DATAFOUNDRY: EVOLVING DATA PREPARATORS VIA RECURSIVE SELF-IMPROVEMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Domain adaptation of large language models increasingly depends on constructing high-quality training data, yet existing data-preparation pipelines typically address quality only after generation through post-hoc filtering. This creates a fundamental mismatch: data-quality issues often originate from the construction process itself, while quality control is applied only to its outputs. We introduce DATAFOUNDRY, a framework for **evolving data preparators through recursive self-improvement** before large-scale data production. DATAFOUNDRY represents a data preparator as an evolvable runtime specification and instantiates its evolution with a SKILLS-AS-MODULES architecture, in which a central CONTROLLER orchestrates modular skills to compile executable runtimes, diagnose deficiencies on small pilot sets using domain-appropriate criteria, and translate diagnostic feedback into adapters that revise individual preparation components while preserving stable interfaces. We evaluate DATAFOUNDRY on DataPrep-Bench across mathematics, finance, law, and medicine, and find that recursively evolved preparators produce training data with higher downstream utility than baselines. Experiments across different backbones further demonstrate that these improvements are not tied to a particular model, while analyses and case studies further reveal the framework's optimization dynamics and illustrate how its evolution unfolds in practice.

1 INTRODUCTION

Large language models adapted to specialized domains rely on the effective preparation of domain knowledge as training data (Ling et al., 2025; Liang et al., 2026). Instead of directly training on raw corpora, the adaptation typically involves restructuring them into examples that better align with the target tasks and desired model behaviors (Lu et al., 2025; Nayak et al., 2024). Synthetic instruction generation has made this process easier to scale. Early pipelines bootstrap instructions and responses from model generations (Wang et al., 2023; Ding et al., 2023; Xu et al., 2025a), then remove invalid, redundant, or low-quality examples (Li et al., 2025; Yang et al., 2026a).

However, existing pipelines exhibit a fundamental mismatch between where quality issues emerge and where quality control is applied: *quality issues can arise during the construction process, whereas quality control is typically applied only to the resulting samples*. As shown in Figure 1, these pipelines often rely on post-hoc filtering to discard large fractions of generated data (Xu et al., 2025b; Yu et al., 2025; Li et al., 2024), rather than correcting the upstream processes that repeatedly produce poor examples. This mismatch suggests that improving data quality requires **optimizing the preparation pipeline itself before scaling it up**.

Addressing this mismatch requires shifting quality intervention upstream, but doing so is far from straightforward. First, data construction often involves long, multi-stage workflows implemented as standalone code (Fan et al., 2026; Shi et al., 2026; Kulikov et al., 2026; Du et al., 2026), making it difficult to **diagnose and incorporate iterative improvements**. Second, even when problematic stages can be identified, there is **limited actionable guidance on what should be optimized and how**, particularly across domains with different requirements, such as mathematics, finance, law, and medicine (Ling et al., 2025). These challenges motivate a systematic approach that can trace quality issues back to their construction processes and provide targeted, domain-aware guidance for improving the pipeline before the final large-scale data preparation.

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.

To this end, we introduce DATAFOUNDRY, a framework for **evolving data preparators through recursive self-improvement** before large-scale data production. It represents it as an evolvable runtime specification. We instantiate this process with a SKILLS-AS-MODULES architecture, in which a central CONTROLLER orchestrates modular skills for compiling the current specification into an executable runtime, diagnosing deficiencies on small pilot sets using domain-appropriate criteria, and translating the resulting feedback into adapters that revise the preparator while preserving stable component interfaces. Through this recursive process, DATAFOUNDRY progressively specializes a general data preparator to the requirements of each target domain, and finally compiles the evolved specification into a domain-adapted runtime for full-scale supervised fine-tuning data production.

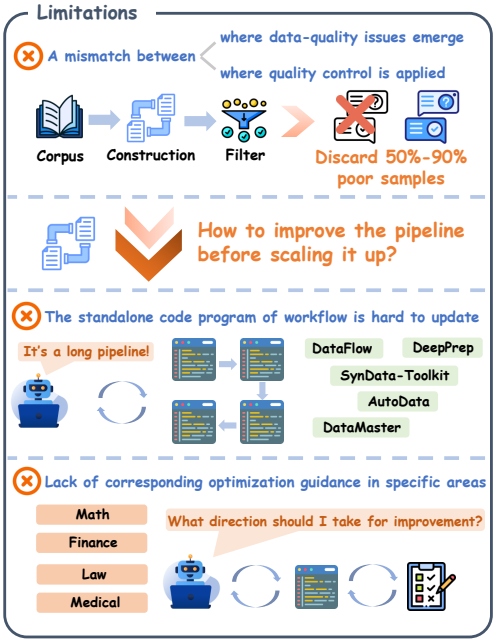

Figure 1: A mismatch between where data-quality issues emerge and where quality control is applied during data preparation. Challenges exist when optimizing the pipeline before scaling it up.

We evaluate DATAFOUNDRY across four specialized domains, including *Math*, *Finance*, *Law*, and *Medical*, and show that its recursively evolved data preparators consistently improve the downstream utility of the prepared training data. Beyond the main results with Qwen2.5-7B, we conduct ablation studies with different downstream base models to examine whether the gains generalize beyond a particular model. We further vary the self-improvement rounds to study how performance evolves and to characterize the growth potential of the proposed framework. Finally, through case studies, we trace the changes introduced during recursive self-improvement to individual pipeline components, providing a component-level explanation of how DATAFOUNDRY progressively adapts the data-production process to domain-specific requirements.

2 RELATED WORK

2.1 LLM DATA CONSTRUCTION

Early data construction pipelines bootstrap instructions from model generations using filters to control noise (Wang et al., 2023; Ding et al., 2023). Subsequent work increases task complexity through iterative instruction rewriting (Xu et al., 2025a), or couples large-scale synthesis to improve instruction-following quality (Li et al., 2025). Recent systems elevate data preparation from a fixed generator to a modular workflow (Shi et al., 2026; Fan et al., 2026; Liang et al., 2025). DataMaster searches over cumulative data states with downstream training feedback, and Autodata iteratively creates, inspects, evaluates, and revises synthetic-data recipes (Du et al., 2026; Kulikov et al., 2026). Some works attempt to optimize the prompt iteratively for data generation (Zeng et al., 2024), in contrast, DATAFOUNDRY recursively adapts the executable data construction runtime itself using contextualized assessment, then performs full-scale construction on evolved runtime.

2.2 AGENT SKILLS AND EVOLUTION

Agent skills externalize procedural knowledge as composable packages of instructions, code, and resources, providing a portable adaptation surface (Xu & Yan, 2026). One line of work turns experience into reusable skills by distilling trajectories or extracting from recurring interactrions (Xia et al., 2026; Yang et al., 2026c; Ma et al., 2026). A complementary line directly optimizes skill artifacts. CoEvoSkills couples multi-file skill generation with a co-evolving surrogate verifier (Zhang et al., 2026); Skill-R1 trains a lightweight skill generator with intra- and inter-generation rewards while

freezing the task model (Vishe et al., 2026); and SkillOpt casts skill revision as bounded text-space optimization with held-out tests (Yang et al., 2026b). Unlike existing methods, DATAFOUNDRY evolves the skill with domain-specific changes isolated as inspectable specification adapters.

2.3 RECURSIVE SELF-IMPROVEMENT

Recursive self-improvement (RSI) spans qualitatively different update surfaces, from bounded output refinement and self-training to evaluator improvement, harness modification, and autonomous research loops (Lee et al., 2026b; Chen et al., 2026; Lee et al., 2026a). Experience-centric accounts accordingly view the runtime harness as the infrastructure that captures trajectories and feedback and routes them into mutable skills, memory, environments, or parameters (Jiang et al., 2026). Harness engineering makes this external update surface concrete through explicit workflows, persistent state, context management, evaluation, and permission boundaries (Weng, 2026). DATAFOUNDRY instantiates a bounded RSI loop specifically for data preparation: multi-criterion assessment grounds each adapter update, while the baseline runtime specification constrains potential drift.

3 DATAFOUNDRY

3.1 BACKGROUND: DATA PREPARATION

Problem Formulation. Let d denote a target domain and let $\mathcal{K}_d$ denote its knowledge corpus, where each $k \in \mathcal{K}_d$ is a domain-relevant knowledge source, such as a textbook, novel, or other document. A data-construction runtime $\mathcal{R}$ transforms the domain corpus into a usable domain-specific supervised fine-tuning dataset:

$$\mathcal{D}_d^{\mathrm{SFT}} = \mathcal{R}(\mathcal{K}_d) = \{z_i\}_{i=1}^{N_d}, \quad z_i = (\iota_i, \rho_i), \tag{1}$$

where N_d denotes the realized data volume yielded, and ι_i and ρ_i denote the instruction and response of a training sample. Given a base model f_{θ_0}, the prescribed fine-tuning procedure $\mathcal{A}$ uses the prepared fine-tuning dataset $\mathcal{D}_d^{\mathrm{SFT}}$ to train a downstream domain model f_{θ_d}:

$$f_{\theta_d} = \mathcal{A}(f_{\theta_0}, \mathcal{D}_d^{\mathrm{SFT}}). \tag{2}$$

Construction-then-Filtering. A common instantiation follows a construction-then-filtering paradigm, in which candidate training data are first constructed and subsequently filtered. The data-construction runtime $\mathcal{R}$ transforms the corpus $\mathcal{K}_d$ into a candidate training data pool:

$$\widetilde{\mathcal{D}}_d^{\mathrm{SFT}} = \mathcal{R}(\mathcal{K}_d) = \{z_i\}_{i=1}^{\widetilde{N}_d}, \tag{3}$$

where $\widetilde{N}_d$ denotes the volume of the candidate pool. A data filter $\mathcal{F}$ evaluates the quality of candidate samples and selects a subset with high value for final downstream fine-tuning, written as

$$\mathcal{D}_d^{\mathrm{SFT}} = \mathcal{F}(\widetilde{\mathcal{D}}_d^{\mathrm{SFT}}) = \{z_i\}_{i=1}^{N_d}, \quad N_d \le \widetilde{N}_d. \tag{4}$$

3.2 INSPIRATION: EVOLVING VIA RECURSIVE SELF-IMPROVEMENT

To address the challenges mentioned above, we shift adaptation upstream from filtering the outputs to improving the preparator itself, using *recursive self-improvement* (Chen et al., 2026) before final production. We begin with an initial runtime specification ϕ_{base} that describes the general workflow of data production. A runtime compiler translates ϕ_{base} into an online construction runtime:

$$\widetilde{\mathcal{R}} = \mathrm{Compile}(\phi_{\text{base}}). \tag{5}$$

For a target domain d, a contextualized quality assessor selects domain-specific criteria $\mathcal{C}_d$ to examine data quality from multiple perspectives. At iteration t, the current specification $\phi_d^{(t)}$ is compiled into a runtime $\widetilde{\mathcal{R}}_d^{(t)}$, which then constructs a pilot set with a small generation budget $\widehat{N}$:

$$\mathcal{P}_d^{(t)} = \widetilde{\mathcal{R}}_d^{(t)}(\mathcal{K}_d; \widehat{N}). \tag{6}$$

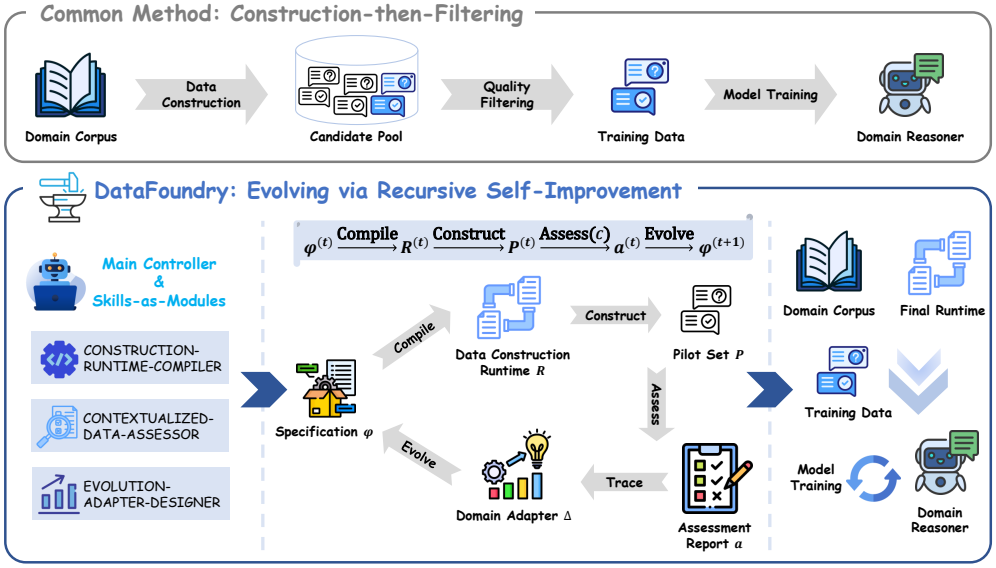

Figure 2: The overall framework of DATAFOUNDRY. Unlike the common method that constructs first then filters, our DATAFOUNDRY evolves the data preparator via recursive self-improvement, prior to final data production.

The assessor evaluates this pilot set and returns the report:

$$a_d^{(t)} = \text{Assess}(\mathcal{P}_d^{(t)}; \mathcal{C}_d). \tag{7}$$

An evolution operator then updates the specification using this feedback:

$$\phi_d^{(t+1)} = \text{Evolve}(\phi_d^{(t)}, a_d^{(t)}), \quad \phi_d^{(0)} = \phi_{\text{base}}. \tag{8}$$

In summary, a *recursive self-improvement* process is formalized in the following loop:

$$\phi_d^{(t)} \xrightarrow{\text{Compile}} \widetilde{\mathcal{R}}_d^{(t)} \xrightarrow{\mathcal{K}_d, \widehat{N}} \mathcal{P}_d^{(t)} \xrightarrow{\text{Assess}(\cdot;\mathcal{C}_d)} a_d^{(t)} \xrightarrow{\text{Evolve}} \phi_d^{(t+1)}. \tag{9}$$

After T rounds, the evolved specification is compiled into the final domain-adapted runtime for full-scale production of the fine-tuning dataset:

$$\mathcal{R}_d^{(T)} = \text{Compile}(\phi_d^{(T)}), \quad \mathcal{D}_d^{\text{SFT}} = \mathcal{R}_d^{(T)}(\mathcal{K}_d) = \{z_i\}_{i=1}^{N_d}. \tag{10}$$

3.3 FRAMEWORK: CONTROLLER AND SKILLS-AS-MODULES

DATAFOUNDRY implements the RSI process above through a SKILLS-AS-MODULES architecture, in which a main CONTROLLER driven by a coding agent $\mathcal{G}$ orchestrates three modular skills to compile, assess, and evolve the data-construction runtime specification ϕ:

- **CONSTRUCTION-RUNTIME-COMPILER** realizes Compile, in which $\mathcal{G}$ translates the specification ϕ into an executable data-construction runtime $\mathcal{R}$.
- **CONTEXTUALIZED-DATA-ASSESSOR** realizes Assess, in which $\mathcal{G}$ selects a set of domain-appropriate criteria $\mathcal{C}_d$ and applies them to the pilot set $\mathcal{P}_d$.
- **EVOLUTION-ADAPTER-DESIGNER** realizes Evolve, in which $\mathcal{G}$ attributes diagnosed deficiencies to components and expresses the required changes as an adapter for $\phi_d^{(t)}$.

The following sections describe the concrete realization of compilation, assessment, and evolution.

3.3.1 DATA-CONSTRUCTION RUNTIME COMPILATION

This skill defines the initial runtime specification ϕ including four stable component interfaces:

- **`corpus_chunk_manager`** samples source chunks from $\mathcal{K}_d$ to provide the knowledge context for the construction of sample $z = (\iota, \rho)$;
- **`instruction_builder`** converts the sampled context into a domain-relevant instruction ι in the required format;
- **`response_generator`** produces a formatted response ρ conditioned on the instruction and its source context; and
- **`batch_generation_service`** concurrently invokes external models to produce samples.

The compiler additionally provides engineering constraints like smoke tests and structured logging, which make each compiled runtime stable and observable.

3.3.2 CONTEXTUALIZED PILOT-SET ASSESSMENT

This skill defines the data quality assessment equipped with a library $\mathcal{L}_{\text{crit}}$ of candidate assessment criteria. The candidate criteria cover several complementary quality signals:

- **`instruction_alignment`**: whether the response directly, accurately, and sufficiently addresses the given instruction and its intended task;
- **`self_containment`**: whether the sample is complete and understandable without relying on missing context or unspecified external information;
- **`information_density`**: whether the response contains sufficient domain-relevant and useful information while avoiding excessive redundancy or unnecessary verbosity;
- **`factual_grounding`**: whether the response is well supported by the provided source context and avoids unsupported, inconsistent, or fabricated claims;
- **`domain_utility`**: whether the sample captures informative domain knowledge, concepts, and reasoning patterns that are useful for downstream domain adaptation; and
- **`numerical_correctness`**: whether numerical values, calculations, quantitative relationships, and arithmetic reasoning in the response are accurate and internally consistent.

Given a domain d, the domain-specific criteria $\mathcal{C}_d \subseteq \mathcal{L}_{\text{crit}}$ are selected contextually and applied to assess the pilot set $\mathcal{P}_d^{(t)}$, and the findings given by $\mathcal{G}$ are aggregated into the assessment report $a_d^{(t)}$.

3.3.3 ADAPTER-BASED SPECIFICATION EVOLUTION

This skill takes the specification ϕ_{base}, the adapter $\Delta_d^{(t-1)}$ and the report $a_d^{(t)}$, then traces the diagnosed deficiencies to components. It then expresses the recommended modifications as:

$$\Delta_d^{(t)} = \text{Trace}(\phi_{\text{base}}, \Delta_d^{(t-1)}, a_d^{(t)}). \tag{11}$$

The adapter may revise component-level policies, prompts, configurations, or coordination logic while preserving the stable interfaces enforced by the compiler. In particular, it can modify the behavior of components such as `corpus_chunk_manager`, `instruction_builder`, and `response_generator`, thereby addressing deficiencies in the construction process.

The CONTROLLER applies the adapter to obtain the specification used in the next pilot iteration:

$$\phi_d^{(t+1)} = \text{Adapt}(\phi_{\text{base}}, \Delta_d^{(t)}). \tag{12}$$

By isolating domain-specific changes in $\Delta_d^{(t)}$ while preserving the baseline ϕ_{base}, it mitigates iterative updates from accumulating uncontrolled modifications that may lead to specification collapse.

Table 1: Performance of Qwen2.5-7B on Math and Finance benchmarks after fine-tuning on Dolly-15k jointly with the datasets synthesized by different generators.

Training Data Generator	Math								Finance			
	GSM8K	AMC23	AIME24	MM	OB	M5000	GK24	Avg	CKQA	FEKR	XFB	Avg
Dolly-15k only	69.9	17.5	0.0	10.7	10.7	39.8	16.5	23.6	57.6	59.4	56.3	57.8
DataFlow-based Generators												
DataFlow	56.5	7.5	0.0	7.7	7.9	27.4	17.6	17.8	51.0	54.5	59.3	54.9
DataFlow-Skill	56.7	10.0	0.0	8.8	6.2	22.8	**28.6**	19.0	60.0	65.4	**68.9**	64.8
LLM-based Generators												
Claude Opus 4.6	68.8	12.5	0.0	8.8	9.8	35.5	14.3	21.4	37.6	39.6	55.9	44.4
Gemini 3.0 Pro	72.1	20.0	0.0	10.7	11.4	37.9	15.4	23.9	48.6	49.5	58.4	52.2
GPT-5.2	66.7	17.5	0.0	7.7	11.1	32.7	14.3	21.4	47.1	43.6	53.6	48.1
Agent-based Generators												
Qwen3.5-Plus	72.7	22.5	3.3	11.0	11.6	38.7	16.5	25.2	48.1	49.5	55.6	51.1
GLM-4.7	71.4	22.5	3.3	9.6	11.6	37.9	20.9	25.3	51.4	55.4	53.1	53.3
Claude Opus 4.6	69.4	10.0	0.0	11.0	8.9	33.8	19.8	21.8	32.4	39.6	53.8	41.9
Gemini 3.0 Pro	70.1	15.0	0.0	8.8	10.8	35.7	20.9	23.0	53.3	52.5	55.4	53.7
GPT-5.2	69.6	25.0	3.3	8.8	9.9	35.6	26.4	25.5	39.1	39.6	51.0	43.2
GPT-5.3-codex	70.8	15.0	0.0	11.0	11.7	38.5	13.2	22.9	58.6	63.4	55.9	59.3
GPT-5.6 Sol	71.8	22.5	3.3	11.4	11.7	37.5	19.8	25.4	50.5	51.5	53.1	51.7
Skill-based Generators												
DataPrep-Skill (Opus 4.6)	72.6	17.5	3.3	**14.0**	11.1	36.8	13.2	24.1	57.6	53.5	55.4	55.5
DataPrep-Skill (GPT-5.6 Sol)	71.3	17.5	3.3	9.6	11.6	40.7	20.9	25.0	49.0	64.4	66.2	59.9
DataFoundry (GPT-5.6 Sol)	**75.1**	**27.5**	**6.7**	11.4	**13.0**	**44.7**	23.1	**28.8**	**61.4**	**71.3**	65.5	**66.1**

4 EXPERIMENTS

4.1 IMPLEMENTATION DETAILS

We conduct experiments on a Linux server equipped with (8×) NVIDIA H800 GPUs. We use GPT-5.6 Sol [1] as the backbone model for the coding agent and Codex CLI [2] as its execution interface. We follow the domain corpus used in (Liang et al., 2026) with 136 math documents, 80 finance documents, 53 law documents and 154 medical documents. The details of our skills-as-modules framework are provided in Appendix A, and the downstream SFT setup are provided in Appendix B.

4.2 EVALUATION SETTINGS

The evaluation benchmarks and existing experimental results are adopted from (Liang et al., 2026), covering four domains and various sub-tasks: (1) **Math**: GSM8K, AMC 2023, AIME 2024, MinervaMath, Math-5000, Gaokao 2024, OlympiadBench; (2) **Finance**: CPA-KQA, FinEval-KR, XFinBench; (3) **Law**: LegalBench, LexGLUE; (4) **Medical**: MedCaseReasoning, MedMCQA, MedRBench. The details of the four benchmark datasets are provided in Appendix C.

All baselines and our DATAFOUNDRY are fine-tuned on Dolly-15k [3] (a general instruction-following corpus) jointly with the domain-specific dataset produced by the named generator. We fix the number of RSI rounds at two, as we observe that the pilot set assessment results typically begin to deteriorate after two rounds. The implementation of baselines are provided in Appendix D.

4.3 MAIN RESULTS

DATAFOUNDRY achieves the best overall performance across all domains. As shown in Tables 1 and 2, DATAFOUNDRY attains average scores of 28.8, 66.1, 78.6, and 46.0 on Math, Finance, Law, and Medical, respectively, outperforming the strongest competing generator by 3.3, 1.3, 1.4, and 2.2 points. The results demonstrate the effectiveness and generality of recursive self-

[1] https://developers.openai.com/api/docs/models

[2] https://learn.chatgpt.com/docs/codex/cli

[3] https://huggingface.co/datasets/databricks/databricks-dolly-15k

Table 2: Performance of Qwen2.5-7B on Law and Medical benchmarks after fine-tuning on Dolly-15k jointly with the datasets synthesized by different generators.

Training Data Generator	Law			Medical			
	LegalBench	LexGLUE	Avg	MedCaseReasoning	MedMCQA	MedR-Bench	Avg
Dolly-15k only	86.9	62.0	74.5	13.6	27.4	67.8	36.3
DataFlow-based Generators							
DataFlow	89.7	64.8	77.2	9.9	29.4	63.6	34.3
DataFlow-Skill	**92.0**	57.4	74.7	11.9	24.1	65.6	33.9
LLM-based Generators							
Claude Opus 4.6	88.0	63.2	75.6	13.9	8.1	66.8	29.6
Gemini 3.0 Pro	85.9	63.5	74.7	12.2	6.0	69.1	29.1
GPT-5.2	89.2	60.9	75.0	10.6	23.3	66.0	33.3
Agent-based Generators							
Qwen3.5-Plus	90.2	61.0	75.6	16.5	16.5	68.3	33.8
GLM-4.7	84.8	61.4	73.1	15.2	10.6	66.8	30.9
Claude Opus 4.6	65.2	48.8	57.0	16.6	50.7	54.3	40.5
Gemini 3.0 Pro	57.5	55.5	56.5	15.8	**52.3**	63.4	43.8
GPT-5.2	61.7	51.9	56.8	14.7	45.0	59.0	39.6
GPT-5.3-codex	87.8	63.4	75.6	13.6	20.3	70.0	34.6
GPT-5.6 Sol	85.0	62.6	73.8	11.3	27.8	65.6	34.9
Skill-based Generators							
DataPrep-Skill (Opus 4.6)	85.5	61.7	73.6	9.9	15.3	65.4	30.2
DataPrep-Skill (GPT-5.6 Sol)	82.6	58.7	70.7	15.8	39.8	64.6	40.1
DataFoundry (GPT-5.6 Sol)	87.4	**69.8**	**78.6**	**20.2**	47.0	**70.8**	**46.0**



Figure 3: Performance on Math and Finance across different RSI rounds using GPT-5.6 Sol as the backbone. DATAFOUNDRY generally improves through recursive evolution and achieves its best overall performance after two rounds, while further evolution leads to performance degradation.

improvement on data-preparation process. While DATAFOUNDRY does not necessarily achieve the highest score on every individual sub-task, it delivers the strongest domain-level averages, indicating that its advantage comes from adapting the preparation runtime to domain-specific needs and producing utility training data.

DATAFOUNDRY derives its gains from the framework rather than the backbone model alone. To isolate the contribution of our framework from that of the backbone model, we compare DATAFOUNDRY with both the *Agent-based Generator* and the *Skill-based Generator* instantiated with the same GPT-5.6 Sol backbone. Compared with the GPT-5.6 Sol Agent-based Generator, DATAFOUNDRY improves the domain-level average by 3.4 points on Math, 14.4 points on Finance, 4.8 points on Law, and 11.1 points on Medical. Moreover, compared with the GPT-5.6 Sol Skill-based Generator (DataPrep-Skill), DATAFOUNDRY achieves further gains of 3.8, 6.2, 7.9, and 5.9 points across the four domains, respectively. These controlled comparisons demonstrate that the improvements cannot be attributed merely to a stronger backbone, agentic execution, or access to reusable skills, rather, they highlight the effectiveness of DATAFOUNDRY's framework-level design.

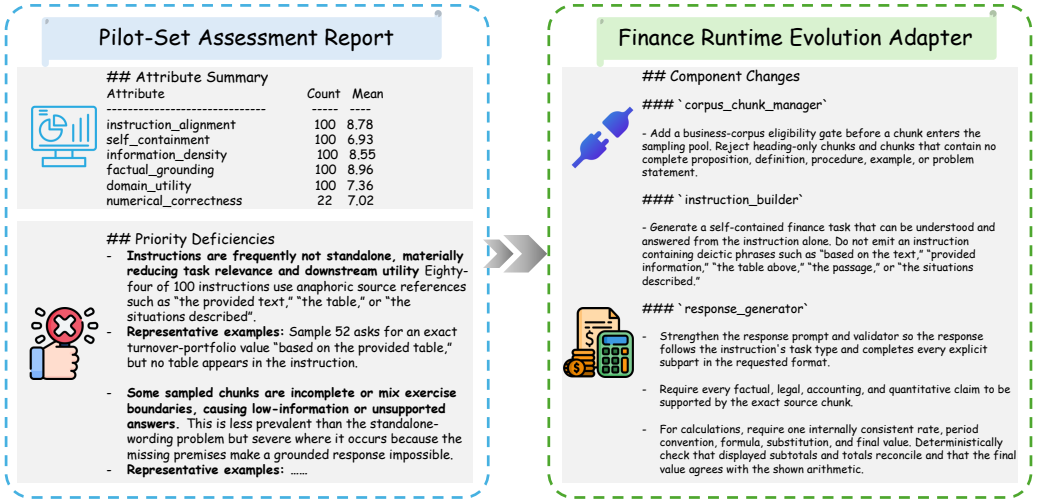

Figure 4: A case of implementation of RSI process in DATAFOUNDRY in Finance domain.

5 ANALYSIS

5.1 ABLATION STUDIES

DATAFOUNDRY benefits from recursive self-improvement, but excessive evolution can cause degradation. As shown in Figure 3, increasing the number of RSI rounds generally improves the downstream utility of the data synthesized by DATAFOUNDRY, with the strongest overall performance reached after two rounds of evolution. On Math, the average score increases from 25.6 at Round 0 to 28.4 and 28.8 at Rounds 1 and 2, respectively, before dropping to 26.1 at Round 3. Finance exhibits a less monotonic trajectory, but similarly reaches its best performance at Round 2 (66.1) and declines substantially to 58.3 after an additional round. These results suggest that recursive optimization is beneficial up to a certain point, while excessive self-improvement may introduce undesirable changes to the data-preparation runtime and lead to partial system collapse.

DATAFOUNDRY's prepared datasets generalize across different models . We replace the downstream base model with Llama-3.1-8B while keeping the training and evaluation settings unchanged. As shown in Table 3, DATAFOUNDRY remains highly competitive across all four domains, achieving the best performance on Math, Finance, and Medical with scores of 13.6, 41.3, and 39.9, respectively. In particular, it outperforms the best data generator by 2.6 points on Math, 4.8 points on Finance, and 3.8 points on Medical. Together with the results above, these findings indicate that the gains of DATAFOUNDRY are not tied to a specific model; instead, the prepared data provide transferable training utility across model families.

Table 3: Performance of Llama-3.1-8B.

Data Generator	Math	Finance	Law	Medical
Dolly-15k only	11.7	15.1	69.5	20.4
DataFlow-based Generators				
DataFlow	5.8	31.4	72.7	29.9
DataFlow-Skill	6.1	36.5	**73.1**	36.0
LLM-based Generators				
Claude Opus 4.6	9.3	28.4	71.0	36.1
Gemini 3.0 Pro	11.0	31.2	74.0	33.4
GPT-5.2	9.2	27.4	76.4	32.4
Agent-based Generators				
Claude Opus 4.6	6.7	28.9	69.8	32.7
Gemini 3.0 Pro	10.9	29.3	73.5	34.2
GPT-5.2	10.0	29.5	72.9	32.7
GPT-5.6 Sol	9.7	30.7	70.2	34.0
Skill-based Generators				
DataPrep-Skill (Opus 4.6)	7.4	34.2	70.2	35.4
DataPrep-Skill (5.6 Sol)	9.2	36.1	72.8	33.6
DataFoundry (5.6 Sol)	**13.6**	**41.3**	71.1	**39.9**

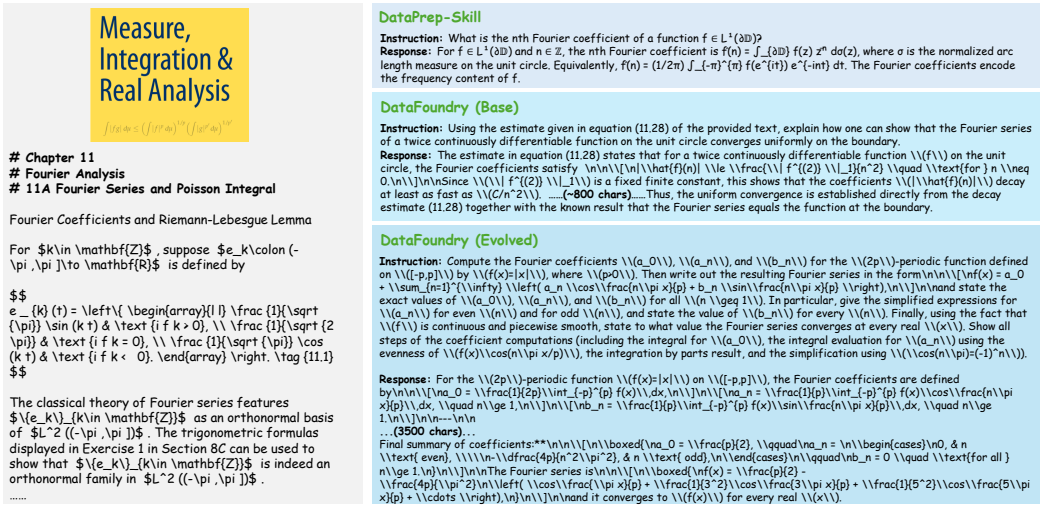

Figure 5: Comparison of instances generated by DataPrep-Skill and DATAFOUNDRY before and after self-improvement, illustrating progressively improved completeness and self-containment.

5.2 CASE STUDIES

DATAFOUNDRY translates diagnosed quality deficiencies into targeted component-level improvements. Figure 4 provides a concrete example of how DATAFOUNDRY translates pilot-set deficiencies into targeted component-level revisions. In the Finance domain, the assessment identifies two major sources of quality degradation: many instructions are not self-contained, with 84 of 100 samples relying on missing contextual references such as 'the provided text" or 'the table", making grounded responses difficult. DATAFOUNDRY traces the deficiencies back to the responsible runtime components and updates them accordingly. These coordinated revisions illustrate how recursive self-improvement operates at the component level: observed output failures are converted into localized changes to the corresponding stages of the preparation pipeline, progressively specializing the runtime toward domain-specific data-quality requirements.

DATAFOUNDRY produces higher-quality training instances, and recursive evolution further improves their quality. Figure 5 illustrates the progressive improvement in the supervision generated from the same domain corpus. The DataPrep-Skill baseline produces a relatively generic question with limited reasoning requirements, whereas DATAFOUNDRY generates substantially richer supervision. Nevertheless, the Base version still exhibits a quality issue identified during assessment: its instruction depends on external context through references such as "equation (11.28) of the provided text," making the example less self-contained. After evolution, DATAFOUNDRY transforms the instance into a fully specified Fourier-series problem. The corresponding response provides a complete and directly verifiable solution rather than relying on missing context. This example demonstrates that DATAFOUNDRY not only produces more reasoning-intensive training data, but also improves the data by enhancing the completeness and utility of individual examples.

6 CONCLUSION

We present DATAFOUNDRY, a framework that improves domain-specific data preparation by recursively evolving the data preparator itself before large-scale synthesis. By diagnosing quality issues on pilot data and translating them into targeted revisions of modular pipeline components, DATAFOUNDRY progressively adapts the preparation runtime to domain-specific requirements. Experiments across Math, Finance, Law, and Medical domains show that the resulting datasets consistently provide stronger downstream training utility and generalize across different base models. Our analysis further demonstrates that recursive self-improvement can systematically enhance data quality, while highlighting the need to control excessive evolution.

REFERENCES

Mingguang Chen, Licheng Wang, and Bo Qu. Recursive self-improvement in ai: From bounded self-refinement to autonomous research loops, 2026. URL https://arxiv.org/abs/2607.07663.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations, 2023. URL https://arxiv.org/abs/2305.14233.

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, 2026. URL https://arxiv.org/abs/2605.10906.

Meihao Fan, Ju Fan, Yuxin Zhang, Shaolei Zhang, Xiaoyong Du, Jie Song, Peng Li, Fuxin Jiang, Tieying Zhang, and Jianjun Chen. Deepprep: An llm-powered agentic system for autonomous data preparation, 2026. URL https://arxiv.org/abs/2602.07371.

Che Jiang, Jincheng Zhong, Yu Fu, Kai Tian, Junlin Yang, Kaikai Zhao, Yuchong Wang, Tianwei Luo, Weizhi Wang, Yuxin Zuo, Guoli Jia, Xingtai Lv, Dianqiao Lei, Sihang Zeng, Yuru Wang, Zhenzhao Yuan, Xinwei Long, Ermo Hua, Can Ren, Xin Jiang, Shulei Xie, Yuanchun Zheng, Youbang Sun, Biqing Qi, Ning Ding, Kaiyan Zhang, and Bowen Zhou. Self-improving agents in the era of experience: A survey of self- to meta-evolution. *OpenReview Archive*, 2026. URL https://openreview.net/pdf?id=IUltZSgLMm.

Ilia Kulikov, Chenxi Whitehouse, Tianhao Wu, Yixin Nie, Swarnadeep Saha, Eryk Helenowski, Weizhe Yuan, Olga Golovneva, Jack Lanchantin, Yoram Bachrach, Jakob Foerster, Xian Li, Han Fang, Sainbayar Sukhbaatar, and Jason Weston. Autodata: An agentic data scientist to create high quality synthetic data, 2026. URL https://arxiv.org/abs/2606.25996.

Hyunin Lee, Jinglue Xu, Jeffrey Seely, Donghyun Lee, Matei Zaharia, and Yujin Tang. Recursive harness self-improvement, 2026a. URL https://arxiv.org/abs/2607.15524.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026b.

Jijie Li, Li Du, Hanyu Zhao, Bo wen Zhang, Liangdong Wang, Boyan Gao, Guang Liu, and Yonghua Lin. Infinity instruct: Scaling instruction selection and synthesis to enhance language models, 2025. URL https://arxiv.org/abs/2506.11116.

Xian Li, Ping Yu, Chunting Zhou, Timo Schick, Omer Levy, Luke Zettlemoyer, Jason E Weston, and Mike Lewis. Self-alignment with instruction backtranslation. In *International Conference on Learning Representations*, volume 2024, pp. 3552–3577, 2024.

Hao Liang, Xiaochen Ma, Zhou Liu, Zhen Hao Wong, Zhengyang Zhao, Zimo Meng, Runming He, Chengyu Shen, Qifeng Cai, Zhaoyang Han, Meiyi Qiang, Yalin Feng, Tianyi Bai, Zewei Pan, Ziyi Guo, Yizhen Jiang, Jingwen Deng, Qijie You, Peichao Lai, Tianyu Guo, Chi Hsu Tsai, Hengyi Feng, Rui Hu, Wenkai Yu, Junbo Niu, Bohan Zeng, Ruichuan An, Lu Ma, Jihao Huang, Yaowei Zheng, Conghui He, Linpeng Tang, Bin Cui, Weinan E, and Wentao Zhang. Dataflow: An llm-driven framework for unified data preparation and workflow automation in the era of data-centric ai, 2025. URL https://arxiv.org/abs/2512.16676.

Hao Liang, Qifeng Cai, Yibo Lin, Jianzhuo Du, Qifeng Xia, Sizhe Qiu, Linzhuang Sun, Meiyi Qiang, Zhaoyang Han, Xiaochen Ma, Bohan Zeng, Ruichuan An, Conghui He, and Wentao Zhang. Dataprep-bench: Benchmarking llms as training data preparators, 2026. URL https://arxiv.org/abs/2607.20465.

Chen Ling, Xujiang Zhao, Jiaying Lu, Chengyuan Deng, Can Zheng, Junxiang Wang, Tanmoy Chowdhury, Yun Li, Hejie Cui, Xuchao Zhang, et al. Domain specialization as the key to make large language models disruptive: A comprehensive survey. *ACM Computing Surveys*, 58(3): 1–39, 2025.

Wei Lu, Rachel K Luu, and Markus J Buehler. Fine-tuning large language models for domain adaptation: Exploration of training strategies, scaling, model merging and synergistic capabilities. *npj Computational Materials*, 11(1):84, 2025.

Ziyu Ma, Shidong Yang, Yuxiang Ji, Xucong Wang, Yong Wang, Yiming Hu, Tongwen Huang, and Xiangxiang Chu. Skillclaw: Let skills evolve collectively with agentic evolver, 2026. URL https://arxiv.org/abs/2604.08377.

Nihal Nayak, Yiyang Nan, Avi Trost, and Stephen Bach. Learning to generate instruction tuning datasets for zero-shot task adaptation. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 12585–12611, 2024.

Zhichao Shi, Cehao Yang, Hao Zhou, Xiaojun Wu, Huajie Li, Xuhui Jiang, Chengjin Xu, Yuanzhuo Wang, and Jian Guo. Dataarc-syndata-toolkit: A unified closed-loop framework for multi-path, multimodal, and multilingual data synthesis. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pp. 318–326, 2026.

Yash Vishe, Rohan Surana, Xunyi Jiang, Zihan Huang, Xintong Li, Nikki Lijing Kuang, Tong Yu, Ryan A. Rossi, Jingbo Shang, Julian McAuley, and Junda Wu. Skill-r1: Agent skill evolution via reinforcement learning, 2026. URL https://arxiv.org/abs/2605.09359.

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, 2023. URL https://arxiv.org/abs/2212.10560.

Lilian Weng. Harness engineering for self-improvement. *lilianweng.github.io*, July 2026. URL https://lilianweng.github.io/posts/2026-07-04-harness/.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning, 2026. URL https://arxiv.org/abs/2602.08234.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. Wizardlm: Empowering large pre-trained language models to follow complex instructions, 2025a. URL https://arxiv.org/abs/2304.12244.

Renjun Xu and Yang Yan. Agent skills for large language models: Architecture, acquisition, security, and the path forward, 2026. URL https://arxiv.org/abs/2602.12430.

Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin. Magpie: Alignment data synthesis from scratch by prompting aligned llms with nothing. In *International Conference on Learning Representations*, volume 2025, pp. 76346–76382, 2025b.

Cehao Yang, Xueyuan Lin, Xiaojun Wu, Chengjin Xu, Xuhui Jiang, Honghao Liu, Hui Xiong, and Jian Guo. Select2reason: Efficient instruction-tuning data selection for long-cot reasoning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 6656–6671, 2026a.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. Skillopt: Executive strategy for self-evolving agent skills, 2026b. URL https://arxiv.org/abs/2605.23904.

Yutao Yang, Junsong Li, Qianjun Pan, Bihao Zhan, Yuxuan Cai, Lin Du, Jie Zhou, Kai Chen, Qin Chen, Xin Li, Bo Zhang, and Liang He. Autoskill: Experience-driven lifelong learning via skill self-evolution, 2026c. URL https://arxiv.org/abs/2603.01145.

Ping Yu, Jack Lanchantin, Tianlu Wang, Weizhe Yuan, Olga Golovneva, Ilia Kulikov, Sainbayar Sukhbaatar, Jason Weston, and Jing Xu. Cot-self-instruct: Building high-quality synthetic prompts for reasoning and non-reasoning tasks. *arXiv preprint arXiv:2507.23751*, 2025.

Weihao Zeng, Can Xu, Yingxiu Zhao, Jian-Guang Lou, and Weizhu Chen. Automatic instruction evolving for large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 6998–7018, 2024.

Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, Xue Liu, Xiaoxiao Li, and Philip S. Yu. Coevoskills: Self-evolving agent skills via co-evolutionary verification, 2026. URL https://arxiv.org/abs/2604.01687.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models, 2024. URL https://arxiv.org/abs/2403.13372.

APPENDIX

A DETAILS OF SKILLS-AS-MODULES FRAMEWORK

A.1 CONSTRUCTION-RUNTIME-COMPILER

- **`corpus_chunk_manager`.** For each input document, the module partitions the document into sections according to its markdown heading hierarchy. It constructs content chunks subject to the length constraints `min_chars`=1200 and `max_chars`=3000. Sections shorter than the minimum length are merged, whereas sections exceeding the maximum length are split.
- **`instruction_builder`.** Given a content chunk, the module treats the chunk as the knowledge source and synthesizes a standalone instruction that is semantically aligned with both the source content and its domain. The prompt used in the **initial specification** is shown below:

instruction-generation prompt

```
<knowledge_context>
{source_chunk}
</knowledge_context>

Create one standalone instruction grounded in the knowledge context
    . The instruction must be answerable using the context, but
    must not mention or depend on access to the context. Return
    only the instruction and do not provide its answer.
```

- **`response_generator`.** The module takes the same content chunk together with the instruction produced by `instruction_builder` and generates a corresponding standalone response. The prompt used in the **initial specification** is as follows:

response-generation prompt

```
<knowledge_context>
{source_chunk}
</knowledge_context>

<instruction>
{instruction}
</instruction>

Write one standalone response to the instruction. Ground every
    factual claim in the knowledge context, but do not mention or
    depend on access to that context. Return only the response.
```

- **`batch_generation_service`.** The module concurrently invokes an external language model to generate training samples at scale. We use DeepSeek-V4-Flash[4] as the external generation model and set the maximum concurrency to 32.

[4] `https://api-docs.deepseek.com/`

A.2 CONTEXTUALIZED-DATA-ASSESSOR

data-assessment workflow

```
1. **Inspect the assessment context.** Resolve the target domain,
    pilot-set path, and output path.
2. **Load the pilot set.** Read the JSON array.
3. **Select criteria.** Load the library in `multiple_criteria` and
    explicitly choose one or more criteria that fit the domain and
    intended downstream use.
4. **Assess by sample.** Judge each candidate independently, one
    sample at a time using the domain context and the rubric in `
    multiple_criteria`.
5. **Generate the report.** Aggregate the selected criteria, and
    identify recurring deficiencies as self-contained dataset-level
    patterns. The diagnosis stays at the data-quality level for use by
    a downstream evolution module.
```

report-generation format

```
# Pilot-Set Assessment Report

## Context
- Domain: ...
- Pilot set: ...
- Selected criteria: ...

## Coverage
- Samples assessed: ...

## Criterion Summary
| Criterion | Count | Mean |
|---|---:|---:|

## Priority Deficiencies
- Self-contained finding describing the affected criterion, breadth,
    severity, and significance.
  - Representative examples: Sample N - brief excerpt or paraphrase
    and why it demonstrates the finding; Sample M - brief excerpt or
    paraphrase and why it demonstrates the finding.
```

A.3 EVOLUTION-ADAPTER-DESIGNER

adapter-designer workflow

```
1. **Review the current setup.** Read the baseline specification,
    current assessment report, and previous adapter if present.
    Separate baseline behavior from earlier domain changes.
2. **Select the priority problems.** Compare the recurring
    deficiencies in the report by prevalence, severity, and likely
    effect on training-data quality. Choose the problems that warrant
    changes.
3. **Trace them to modules.** Identify where each problem first
    appears in the construction process and select the modules
    responsible for that behavior.
4. **Design the adapters.** State the required module-level changes
    clearly enough for the runtime compiler to implement.
5. **Carry forward earlier changes.** Merge applicable requirements
    from the previous adapter into the final effective requirements for
     each module.
6. **Write the adapter.** Use the structure below.
```

report-generation format

```
# {Domain} Runtime Evolution Adapter

## Component Changes

### `module_name`
- State the final effective policy, prompt, configuration, or
    coordination requirements for this module.

## Added Modules
Include this section only when adding modules.

### `module_name`
- Responsibility: ...
- Input: ...
- Output: ...
- Workflow position: ...
- Required behavior: ...
```

B Downstream Training Setup

Table 4: Training configuration used in our experiments. We fine-tune all models with LlamaFactory (Zheng et al., 2024) using full-parameter supervised fine-tuning (SFT) following (Liang et al., 2026). Unless otherwise noted, the same configuration is used across all training runs.

Training Parameter	Setting
`fine_tuning_type`	Full
`cutoff_length`	4,096
`deepspeed_strategy`	ZeRO-3
`attention_backend`	FlashAttention-2
`precision`	bfloat16
`batch_size_per_device`	1
`gradient_accumulation`	4
`learning_rate`	5×10^{-6}
`training_epochs`	3
`lr_scheduler`	Cosine
`warmup_ratio`	0.1

C Evaluation Benchmarks

Table 5: Downstream evaluation benchmarks used in our experiments, following Liang et al. (2026). The released MATH-500 evaluation implementation uses 5,000 examples, on which the reported results were obtained. To avoid ambiguity, we refer to it as MATH-5000 throughout this paper.

Benchmark	Description	Count
	Mathematics	
GSM8K	Grade-school math word problems requiring multi-step reasoning.	1,319
AMC 2023	Multiple-choice problems from the 2023 American Mathematics Competitions.	40
AIME 2024	Competition-level mathematics problems from the 2024 American Invitational Mathematics Examination.	30
MinervaMath	Quantitative reasoning problems drawn from STEM and other technical subjects.	272
MATH-5000	A representative datasets of the challenging MATH exam set.	5000
Gaokao 2024	Mathematics questions from China's 2024 National College Entrance Examination.	91
OlympiadBench	Bilingual multimodal olympiad-level mathematics and science problems.	675
	Finance	
CPA-KQA	Expert-authored accounting and finance questions covering CPA-derived concepts.	210
FinEval-KR	Chinese financial knowledge-and-reasoning questions with annotations.	101
XFinBench	Multimodal, knowledge-intensive financial problems covering terminology, temporal reasoning, forecasting, planning, and numerical modelling.	435
	Law	
LegalBench	Collaboratively constructed legal-reasoning tasks designed with legal professionals.	413
LexGLUE	A standardized collection of legal natural-language-understanding datasets.	7,815
	Medical	
MedCaseReasoning	Clinical diagnostic cases paired with clinician-authored reasoning traces.	897
MedMCQA	Medical entrance-exam multiple-choice questions spanning 21 subjects.	4,183
MedR-Bench	Structured real-world clinical cases testing examination recommendation, diagnosis, and treatment planning.	957

D BASELINES IMPLEMENTATION

Table 6: Implementation details of the baseline generators from (Liang et al., 2026).

Generator	Operational mechanism
DataFlow-Based Generator	DataFlow executes a multi-step workflow, which processes and generates training data through predefined operators. DataFlow-Skill selects or creates DataFlow operators and compose them into an executable pipeline.
LLM-Based Generator	Prompts an LLM to convert source-book excerpts directly into QA pairs.
Agent-Based Generator	Runs a ReAct-style agent in a coding harness environment with a standard prompt. The agent iterates until it judges construction complete.
Skill-Based Generator	Augments the ReAct-style agent in a coding harness environment with an automatically loaded data construction skill. The agent iterates until it judges construction complete.

E PREPARED DATASET SIZE

Table 7: Number of training samples synthesized by each data construction method across domains, all operating on the *same* raw source corpus. The Total column aggregates yield across all domains.

Generator	Math	Finance	Law	Medicine	Total
Dolly-15k	-	-	-	-	15,011
DataFlow-based Generators					
DataFlow	90,222	111,774	53,891	253,355	509,242
DataFlow-Skill	165,345	49,841	21,659	113,305	350,150
LLM-based Generators					
Claude Opus 4.6	2,563	3,750	2,184	7,677	16,174
Gemini 3.0 Pro	1,619	1,584	893	2,992	7,088
GPT-5.2	9,179	5,832	3,949	14,945	33,905
Agent-based Generators					
Qwen3.5-Plus	684	21,693	13,712	25,965	62,054
GLM-4.7	684	23,246	13,326	24,734	61,990
Claude Opus 4.6	14,892	17,059	19,505	44,284	95,740
Gemini 3.0 Pro	5,340	249	389	465	6,443
GPT-5.2	14,337	11,999	4,967	30,765	62,068
GPT-5.3-codex	2,500	249	1,584	463	4,796
GPT-5.6 Sol	13,058	798	1,052	1,216	16,124
Skill-based Generators					
DataPrep-Skill (Opus 4.6)	1,439	106,401	24,255	38,351	170,446
DataPrep-Skill (GPT-5.6 Sol)	3,501	72,420	34,584	196,747	307,252
DataFoundry (GPT-5.6 Sol)	32,692	15,689	17,474	21,711	87,566

AI USE STATEMENT

In this work, we used generative AI tools to generate synthetic datasets, assist in developing theoretical models or conceptual frameworks, design or provide feedback on research methodology or experiments, implement methods, clean or reformat datasets, and assist with translation. We did not use generative AI tools to interpret results, or support qualitative or thematic data analysis. The use of generative AI to formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in writing proofs, or propose or refine hypotheses was not applicable to this work.

Additionally, we used generative AI tools to draft parts of the paper, summarize and analyze existing literature, identify research topics or gaps, support brainstorming, search for information, improve the readability of the manuscript, identify relevant literature, format references, and suggest possible titles and keywords.

We carefully reviewed all AI-assisted work. Specifically, generative AI was used to synthesize datasets for model training. These synthetic data were reviewed by multiple authors to ensure that they did not contain privacy leakage, harmful content, or other ethical issues. Generative AI was also used to assist in developing the conceptual framework, for example by checking and refining definitions of symbols, notation, and formulas. The resulting framework was independently reviewed and tested by multiple authors for factual correctness and internal consistency.

Generative AI assisted with method implementation, particularly in writing code for automated components of our system. AI-assisted code was reviewed, verified, and tested for correctness by multiple authors. Generative AI was also used for translation and language editing to improve the fluency and naturalness of the manuscript.

For literature-related tasks, generative AI was used to support brainstorming, literature search, and the summarization and analysis of prior work. The authors manually verified relevant literature, source attribution, and the resulting descriptions of prior work, and reviewed the manuscript to avoid unsupported claims or potential plagiarism. Generative AI also assisted with drafting and editing parts of the manuscript, improving readability, formatting references, and suggesting candidate titles and keywords. All AI-generated or AI-assisted text was reviewed and revised by the authors before inclusion in the final manuscript.

We take responsibility for the final content of this work, including all text, claims, code, data, and other artifacts produced with the aid of generative AI.