61.59	2.50	31.70	16.64	9.95	43.12
T16	87.19	90.00	64.63	1.67	32.59	17.97	10.24	43.47
T17	86.88	90.00	60.98	0.83	28.79	18.71	9.30	42.21
T18	87.87	90.00	60.98	0.00	29.02	17.71	8.80	42.05
(b) DATAOS–Codex								
T0	14.33	71.00	41.46	7.08	12.95	11.72	4.70	23.32
T1	88.25	90.00	58.54	0.42	28.79	11.00	6.10	40.44
T2	39.35	89.00	0.61	0.83	23.44	6.48	0.35	22.87
T3	89.61	91.00	67.07	2.08	32.59	19.97	13.60	45.13
T4	89.31	90.00	66.46	0.42	29.24	20.21	12.30	43.99
T5	88.48	90.00	66.46	0.83	28.57	18.31	12.10	43.54
T6	**89.08**	**92.00**	**71.34**	**2.08**	**29.46**	**22.32**	**21.23**	**46.79**
T7	90.14	91.00	67.07	0.00	32.59	21.79	16.90	45.64
T8	82.11	71.00	43.29	1.67	26.56	18.56	6.10	35.61
T9	87.41	80.00	60.98	4.58	29.02	25.36	16.20	43.36
T10	86.05	79.00	59.76	4.58	29.24	26.11	18.30	43.29
T11	85.82	78.00	62.80	5.42	28.12	27.90	29.40	45.35
T12	86.35	82.00	61.59	3.75	28.57	31.55	30.90	46.39
T13	86.58	81.00	60.98	3.33	31.47	27.52	28.10	45.57

Table 11: Interventions and resource reuse across the DATAOS–Codex joint-optimization trajectory. Stage IDs and mean scores match Table 10(b). The highlighted row marks the best observed checkpoint.

Stage	Starting point / reused resources	Executed change	Avg.
T0	Base model.	Initial evaluation; no training.	23.32
T1	Base model; 28K multi-source examples covering mathematics, code, tools, and medicine.	Message normalization, deduplication, and length filtering; full-parameter SFT, LR $= 10^{-5}$, 1 epoch, 875 steps.	40.44
T2	Base model; the same 28K dataset as T1.	Attention LoRA, LR $= 2 \times 10^{-4}$, rank 64, $\alpha = 128$, dropout 0.05, 1 epoch.	22.87
T3	T1 checkpoint; 32K multilingual dialogue and task-replay examples.	Full-parameter SFT, LR $= 5 \times 10^{-6}$.	45.13
T4	T3 checkpoint and its 32K recipe.	Full-parameter SFT continuation.	43.99
T5	T4 checkpoint; the T3 data recipe.	Full-parameter SFT continuation.	43.54
T6[a]	T3 checkpoint and its 32K recipe.	New full-parameter SFT continuation, LR $= 5 \times 10^{-6}$.	**46.79**
T7	T6 checkpoint and its data recipe.	Full-parameter SFT continuation.	45.64
T8[b]	32K WildChat plus 45,983 replay examples; initialization unspecified.	Full-parameter SFT, LR $= 3 \times 10^{-6}$, 1 epoch, 566 steps; batch 16, sequence length 8,192, packing.	35.61
T9	Checkpoints `C12`, `C11`, and `C04`.	Weight arithmetic: $\texttt{C12} + 0.2(\texttt{C11} - \texttt{C04})$.	43.36
T10	Preference subsets $P \cup N$ (19,955 pairs).	Full-parameter DPO, LR $= 10^{-6}$, 2 epochs, 1,248 steps, $\beta = 0.1$.	43.29
T11	T10's DPO pipeline; subset P.	Full-parameter DPO, LR $= 10^{-6}$, 2 epochs, 814 steps, $\beta = 0.025$.	45.35
T12	T10's DPO pipeline; subset N (6,955 pairs).	Full-parameter DPO, LR $= 5 \times 10^{-7}$, 1 epoch, 218 steps, $\beta = 0.05$.	46.39
T13	Checkpoints `C14` and `C18`.	Weight merge: $0.5\texttt{C14} + 0.5\texttt{C18}$.	45.57

Notes. T3–T7 are separate training runs. [a]The planned T6 filter did not take effect. [b]The planned 6K ToolACE component was unused at T8. P and N denote the recorded preference-data subsets. Initial checkpoints and reference models for the DPO runs are unspecified, as are additional training settings for T4–T5. Internal checkpoint IDs (`Cxx`) are retained where their mappings to stage IDs are unspecified.