DataChef: Cooking Up Optimal Data Recipes for LLM Adaptation via Reinforcement Learning

Anonymous ACL submission

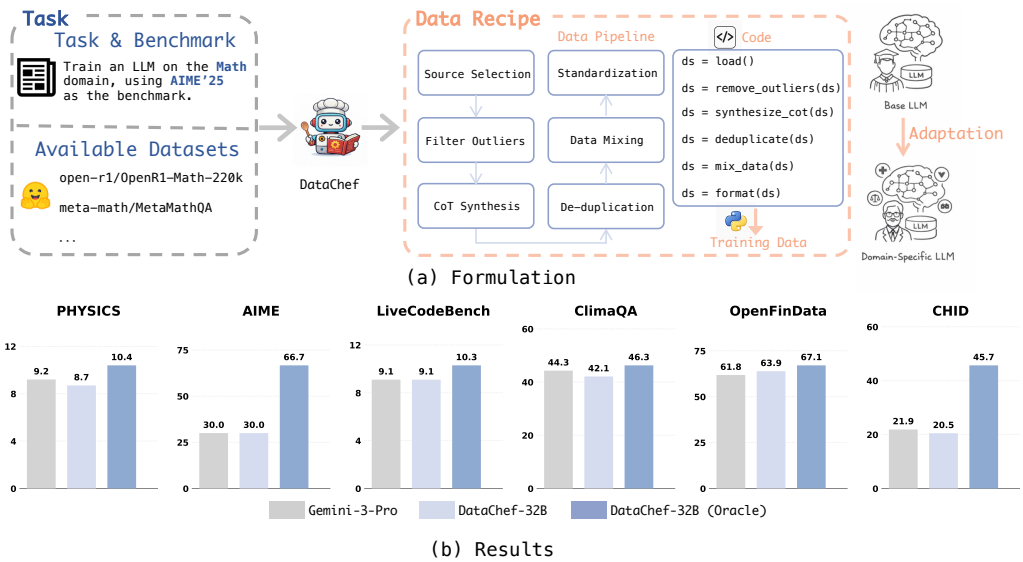

Figure 1: (a) **Formulation**. Given a task instruction, evaluation protocol, and raw data sources, a model is required to generate a data recipe, including an executable pipeline and the resulting training dataset, for LLM adaptation. (b) **Main results**. DataChef matches the performance of recipes from Gemini-3-Pro across six held-out tasks. See details in Sec. 4.2.

Abstract

In the current landscape of Large Language Models (LLMs), the curation of large-scale, high-quality training data is a primary driver of model performance. A key lever is the *data recipe*, which comprises a data processing pipeline to transform raw sources into training corpora. Despite the growing use of LLMs to automate individual data processing steps, such as data synthesis and filtering, the overall design of data recipes remains largely manual and labor-intensive, requiring substantial human expertise and iteration. To bridge this gap, we formulate *end-to-end data recipe generation* for LLM adaptation. Given a target benchmark and a pool of available data sources, a model is required to output a complete data recipe that adapts a base LLM to the target task. We present DataChef-32B, which performs online reinforcement learning using a proxy reward that predicts downstream performance for candidate recipes. Across six held-out tasks, DataChef-32B produces recipes that yield performance comparable to those curated by human experts. Notably, the recipe from DataChef-32B adapts Qwen3-1.7B-Base to the math domain, achieving 66.7 on AIME'25 and surpassing the official post-training checkpoint (Qwen3-1.7B). This work sheds new light on automating LLM training and developing self-evolving AI systems.

1 Introduction

The rapid evolution of LLMs (DeepSeek-AI et al., 2025; OpenAI, 2025) has precipitated a shift toward data-centric AI (Jakubik et al., 2024), identifying the composition and quality of training data as decisive factors in shaping model performance. In practice, constructing effective training data requires a well-designed multi-stage pipeline that processes heterogeneous raw data through a sequence of operations, such as transformation, filtering, mixing, synthesis, and refinement, tailored to specific training goals or stages (Yang et al., 2025; Cai et al., 2025). In this work, we formalize such processing pipelines with the resulting dataset as a *data recipe*.

Currently, formulating an effective data recipe relies heavily on human heuristics, where experts manually orchestrate data processing operations

and iteratively refine them based on empirical feedback (Penedo et al., 2025; Gururajan et al., 2024). While LLMs are widely used to automate individual pipeline components, such as data filtering (Liu et al., 2024; Zhang et al., 2025c) and synthesis (Mitra et al., 2024; Huang et al., 2024), they typically operate under rigid, hand-crafted prompts or algorithms. Recent studies have explored automating data pipeline orchestration to reduce the reliance on manual effort. In particular, Data-Juicer Sandbox (Chen et al., 2025a) proposes a Probe-Analyze-Refine workflow to identify the most impactful operators from a predefined operation pool, combine effective operations, and optimize data utilization through systematic experiments in data processing, model training, and evaluation with model performance as feedback. However, the continuous scaling of data and model sizes, coupled with the increasing complexity of processing operations, renders an exhaustive exploration of the combinatorial space of data recipes infeasible. Therefore, an essential question arises: can AI systems automatically generate a data recipe for training LLMs, including the orchestration of data pipelines and the implementation of each operation, in a cost-efficient way?

To bridge this gap, we introduce a new task: *end-to-end data recipe generation* for LLM adaptation. As shown in Fig 1(a), given a target benchmark and a pool of available data sources, the objective is to generate a data recipe by specifying the precise data processing pipeline to yield training data for adapting an LLM to the target task. This task requires sophisticated reasoning to analyze heterogeneous data sources, apply domain-specific processing operations, and generate executable code. While reinforcement learning with verifiable rewards (RLVR) has proven effective in enhancing LLM reasoning in domains such as coding (Zeng et al., 2025) and mathematics (Yeo et al., 2025), applying this paradigm to our task poses two key challenges: (1) **Data absence**: As a novel task, there are no curated datasets or standardized evaluation benchmarks for data recipe generation. (2) **Expensive and delayed supervision**: While downstream performance naturally serves as the reward signal, it is impractical to incorporate full LLM training into an online RL loop.

To address these challenges, we curate a comprehensive task pool comprising 31 widely used benchmarks across 19 distinct domains. These domains encompass reasoning-heavy fields, such as mathematics and coding, as well as knowledge-centric fields, such as finance, medicine, and natural sciences. The pool is partitioned into 25 training tasks and 6 held-out evaluation tasks, with each task supported by 8–15 source training datasets. We further propose a *Data Verifier* that estimates the utility of processed data without requiring model training, providing a low-latency proxy reward for scalable online RL. Empirical validation demonstrates that our Data Verifier correlates well with downstream performance and exhibits superior robustness across diverse tasks compared to existing data scoring metrics (Li et al., 2024a; Friedman and Dieng, 2023). Leveraging this task pool and proxy reward, we present DataChef-32B, an LLM specialized in generating optimal data recipes.

Extensive evaluations on 6 held-out tasks demonstrate that DataChef-32B matches the capabilities of the state-of-the-art proprietary model, Gemini-3-Pro, as illustrated in Fig. 1(b). Furthermore, recipes generated by DataChef-32B outperform the SOTA selection algorithm, DEITA (Liu et al., 2024), on most tasks, highlighting the superior potential of exploring a vast coding space over relying on hand-designed selection heuristics. Notably, our recipes adapt Qwen3-1.7B-Base to achieve 66.7 on AIME'25 (AIME, 2025) and 46.3 on ClimaQA (Manivannan et al., 2025), respectively, surpassing the official Qwen3-1.7B checkpoint with industry-level post-training on expert-curated data recipes.

In summary, our contributions are as follows:

- We formulate a new task, end-to-end data recipe generation for LLM adaptation, requiring models to automatically generate data recipes from a benchmark and available data sources.
- We construct a large-scale and diverse data pool covering 19 domains, 31 benchmarks, and 257 datasets to facilitate research in this area.
- We propose an efficient learning framework with a proxy reward that enables scalable online RL. Experiments show that DataChef-32B achieves performance comparable to top-tier proprietary models on the data recipe generation task.

2 Related Work

Data Pipelines. Many existing approaches rely on human experts to design individual data processing heuristics, including data mixing (Liu et al., 2025b), data sampling (Xu et al., 2024; Chen

et al., 2025d), and data synthesis (Chen et al., 2025c). General-purpose data processing frameworks (Chen et al., 2024a; Park et al., 2025) provide standardized modules and scalable pipeline construction for large-scale data processing, and are adopted to curate large-scale, high-quality training data, such as FinWeb2 (Penedo et al., 2025) for multilingual pre-training and Aloe (Gururajan et al., 2024) for medical-domain fine-tuning. However, their efficiency remains constrained by the manual pipeline design and iterative trial-and-error on downstream tasks. DataJuicer Sandbox (Chen et al., 2025a) marking a step further towards automated data pipeline construction by employing a Probe-Analyze-Refine workflow to assess operator effectiveness, but still relies on feedback derived from downstream model training, which is time and computation-consuming. In contrast, our work aims to end-to-end generate data recipes from scratch.

LLM Agents for Data Science. LLM agents automate data science via heuristic prompt-based workflows (Li et al., 2024b; Hong et al., 2025; Jiang et al., 2025; Chi et al., 2024), external knowledge retrieval (e.g., from Kaggle solutions or research paper) (Guo et al., 2024; Kulibaba et al., 2025; Nam et al., 2025), or learning-based approaches that fine-tune models through interaction (Liu et al., 2025c; Zhang et al., 2025b). However, these methods are typically evaluated on well-defined Kaggle benchmarks (Chan et al., 2025; Zhang et al., 2025a; Jing et al., 2025) using static datasets, and some use human-curated code as a starting point. In contrast, we address an open-ended setting, taking arbitrary tasks and available datasets as input and directly generating data recipes for LLM training.

Data Evaluation. Training and evaluating LLMs require significantly more computational resources, motivating the use of lightweight proxies to assess model performance (Chen et al., 2025a). Existing methods include handcrafted indicator-based metrics (Li et al., 2024a; Friedman and Dieng, 2023), model-based predictors (Ge et al., 2024; Liu et al., 2024), and LLM-as-a-judge prompting (Chen et al., 2024b). However, the systematic correlation between these assessment scores and downstream model performance remains underexplored, as prior work often relies on isolated data selection experiments. We bridge this gap by comprehensively analyzing the alignment between representative assessment methods and actual fine-tuning performance across diverse tasks.

3 Methodology

In this section, we first formalize some core concepts and define the data recipe generation task in Sec 3.1. Then, we introduce the specific data pool constructed for this study in Sec 3.2. Finally, we present our learning framework in Sec 3.3.

3.1 Problem Formulation

The goal of our method is to automatically generate a data recipe given a specific task. We formulate a **task** as a triplet $T = (I, \tau, \mathcal{D})$, where I is a natural language instruction, including description of the task requirement, along with meta-information of data sources and evaluation protocol, $\mathcal{D}$ denotes the set of available raw data sources, and τ is an evaluation metric that maps any model $\mathcal{M}$ to a scalar performance score $\tau(\mathcal{M}) \in \mathbb{R}$. A **data recipe** is formulated as $r = (g, d)$, where $g \in \mathcal{G}$ is a data pipeline and $d = g(\mathcal{D})$ is the resulting training dataset. In our experiments, the data pipeline is implemented as Python scripts.

Let $\mathcal{M}_\theta$ denote a language model. We use θ_d to present the parameters fine-tuned on a dataset d. We aim to learn a policy $\pi_\phi(r \mid T)$ that generates data recipes to maximize the expected downstream performance of the trained model. Formally, the objective function is defined as:

$$\mathcal{J}(\phi) = \mathbb{E}_{r \sim \pi_\phi(\cdot|T)}[\tau(\mathrm{LM}_{\theta_d})] \tag{1}$$

3.2 Task Pool Construction

Seed Task Curation. We construct a diverse task pool encompassing 19 heterogeneous domains, including reasoning, coding, and knowledge-intensive fields such as healthcare, finance, and natural science. As shown in Fig 2(a), for each domain, we curate representative benchmarks (e.g., GSM8K and AIME'25 for mathematics), totaling 31 benchmarks. For each benchmark, we retrieve relevant candidate datasets from Hugging Face, prioritizing those with high community engagement (downloads and likes), yielding a repository of 257 distinct data sources. From this collection, we construct 25 seed tasks for training and reserve 6 held-out tasks (3 in-domain, 3 out-of-domain) for evaluation. Details of the selected benchmarks and automatic retrieval procedure for candidate datasets are provided in Appx. A.1.

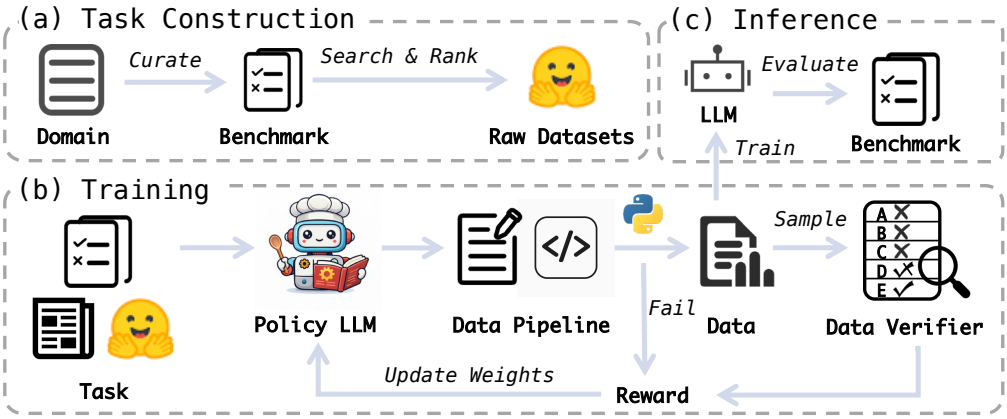

Figure 2: **Illustration of task construction and DataChef framework**. Given a task, a policy LLM generates a data recipe, which is executed to produce a training dataset. The Data Verifier then evaluates a sampled subset to provide a scalar reward, guiding the policy update via GRPO to optimize for data quality and executability.

Training Task Augmentation. To facilitate robust policy learning, we expand the 25 seed tasks into a large-scale training set $\mathcal{T}_{\text{train}}$. We employ a probabilistic sampling strategy where a benchmark τ is selected proportional to its source count $|\mathcal{D}|$, followed by uniform sampling of a subset $\mathcal{D}' \subseteq \mathcal{D}$ to form a new instance $T' = (I', \tau, \mathcal{D}')$. After deduplication, the expansion strategy yields 5K unique task instances.

3.3 End-to-end Data Recipe Generation

Framework Overview. As illustrated in Fig. 2, our framework optimizes the policy π_ϕ to generate high-quality data recipes. Given a task T, the policy generates a data pipeline g, which consists of a natural language plan for orchestrating data pipelines and its corresponding implementation as an executable code block. During training, the pipeline transforms raw data sources $\mathcal{D}$ into a training dataset d, which is then evaluated by the Data Verifier to guide policy updates via reinforcement learning. During inference, the data recipe is directly used for downstream model adaptation.

Cold-start Initialization. Training the policy from scratch using RL is non-trivial due to the low executability of data recipes, leading to sparse, high-variance rewards and ineffective exploration (Shao et al., 2024; Liu et al., 2025c). To mitigate this, we employ a cold-start Supervised Fine-Tuning (SFT) phase. We observe that decoupling reasoning and coding yields superior inference-time performance (detailed in Sec. 4.4). Therefore, we construct a demonstration set using a decoupled generation process: a strong reasoning model proposes plans, and a specialized coding model implements them. Rejection sampling is used to retain only high-quality rollouts based on execution success and data quality. Initializing π_ϕ on this curated dataset equips the policy with a foundational capability for code generation, significantly stabilizing the subsequent RL phase.

Reward Modeling. Ideally, the reward signal would be the downstream performance $\tau(\mathcal{M}_{\theta_d})$. However, using this as an online reward is computationally prohibitive due to the cost of repeated model training and evaluation. Instead, we design a computationally efficient surrogate reward based on the quality of the generated dataset d. Inspired by rubrics-based rewards (Gunjal et al., 2025), we employ a strong LLM as a *Data Verifier* to classify each instance $x \in d$ into one of five categories with assigned scalar scores $s(x)$:

- *Invalid* (0): Samples with missing essential information or severe repetition.
- *Format Error* (0): Samples violating explicit output format constraints.
- *Incorrect* (0): Samples containing factual errors or wrong answers.
- *Task Mismatch* (0.4): Valid samples that are semantically irrelevant to the target task I.
- *Pass* (1.0): High-quality samples that satisfy all criteria.

To ensure computational efficiency during online training, we estimate the dataset quality by randomly sampling a subset $\hat{d} \subset d$. Let $\bar{s}(\hat{d})$ be the average instance score over this sampled subset. We define the final recipe reward $\mathcal{R}(r)$ by incorporat-

Table 1: **Main Results on six held-out tasks**. We report the mean Data Verifier Score $\mathrm{DVS}_{\mathrm{avg@32}}$ and the Downstream Benchmark Score DBS, where the Average column presents $\mathrm{DBS}_{\mathrm{norm}}$ as a normalized score relative to $\textsc{Single-Source}_{\mathrm{best}}$ (100.0). Qwen3-Next-80B $\oplus$ Kimi-K2 denotes a combination using Qwen3-Next-80B for reasoning and Kimi-K2-Instruct for coding. DataChef-32B achieves performance comparable to the closed-source Gemini-3-Pro and significantly outperforms other open-source baselines across all settings.

Note: All Methods are fine-tuned from Qwen3-1.7B-Base								
Indomain Tasks								
Method	**PHYSICS**		**AIME**		**LIVECODE**		**Average**	
	$\mathrm{DVS}_{\mathrm{avg@32}}$ ↑	DBS ↑	$\mathrm{DVS}_{\mathrm{avg@32}}$ ↑	DBS ↑	$\mathrm{DVS}_{\mathrm{avg@32}}$ ↑	DBS ↑	$\mathrm{DVS}_{\mathrm{avg@32}}$ ↑	$\mathrm{DBS}_{\mathrm{norm}}$ ↑
Model Reference								
Qwen3-1.7B-Base	-	0.8	-	20.0	-	1.7	-	25.5
Qwen3-1.7B	-	20.8	-	33.3	-	25.7	-	123.3
Human Preprocess $\oplus$ Data Selection								
$\textsc{Single-Source}_{\mathrm{avg}}$	-	6.1	-	23.4	-	6.3	-	63.9
$\textsc{Single-Source}_{\mathrm{best}}$	-	8.5	-	39.6	-	10.3	-	100.0
$\textsc{Algorithm}_{\mathrm{IFD}}$	-	7.0	-	3.3	-	4.6	-	45.3
$\textsc{Algorithm}_{\mathrm{DEITA}}$	-	7.5	-	6.7	-	10.9	-	70.5
LLM-as-a-Chef								
Qwen3-32B	11.0	5.9	31.5	13.3	24.3	8.0	22.3	56.7
Kimi-K2	19.7	9.0	35.4	20.0	19.3	9.7	24.8	83.7
Qwen3-Next-80B $\oplus$ Kimi-K2	48.7	8.9	78.3	23.3	39.2	7.4	55.4	78.6
Gemini-3-Pro	69.7	9.2	80.7	30.0	53.6	9.1	68.0	91.2
DataChef-32B	61.4	8.7	84.7	30.0	45.8	9.1	64.0	89.3
DataChef-32B (Oracle)	-	10.4	-	66.7	-	10.3	-	130.3
Out-of-domain Tasks								
Method	**CLIMAQA**		**OPENFIN**		**CHID**		**Average**	
	$\mathrm{DVS}_{\mathrm{avg@32}}$ ↑	DBS ↑	$\mathrm{DVS}_{\mathrm{avg@32}}$ ↑	DBS ↑	$\mathrm{DVS}_{\mathrm{avg@32}}$ ↑	DBS ↑	$\mathrm{DVS}_{\mathrm{avg@32}}$ ↑	$\mathrm{DBS}_{\mathrm{norm}}$ ↑
Model Reference								
Qwen3-1.7B-Base	-	15.1	-	33.7	-	14.2	-	35.9
Qwen3-1.7B	-	44.2	-	73.4	-	59.8	-	101.0
Human Preprocess $\oplus$ Data Selection								
$\textsc{Single-Source}_{\mathrm{avg}}$	-	26.0	-	41.7	-	49.3	-	65.1
$\textsc{Single-Source}_{\mathrm{best}}$	-	43.6	-	63.7	-	70.3	-	100.0
$\textsc{Algorithm}_{\mathrm{IFD}}$	-	37.9	-	60.8	-	56.6	-	87.7
$\textsc{Algorithm}_{\mathrm{DEITA}}$	-	38.0	-	63.2	-	63.7	-	92.4
LLM-as-a-Chef								
Qwen3-32B	20.6	35.6	34.9	23.8	1.1	12.3	18.9	45.5
Kimi-K2	18.3	41.8	51.5	46.5	23.1	3.9	31.0	58.2
Qwen3-Next-80B $\oplus$ Kimi-K2	41.5	42.6	54.7	64.0	8.6	4.1	34.9	68.0
Gemini-3-Pro	58.4	44.3	54.9	61.8	29.7	21.9	47.6	76.6
DataChef-32B	57.3	42.1	67.0	63.9	7.9	20.5	44.1	75.4
DataChef-32B (Oracle)	-	46.3	-	67.1	-	45.7	-	92.2

ing penalties for execution failures:

$$\mathcal{R}(r) = \begin{cases} -\lambda_{\emptyset}, & \text{if } d = \emptyset \text{ (execution failure)}, \\ -\lambda_{\mathrm{fmt}}, & \text{if } d \text{ violates training format}, \\ \bar{s}(\hat{d}), & \text{otherwise}, \end{cases} \tag{2}$$

where $\lambda_{\emptyset}$ and λ_{fmt} are positive penalty coefficients. Please refer to Appx. A.2 for a detailed description of the category definitions used in the prompt and to Appx. E for computational cost analysis.

Reinforcement Learning. We employ Group Relative Policy Optimization (GRPO) for policy optimization. For each task $T \sim \mathcal{T}_{\mathrm{train}}$, we sample a group of G candidate data recipes $\{r_i\}_{i=1}^{G}$ from the current policy $\pi_{\phi_{\mathrm{old}}}$. The policy parameters are optimized by maximizing the following objective:

$$\mathcal{J}(\phi) = \mathbb{E}\Bigg[\frac{1}{G}\sum_{i=1}^{G} \min\Big(\rho_i A_i,\ \mathrm{clip}(\rho_i, 1-\epsilon, 1+\epsilon)\, A_i\Big) - \beta\, D_{\mathrm{KL}}\Big(\pi_{\phi} \,\|\, \pi_{\mathrm{ref}}\Big)\Bigg] \tag{3}$$

where $\rho_i = \frac{\pi_{\phi}(r_i|T)}{\pi_{\phi_{\mathrm{old}}}(r_i|T)}$ is the importance ratio, $A_i = \frac{\mathcal{R}(r_i)-\mu}{\sigma+\delta}$ is the group-relative advantage, ϵ is the clipping parameter, π_{ref} is a fixed reference policy, and β controls KL regularization.

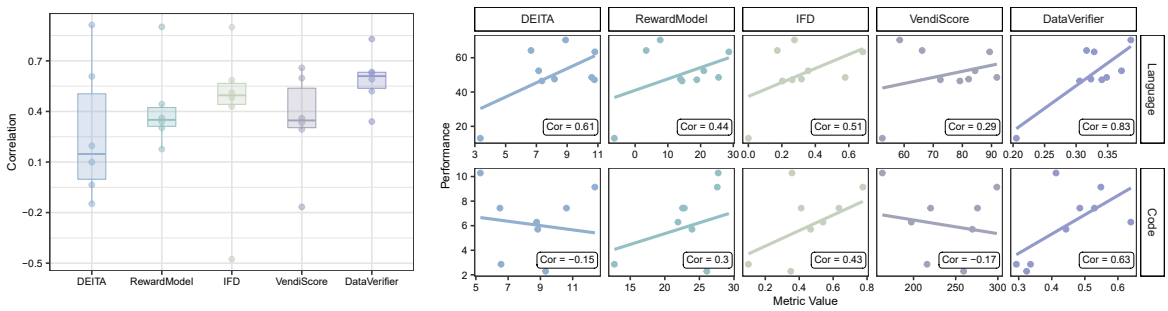

Figure 3: **Correlation analysis of data evaluation metrics**. (left) We summarize the Pearson correlation coefficients across all six evaluated tasks. (right) We detail the relationship between metric scores (X-axis) and downstream performance (Y-axis) on Language and Code tasks. The Data Verifier maintains a strong, consistent positive correlation across disparate domains. Please refer to Table 5 and Fig. 12 in Appx. D for complete results.

4 Experiments

4.1 Setups

Training. For cold-start SFT, we train Qwen3-32B (Yang et al., 2025) on 5K high-quality synthetic instances for 2 epochs, utilizing a learning rate of 2e-5 and a batch size of 32. In the RL phase, we further optimize the SFT checkpoint using GRPO (Shao et al., 2024) for 1 epoch on the same dataset, with a learning rate of 5e-7. During RL, the rollout batch size is set to 128 with a temperature of 1.0, and we sample 8 candidate data recipes per task.

Evaluation Set. We evaluate on 6 held-out tasks: 3 in-domain tasks and 3 out-of-domain tasks. Notably, these in-domain evaluation tasks share domains with the training set but remain strictly unseen during training. The in-domain benchmarks include PHYSICS (Feng et al., 2025), AIME'25 (AIME, 2025), and LiveCodeBench v6 (Jain et al., 2024); the out-of-domain benchmarks are OpenFinData (Information, 2023), ClimaQA (Manivannan et al., 2025), and CHID (Zheng et al., 2019).

Metrics. Executing recipes and performing downstream fine-tuning and evaluation are compute-intensive, rendering large-scale end-to-end evaluation impractical. Accordingly, for each evaluation task, we generate a candidate set of $N = 32$ independent data recipes. Based on this candidate set, we report two metrics: (1) $\mathrm{DVS}_{\mathrm{avg@32}}$: the mean Data Verifier Score across all 32 recipes. This metric quantifies the expected quality and stability of the policy, where recipes failing to yield valid training data are assigned a score of 0. (2) DBS: the Downstream Benchmark Score of a model trained on a single recipe, which is randomly sampled from the subset of candidates with valid execution (i.e., DVS > 0). This metric reflects the actual performance on the downstream benchmark of a successfully executed recipe. Additionally, to approximate the oracle upper bound for DataChef-32B, we select the most promising recipe from the candidate set and report its downstream score. For all downstream evaluation, we fine-tune Qwen3-1.7B-Base for 3 epochs with a learning rate of 2e-5 and a batch size of 64.

Baselines. We compare DataChef-32B against leading LLMs, including Qwen3-32B, Kimi-K2-Instruct (Team et al., 2025), Qwen3-Next-80B-A3B-Thinking, and Gemini-3-Pro (Google, 2025). Additionally, we incorporate the following results as reference: (1) To benchmark raw data quality and hand-designed strategies, we manually filter and format each available source, reporting the average and best downstream performance trained on single-source datasets. Furthermore, we apply SOTA data selection algorithms, IFD (Li et al., 2024a) and DEITA (Liu et al., 2024), to this manually pre-processed pool. (2) We report the performance of the official Qwen3-1.7B checkpoint, representing an industry-standard topline achieved via expert-curated data recipes. Detailed experiment setups are provided in Appx. B.

4.2 Main Results

Main Comparison. Table 1 presents the performance of DataChef-32B against baselines across in-domain and out-of-domain tasks. DataChef-32B achieves superior performance compared to a strong practical baseline, Qwen3-Next-80B $\oplus$ Kimi-K2, which leverages open-source state-of-the-art specialized models (Qwen3-Next-80B-A3B-Thinking for reasoning and Kimi-K2-Instruct for coding). Specifically, our end-to-end model surpasses this composite system with

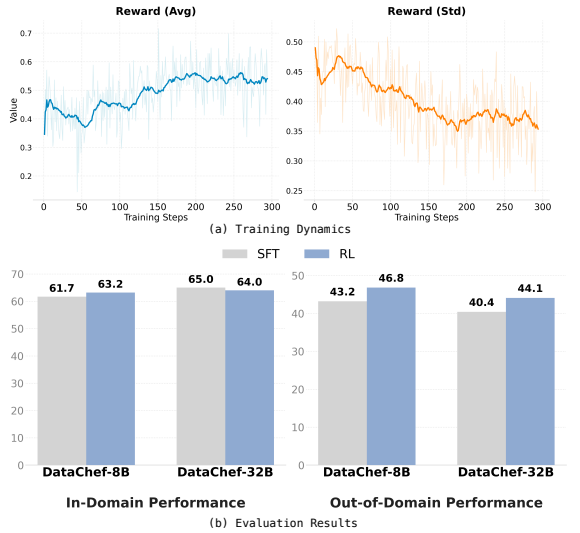

Figure 4: **Analysis of RL Effectiveness**. (a) RL training dynamics indicate that the policy consistently converges toward high-quality data recipe generation. (b) Evaluation results show that RL yields substantial improvements on out-of-domain tasks.

average improvements of +8.6% and +9.2% in $\text{DVS}_{\text{avg@32}}$, and +10.7% and +7.4% in DBS on in-domain and out-of-domain tasks, respectively. Notably, DataChef-32B achieves performance comparable to the closed-source top-tier Gemini-3-Pro, demonstrating exceptional robustness and effectiveness in automated data recipe generation.

Surpassing Human Baselines. By selecting the most promising recipe from 32 samples (Oracle Upper Bound), DataChef-32B outperforms $\text{SINGLE-SOURCE}_{\text{best}}$, IFD, and DEITA across most tasks, achieving an average in-domain score of 130.3. This indicates that DataChef goes beyond simple dataset selection and synthesizes novel data processing pipelines, including effective selection, mixing, synthesis, and filtering, thereby demonstrating the advantage of automatically exploring a vast code space over human-designed heuristics. Remarkably, it achieves 66.7 on AIME'25 and 46.3 on ClimaQA, surpassing the official checkpoint trained with industry-level, expert-curated data recipes. These results underscore the potential of fully automating data recipe generation for LLM training.

4.3 Data Verifier

Setup. To validate the proposed Data Verifier, we analyze the Pearson correlation between the verifier scores and downstream benchmark performance. We benchmark against several widely used data evaluation metrics, including IFD (Li et al., 2024a), RewardModelScore (Liu et al., 2025a), DEITA (Liu et al., 2024), and VendiScore (Friedman and Dieng, 2023). To ensure diversity in data quality and model performance, we construct 8–12 datasets per task under a fixed data budget using two strategies: (1) Direct sampling from available task-specific data sources. (2) Subset selection from the pool formed in (1) based on response length.

Correlation Analysis. As shown in Fig. 3 and detailed in Appx. D, the Data Verifier exhibits superior capability compared to existing metrics, achieving the highest average Pearson correlation of 0.59 across all six domains. Crucially, while baseline metrics suffer from severe cross-domain variance, frequently yielding negative correlations that can provide misleading optimization signals in domains such as Math (-0.48 for IFD) and Code (-0.15 for DEITA), our Data Verifier maintains a strictly positive correlation across all evaluated tasks. Furthermore, it demonstrates the highest statistical significance, with 4 out of 6 tasks satisfying $p < 0.1$, confirming that the Data Verifier provides a statistically robust and globally consistent signal for automated data recipe generation.

Table 2: **Ablation study on training stages and reward design**. We investigate the impact of the cold-start phase and the granularity of the reward signal. $\mathcal{R}_{\text{dense}}$ denotes our proposed fine-grained Data Verifier score, while $\mathcal{R}_{\text{sparse}}$ represents a constant success reward for valid execution.

Model	Cold Start	RL	Reward	Performance ($\text{DVS}_{\text{avg@32}}$)	
				In-Domain	Out-of-Domain
$\mathcal{M}_{Baseline}$	×	×	-	4.1	5.5
$\mathcal{M}_{RL}$	×	✓	$\mathcal{R}_{\text{dense}}$	32.9	23.9
$\mathcal{M}_{Sparse}$	✓	✓	$\mathcal{R}_{\text{sparse}}$	62.7	44.1
DataChef-8B	✓	✓	$\mathcal{R}_{\text{dense}}$	63.2	46.8

4.4 Ablation and Analysis

Effectiveness of RL. Fig. 4 illustrates that reward values consistently trend upward while the standard deviation decreases during training, confirming the convergence and effectiveness of RL process. Held-out evaluation reveal that RL primarily enhances generalization, yielding significant gains on out-of-domain tasks while preserving in-domain performance. Quantitatively, RL delivers an average $\text{DVS}_{\text{avg@32}}$ improvement of 3.6% for the 8B model and 3.7% for the 32B model.

Effectiveness of Cold Start. Table. 2 shows that omitting the cold start leads to significant performance degradation across all domains. To understand this behavior, we analyze the distribution of

Table 3: **Analysis of collaborating with strong coding models**. We compare the end-to-end paradigm against decoupled approaches where the model acts solely as a planner, relying on an external coder (Kimi-K2-Instruct) for implementation.

Model	External Coder	Performance ($DVS_{avg@32}$)	
		In-Domain	Out-of-Domain
Inference-Time			
Qwen3-32B	×	22.3	18.9
Qwen3-32B	✓	40.3	33.1
Training Paradigm			
Planner-32B	✓	56.7	37.3
DataChef-32B	×	**64.0**	**44.1**

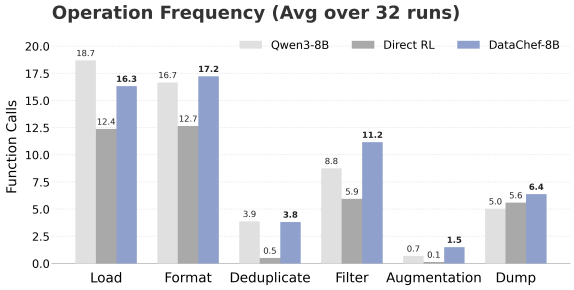

Figure 5: **Analysis of operation frequency in generated recipes**. We compare the average number of function calls per recipe across different models.

function calls in Fig. 5. The results demonstrate that the direct RL model tends to generate simplistic data pipelines, reducing the usage of complex data processing operations. We hypothesize that without the SFT warm-up, the model succumbs to *reward hacking*. It avoids execution penalties by generating safe, trivial scripts rather than optimizing for data quality. In contrast, DataChef-8B leverages the SFT foundation to explore and deploy sophisticated operations, such as filtering and data augmentation.

Ablation on Reward Signal. To assess the effectiveness of the fine-grained data quality feedback, we conduct an ablation where the continuous verifier score $s(\hat{d})$ in Eq. 2 is replaced by a constant success reward (i.e., assigning a fixed value of 1.0 to any valid data recipe). Table 2 demonstrates that this quality-agnostic signal leads to noticeable performance drops. This result confirms that the model relies on the guidance from the Data Verifier to distinguish high-utility recipes from merely executable ones.

Collaborating with Strong Coder. Given the proliferation of specialized coding models, a natural idea is to decouple this task: use the primary model as a planner (natural language orchestration) and an external coder for implementation. Table 3 shows that this paradigm enhances inference-time performance, with Qwen3-32B paired with Kimi-K2-Instruct yielding 18.0% and 14.5% $DVS_{avg@32}$ gains on in-domain and out-of-domain tasks, respectively. However, training the model solely as a planner leads to suboptimal results compared to the end-to-end approach. This suggests that integrated training of planning and coding capabilities is essential for optimal data recipe generation.

Case Study. We first present detailed examples of the Data Verifier in Fig. 10, demonstrating its capability to identify data anomalies and accurately classify samples into pre-defined categories. For DataChef-32B, we quantitatively analyze the data recipes generated by Qwen-32B and DataChef-32B for the out-of-domain financial task in Fig 6. By categorizing source datasets that yield strong downstream performance as *High-Perf Proxies* and those performing poorly as *Low-Perf Sources*, we observe that DataChef-32B exhibits an emergent ability to effectively identify and prioritize high-utility datasets. Additionally, we provide detailed data processing pipelines with code examples in Appx. C, which reveals that DataChef-32B can: (1) automatically leverage LLMs to augment data into task-specific formats or to synthesize data to enhance target ability; and (2) extract the most relevant data subsets using self-generated keywords.

5 Conclusion

In this paper, we propose a novel paradigm for automated data recipe generation to streamline LLM adaptation. To facilitate this, we establish a holistic dataset for both training and evaluation. Building on this foundation, we present DataChef-32B, incorporating a data verifier that serves as a cost-effective reward function for online RL. DataChef-32B demonstrates strong generalization capabilities, matching human-level expertise on specific benchmarks. Our work bridges the gap between data curation and model evolution, fostering the development of self-evolving AI.

Limitations. Our reliance on Data Verifier for proxy rewards prioritizes generalizability but may sacrifice precision in some tasks. Developing specialized evaluators to offer higher-resolution reward signals remains a valuable direction for future research. We present more details on limitations and future work in the Appx. F.

References

AIME. 2025. AIME problems and solutions.

Chenxin An, Shansan Gong, Ming Zhong, Mukai Li, Jun Zhang, Lingpeng Kong, and Xipeng Qiu. 2023. L-eval: Instituting standardized evaluation for long context language models. *arXiv preprint arXiv:2307.11088*.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2023. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*.

Mengzhang Cai, Xin Gao, Yu Li, Honglin Lin, Zheng Liu, Zhuoshi Pan, Qizhi Pei, Xiaoran Shang, Mengyuan Sun, Zinan Tang, Xiaoyang Wang, Zhanping Zhong, Yun Zhu, Dahua Lin, Conghui He, and Lijun Wu. 2025. Opendataarena: A fair and open arena for benchmarking post-training dataset value. *arXiv preprint arXiv:2512.14051*.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Mdry. 2025. Mle-bench: Evaluating machine learning agents on machine learning engineering. In *ICLR*.

Daoyuan Chen, Yilun Huang, Zhijian Ma, Hesen Chen, Xuchen Pan, Ce Ge, Dawei Gao, Yuexiang Xie, Zhaoyang Liu, Jinyang Gao, and 1 others. 2024a. Data-juicer: A one-stop data processing system for large language models. In *Companion of the 2024 International Conference on Management of Data*.

Daoyuan Chen, Haibin Wang, Yilun Huang, Ce Ge, Yaliang Li, Bolin Ding, and Jingren Zhou. 2025a. Data-juicer sandbox: A feedback-driven suite for multimodal data-model co-development. In *ICML*.

Ding Chen, Qingchen Yu, Pengyuan Wang, Mengting Hu, Wentao Zhang, Zhengren Wang, Bo Tang, Feiyu Xiong, Xinchi Li, Chao Wang, Minchuan Yang, and Zhiyu Li. 2025b. xverify: Efficient answer verifier for reasoning model evaluations. *arXiv preprint arXiv:2504.10481*.

Lichang Chen, Shiyang Li, Jun Yan, Hai Wang, Kalpa Gunaratna, Vikas Yadav, Zheng Tang, Vijay Srinivasan, Tianyi Zhou, Heng Huang, and Hongxia Jin. 2024b. Alpagasus: Training a better alpaca with fewer data. In *ICLR*.

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, and 39 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Yicheng Chen, Xiangtai Li, Yining Li, Yanhong Zeng, Jianzong Wu, Xiangyu Zhao, and Kai Chen. 2025c. Auto cherry-picker: Learning from high-quality generative data driven by language. In *CVPR*.

Yicheng Chen, Yining Li, Kai Hu, Zerun Ma, Haochen Ye, and Kai Chen. 2025d. Mig: Automatic data selection for instruction tuning by maximizing information gain in semantic space. In *Findings of the Association for Computational Linguistics: ACL 2025*.

Yizhou Chi, Yizhang Lin, Sirui Hong, Duyi Pan, Yaying Fei, Guanghao Mei, Bangbang Liu, Tianqi Pang, Jacky Kwok, Ceyao Zhang, Bang Liu, and Chenglin Wu. 2024. Sela: Tree-search enhanced llm agents for automated machine learning. *arXiv preprint arXiv:2410.17238*.

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. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, and 245 others. 2025. Deepseek-v3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*.

Kaiyue Feng, Yilun Zhao, Yixin Liu, Tianyu Yang, Chen Zhao, John Sous, and Arman Cohan. 2025. Physics: Benchmarking foundation models on university-level physics problem solving. In *Findings of the Association for Computational Linguistics: ACL 2025*.

Dan Friedman and Adji Bousso Dieng. 2023. The vendi score: A diversity evaluation metric for machine learning. In *TMLR*.

Yuan Ge, Yilun Liu, Chi Hu, Weibin Meng, Shimin Tao, Xiaofeng Zhao, Mahong Xia, Zhang Li, Boxing Chen, Hao Yang, and 1 others. 2024. Clustering and ranking: Diversity-preserved instruction selection through expert-aligned quality estimation. In *EMNLP*.

Google. 2025. Gemini 3 pro.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean Hendryx. 2025. Rubrics as rewards: Reinforcement learning beyond verifiable domains. *arXiv preprint arXiv:2507.17746*.

Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. 2024. Ds-agent: Automated data science by empowering large language models with case-based reasoning. In *ICML*.

Ashwin Kumar Gururajan, Enrique Lopez-Cuena, Jordi Bayarri-Planas, Adrian Tormos, Daniel Hinjos, Pablo Bernabeu-Perez, Anna Arias-Duart, Pablo Agustin Martin-Torres, Lucia Urcelay-Ganzabal, Marta Gonzalez-Mallo, and 1 others. 2024. Aloe: A family of fine-tuned open healthcare llms. *arXiv preprint arXiv:2405.01886*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *ICLR*.

Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Li Zhang, Lingyao Zhang, Min Yang, Mingchen Zhuge, Taicheng Guo, Tuo Zhou, Wei Tao, Robert Tang, Xiangtao Lu, and 9 others. 2025. Data interpreter: An LLM agent for data science. In *Findings of the Association for Computational Linguistics: ACL 2025*.

Yinya Huang, Xiaohan Lin, Zhengying Liu, Qingxing Cao, Huajian Xin, Haiming Wang, Zhenguo Li, Linqi Song, and Xiaodan Liang. 2024. Mustard: Mastering uniform synthesis of theorem and proof data. In *ICLR*.

Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Jiayi Lei, Yao Fu, Maosong Sun, and Junxian He. 2023. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. *arXiv preprint arXiv:2305.08322*.

East Money Information. 2023. Openfindata.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.

Johannes Jakubik, Michael Vössing, Niklas Kühl, Jannis Walk, and Gerhard Satzger. 2024. Data-centric artificial intelligence. *Business & Information Systems Engineering*, 66(4):507–515.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. 2025. Aide: Ai-driven exploration in the space of code. *arXiv preprint arXiv:2502.13138*.

Liqiang Jing, Zhehui Huang, Xiaoyang Wang, Wenlin Yao, Wenhao Yu, Kaixin Ma, Hongming Zhang, Xinya Du, and Dong Yu. 2025. Dsbench: How far are data science agents to becoming data science experts? In *ICLR*.

Josué Kpodo, Parisa Kordjamshidi, and A Pouyan Nejadhashemi. 2024. Agxqa: A benchmark for advanced agricultural extension question answering. *Computers and Electronics in Agriculture*.

Stepan Kulibaba, Artem Dzhalilov, Roman Pakhomov, Oleg Svidchenko, Alexander Gasnikov, and Aleksei Shpilman. 2025. Kompeteai: Accelerated autonomous multi-agent system for end-to-end pipeline generation for machine learning problems. *arXiv preprint arXiv:2508.10177*.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. 2017. Race: Large-scale reading comprehension dataset from examinations. In *EMNLP*.

Junyi Li, Xiaoxue Cheng, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. 2023. Halueval: A large-scale hallucination evaluation benchmark for large language models. *arXiv preprint arXiv:2305.11747*.

Ming Li, Yong Zhang, Zhitao Li, Jiuhai Chen, Lichang Chen, Ning Cheng, Jianzong Wang, Tianyi Zhou, and Jing Xiao. 2024a. From quantity to quality: Boosting llm performance with self-guided data selection for instruction tuning. In *ACL*.

Ziming Li, Qianbo Zang, David Ma, Jiawei Guo, Tuney Zheng, Minghao Liu, Xinyao Niu, Yue Wang, Jian Yang, Jiaheng Liu, Wanjun Zhong, Wangchunshu Zhou, Wenhao Huang, and Ge Zhang. 2024b. Autokaggle: A multi-agent framework for autonomous data science competitions. *arXiv preprint arXiv:2410.20424*.

Chris Yuhao Liu, Liang Zeng, Yuzhen Xiao, Jujie He, Jiacai Liu, Chaojie Wang, Rui Yan, Wei Shen, Fuxiang Zhang, Jiacheng Xu, Yang Liu, and Yahui Zhou. 2025a. Skywork-reward-v2: Scaling preference data curation via human-ai synergy. *arXiv preprint arXiv:2507.01352*.

Qian Liu, Xiaosen Zheng, Niklas Muennighoff, Guangtao Zeng, Longxu Dou, Tianyu Pang, Jing Jiang, and Min Lin. 2025b. Regmix: Data mixture as regression for language model pre-training. In *ICLR*.

Wei Liu, Weihao Zeng, Keqing He, Yong Jiang, and Junxian He. 2024. What makes good data for alignment? a comprehensive study of automatic data selection in instruction tuning. In *ICLR*.

Zexi Liu, Jingyi Chai, Xinyu Zhu, Shuo Tang, Rui Ye, Bo Zhang, Lei Bai, and Siheng Chen. 2025c. Ml-agent: Reinforcing llm agents for autonomous machine learning engineering. *arXiv preprint arXiv:2505.23723*.

Xingyu Lu, He Cao, Zijing Liu, Shengyuan Bai, Leqing Chen, Yuan Yao, Hai-Tao Zheng, and Yu Li. 2024. MoleculeQA: A dataset to evaluate factual accuracy in molecular comprehension. In *Findings of the Association for Computational Linguistics: EMNLP 2024*.

Veeramakali Vignesh Manivannan, Yasaman Jafari, Srikar Eranky, Spencer Ho, Rose Yu, Duncan Watson-Parris, Yian Ma, Leon Bergen, and Taylor Berg-Kirkpatrick. 2025. Climaqa: An automated evaluation framework for climate question answering models. In *ICLR*.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*.

Adrian Mirza, Nawaf Alampara, Sreekanth Kunchapu, Martiño Ríos-García, Benedict Emoekabu, Aswanth Krishnan, Tanya Gupta, Mara Schilling-Wilhelmi, Macjonathan Okereke, Anagha Aneesh, Amir Mohammad Elahi, Mehrdad Asgari, Juliane Eberhardt, Hani M. Elbeheiry, María Victoria Gil, Maximilian Greiner, Caroline T. Holick, Christina Glaubitz, Tim Hoffmann, and 16 others. 2024. Are large language models superhuman chemists? *arXiv preprint arXiv: 2404.01475*.

Arindam Mitra, Luciano Del Corro, Guoqing Zheng, Shweti Mahajan, Dany Rouhana, Andres Codas, Yadong Lu, Wei ge Chen, Olga Vrousgos, Corby Rosset, Fillipe Silva, Hamed Khanpour, Yash Lara, and Ahmed Awadallah. 2024. Agentinstruct: Toward generative teaching with agentic flows. *arXiv preprint arXiv:2407.03502*.

Jaehyun Nam, Jinsung Yoon, Jiefeng Chen, Jinwoo Shin, Sercan Ö Arık, and Tomas Pfister. 2025. Mlestar: Machine learning engineering agent via search and targeted refinement. In *NIPS*.

OpenAI, :, Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sebastien Bubeck, and 108 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*.

OpenAI. 2025. Introducing gpt-5.

Hyunbyung Park, Sukyung Lee, Gyoungjin Gim, Yungi Kim, Dahyun Kim, and Chanjun Park. 2025. Dataverse: Open-source etl (extract, transform, load) pipeline for large language models. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (System Demonstrations)*.

Guilherme Penedo, Hynek Kydlíček, Vinko Sabolčec, Bettina Messmer, Negar Foroutan, Amir Hossein Kargaran, Colin Raffel, Martin Jaggi, Leandro Von Werra, and Thomas Wolf. 2025. Fineweb2: One pipeline to scale them all–adapting pre-training data processing to every language. *arXiv preprint arXiv:2506.20920*.

Shi Qiu, Shaoyang Guo, Zhuo-Yang Song, Yunbo Sun, Zeyu Cai, Jiashen Wei, Tianyu Luo, Yixuan Yin, Haoxu Zhang, Yi Hu, Chenyang Wang, Chencheng Tang, Haoling Chang, Qi Liu, Ziheng Zhou, Tianyu Zhang, Jingtian Zhang, Zhangyi Liu, Minghao Li, and 33 others. 2025. Phybench: Holistic evaluation of physical perception and reasoning in large language models. *arXiv preprint arXiv:2504.16074*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y.K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Yiqing Shen, Zan Chen, Michail Mamalakis, Luhan He, Haiyang Xia, Tianbin Li, Yanzhou Su, Junjun He, and Yu Guang Wang. 2024. A fine-tuning dataset and benchmark for large language models for protein understanding. *arXiv e-prints arXiv:2406.05540*.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, , and Jason Wei. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong, Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, and 150 others. 2025. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*.

Yuan-Sen Ting, Tuan Dung Nguyen, Tirthankar Ghosal, Rui Pan, Hardik Arora, Zechang Sun, Tijmen de Haan, Nesar Ramachandra, Azton Wells, Sandeep Madireddy, and Alberto Accomazzi. 2024. AstroMLab 1: Who Wins Astronomy Jeopardy!? *arXiv e-prints arXiv:2407.11194*.

Yuning Wu, Jiahao Mei, Ming Yan, Chenliang Li, Shaopeng Lai, Yuran Ren, Zijia Wang, Ji Zhang, Mengyue Wu, Qin Jin, and Fei Huang. 2025. Writingbench: A comprehensive benchmark for generative writing. *arXiv preprint arXiv:2503.05244*.

Hu Xu, Saining Xie, Xiaoqing Ellen Tan, Po-Yao Huang, Russell Howes, Vasu Sharma, Shang-Wen Li, Gargi Ghosh, Luke Zettlemoyer, and Christoph Feichtenhofer. 2024. Demystifying clip data. In *ICLR*.

Wanghan Xu, Xiangyu Zhao, Yuhao Zhou, Xiaoyu Yue, Ben Fei, Fenghua Ling, Wenlong Zhang, and Lei Bai. 2025. Earthse: A benchmark for evaluating earth scientific exploration capability of llms. *arXiv e-prints arXiv:2505.17139*.

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, and 41 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *EMNLP*.

Ken Yano, Zheheng Luo, Jimin Huang, Qianqian Xie, Masaki Asada, Chenhan Yuan, Kailai Yang, Makoto Miwa, Sophia Ananiadou, and Jun'ichi Tsujii. 2025. ELAINE-medLLM: Lightweight English Japanese Chinese trilingual large language model for biomedical domain. In *Proceedings of the 31st International Conference on Computational Linguistics*.

Edward Yeo, Yuxuan Tong, Morry Niu, Graham Neubig, and Xiang Yue. 2025. Demystifying long chain-of-thought reasoning in llms. *arXiv preprint arXiv:2502.03373*.

Ming Yin, Yuanhao Qu, Ling Yang, Le Cong, and Mengdi Wang. 2025. Toward scientific reasoning in llms: Training from expert discussions via reinforcement learning.

Jie Ying, Zihong Chen, Zhefan Wang, Wanli Jiang, Chenyang Wang, Zhonghang Yuan, Haoyang Su, Huanjun Kong, Fan Yang, and Nanqing Dong. 2025. Seedbench: A multi-task benchmark for evaluating large language models in seed science. *arXiv preprint arXiv:2505.13220*.

Huaye Zeng, Dongfu Jiang, Haozhe Wang, Ping Nie, Xiaotong Chen, and Wenhu Chen. 2025. Acecoder: Acing coder rl via automated test-case synthesis. In *ACL*.

Dan Zhang, Sining Zhoubian, Min Cai, Fengzu Li, Lekang Yang, Wei Wang, Tianjiao Dong, Ziniu Hu, Jie Tang, and Yisong Yue. 2025a. Datascibench: An llm agent benchmark for data science. *arXiv preprint arXiv:2502.13897*.

Shaolei Zhang, Ju Fan, Meihao Fan, Guoliang Li, and Xiaoyong Du. 2025b. Deepanalyze: Agentic large language models for autonomous data science. *arXiv preprint arXiv:2510.16872*.

Yifan Zhang, Yifan Luo, Yang Yuan, and Andrew C Yao. 2025c. Autonomous data selection with zero-shot generative classifiers for mathematical texts. In *Findings of the Association for Computational Linguistics: ACL 2025*.

Zhexin Zhang, Leqi Lei, Lindong Wu, Rui Sun, Yongkang Huang, Chong Long, Xiao Liu, Xuanyu Lei, Jie Tang, and Minlie Huang. 2023. Safetybench: Evaluating the safety of large language models with multiple choice questions. *arXiv preprint arXiv:2309.07045*.

Chujie Zheng, Minlie Huang, and Aixin Sun. 2019. ChID: A large-scale Chinese IDiom dataset for cloze test. In *ACL*.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*.

Yuxin Zuo, Shang Qu, Yifei Li, Zhangren Chen, Xuekai Zhu, Ermo Hua, Kaiyan Zhang, Ning Ding, and Bowen Zhou. 2025. Medxpertqa: Benchmarking expert-level medical reasoning and understanding. *arXiv preprint arXiv:2501.18362*.

A Implementation Details of DataChef

A.1 Details of Task Pool

Full Benchmark List. Table 4 presents the full list of benchmarks used in our task pool, along with their domains and usage.

Table 4: List of benchmarks used in the task pool.

Domain	Benchmark	Usage
Code	HumanEval (Chen et al., 2021) LiveCodeBench v6 (Jain et al., 2024)	Train Test
Comprehension	OpenbookQA (Mihaylov et al., 2018) RACE (Lai et al., 2017)	Train Train
Instruction Following	IFEval (Zhou et al., 2023)	Train
Agriculture	AgXQA (Kpodo et al., 2024) SeedBench (Ying et al., 2025)	Train Train
Astronomy	Astrobench (Ting et al., 2024)	Train
Biology	Genome-Bench (Yin et al., 2025) MoleculeQA (Lu et al., 2024) ProteinLMBench (Shen et al., 2024)	Train Train Train
Chemistry	ChemBench (Mirza et al., 2024)	Train
Earth Science	EarthSE (Xu et al., 2025)	Train
General Knowledge	C-Eval (Huang et al., 2023) MMLU (Hendrycks et al., 2021)	Train Train
Medical	MedXpertQA (Zuo et al., 2025) MedQA (Yano et al., 2025)	Train Train
Finance	OpenFinData (Information, 2023)	Test
Atmosphere	ClimaQA (Manivannan et al., 2025)	Test
Physics	PHYBench (Qiu et al., 2025) PHYSICS (Feng et al., 2025)	Train Test
Long Context	L-Eval (An et al., 2023) LongBench (Bai et al., 2023)	Train Train
Math	GSM8K (Cobbe et al., 2021) AIME'25 (AIME, 2025)	Train Test
Reasoning	BBH (Suzgun et al., 2022) HotpotQA (Yang et al., 2018)	Train Train
Safety	HaluEval (Li et al., 2023) SafetyBench (Zhang et al., 2023)	Train Train
Writing	WritingBench (Wu et al., 2025)	Train
Language	CHID (Zheng et al., 2019)	Test

Datasets Retrieval Procedure. We automate the retrieval of candidate datasets relevant to the benchmark and task through the following procedure:

- **Keyword Synthesis.** We use an LLM to generate 3–5 high-relevance search keywords tailored to the downstream task.
- **Search.** Leveraging the generated keywords, we query both the Hugging Face and Google Search APIs to harvest a broad spectrum of potential datasets.
- **Rank.** To ensure dataset utility, we rank retrieved datasets by community popularity (e.g., likes) and retain the top-4 candidates for each keyword.
- **Verify.** We implement a strict verification protocol to preclude data leakage, explicitly ensuring no overlap exists between the retrieved candidates and the benchmark.

A.2 Prompt Templates and Model Selection

Data Verifier. We utilize gpt-oss-120b (OpenAI et al., 2025) as the Data Verifier. This is a Mixture-of-Experts (MoE) model with only about 5B active parameters per token, ensuring high inference speed. The detailed rubric-based prompt used for evaluation is presented in Fig. 9.

Cold-start Models. To construct high-quality cold-start supervision, we employ two specialized models: Qwen3-Next-80B-A3B-Thinking (Yang et al., 2025) for planning and reasoning, and Kimi-K2-Instruct (Team et al., 2025) for code implementation. The corresponding prompts are detailed in Fig. 7 and Fig. 8.

B Details of Experiments Setup

B.1 Data Evaluation Metrics Settings

We use the OpenDataArena-Tool[1] for data assessment, adhering to its default configurations. The specific settings for the data evaluation metrics used in our experiments are as follows:

- **IFD.** We employ Qwen2.5-3B-Instruct as the backend model to calculate the Instruction-Following Difficulty (IFD) score. Following (Li et al., 2024a), instances with an IFD score > 1 are treated as outliers. To ensure robust correlation analysis, we assign a score of 0 to these anomalies.
- **DEITA.** Following (Liu et al., 2024), we define the final data score as the product of the *Complexity Score* and the *Quality Score*. These scores are computed using the checkpoints provided in the official DEITA repository.
- **RewardModelScore.** We utilize Skywork-Reward-V2-Llama-3.1-8B-40M (Liu et al., 2025a) to compute the reward score, serving as a proxy for response quality.
- **VendiScore.** We employ Qwen3-Embedding-0.6B to compute sample embeddings and utilize Euclidean distance as the similarity metric to calculate VendiScore, measuring the diversity of the dataset.

B.2 Evaluation Setup

All downstream task evaluations are conducted using the OpenCompass framework[2]. The detailed settings for each benchmark are as follows:

[1] https://github.com/OpenDataArena/OpenDataArena-Tool

[2] https://github.com/open-compass/opencompass

Table 5: **Detailed correlation results across different domains**. For each domain, we report the Pearson correlation coefficient (r) and the corresponding p-value.

Metric	PHYSICS		AIME		LiveCodeBench		ClimaQA		OpenFinData		CHID		Summary		
	r	p	r	p	r	p	r	p	r	p	r	p	Positive	$p < 0.05$	$p < 0.1$
IFD	0.48	0.19	-0.48	0.12	0.43	0.25	0.90	0.00	0.59	0.10	0.51	0.16	5/6	1/6	2/6
Deita	0.20	0.61	-0.04	0.91	-0.15	0.70	0.91	0.00	0.10	0.80	0.61	0.08	4/6	1/6	2/6
RewardModel	0.34	0.37	0.18	0.58	0.30	0.43	0.90	0.00	0.36	0.34	0.44	0.23	6/6	1/6	1/6
VendiScore	0.36	0.34	0.33	0.29	-0.17	0.67	0.66	0.08	0.60	0.09	0.29	0.44	5/6	0/6	2/6
DataVerifier	0.63	0.07	0.59	0.04	0.63	0.07	0.52	0.19	0.34	0.37	0.83	0.01	6/6	2/6	4/6

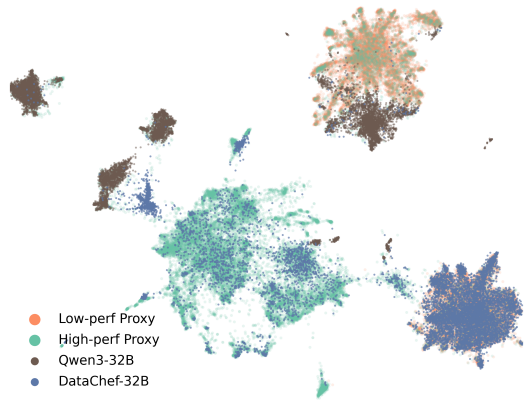

Figure 6: **Visualization of data distribution in generated recipes**. We project the source datasets and the data recipes generated by different models into a 2D embedding space.

- **PHYSICS.** We employ xVerify-9B-C (Chen et al., 2025b) as the evaluator and report the average accuracy across all sub-tasks.
- **AIME'25.** We evaluate on the 2025 subset (covering both Part I and Part II). For each question, we generate 8 responses and report the average accuracy. xVerify-9B-C is used as the evaluator.
- **LiveCodeBench v6.** We utilize the official prompt guidelines and report the pass@1 metric. The LCBCGenerationEvaluator is used for assessment.
- **ClimaQA.** We employ xVerify-9B-C as the evaluator and report the average accuracy across all sub-tasks.
- **OpenFinData.** We use the OpenFinDataKWEvaluator and report the average accuracy across all sub-tasks.
- **CHID.** We report the average accuracy on both the development and test sets.

B.3 Oracle Selection

The specific procedure for Oracle Selection is:

1. Filter: From the 32 generated recipes, we select the top-8 candidates based on the Data Verifier Score.

2. Verify: A human expert reviews these top-8 candidates to ensure quality, verifying key aspects including: (a) format alignment with the target benchmark (e.g., the processed dataset should include multiple-choice structures for MCQ tasks); (b) context integrity (e.g., verifying that inputs include necessary reference texts for reading comprehension, rather than just questions); and (c) logical comprehensiveness of the pipeline operations. Based on this comprehensive assessment, the expert selects the optimal candidate for final training.

B.4 Budget

For DBS evaluation, the data budget is set to 10K. If a recipe generates more, we randomly downsample to 10K; if fewer, we use all. During training and DVS evaluation, we use a budget of 100 samples for rapid proxy reward calculation.

C Case Study

Data Verifier. We first present detailed examples of the Data Verifier in Fig. 10, demonstrating its capability to identify data anomalies and accurately classify samples into pre-defined categories.

DataChef. We next present a complete data processing pipeline generated by DataChef-32B for the ClimaQA task. As shown in Fig. 11, the generated code successfully produces valid training data by: (1) automatically leveraging LLMs to augment data into task-specific formats and synthesize samples to enhance target capabilities; and (2) extracting the most relevant data subsets using self-generated keywords.

D Additional Results on Correlation Analysis

We provide the comprehensive correlation analysis results across all six evaluation tasks in Fig. 12.

These results validate the robustness of our Data Verifier compared to baseline metrics across diverse domains.

E Computational Cost Analysis

For SFT, the average per-step time is 2.4 minutes with a batch size of 32 on 8 H200 GPUs. For RL, we allocate 8 H200 GPUs to the policy model and 2 H200 GPUs to the Data Verifier deployment. A single RL step, which processes 128 candidate recipes, takes an average of 20.2 minutes. Notably, the Data Verifier inference accounts for less than 2 minutes of this duration, demonstrating its high computational efficiency.

F Limitations and Future Work

Predicting downstream model performance without actual training remains a challenging open problem, primarily because data utility is inherently model-dependent. For instance, data that benefits a weaker model might yield no gain or even negatively impact a more capable model that requires more challenging samples. Although the correlation analysis in Sec. 4.3 demonstrates the superior correlation of our proposed Data Verifier over mainstream baselines, and the ablation study in Table 2 validates the effectiveness of this proxy reward, it still serves as an approximation and cannot provide an absolutely precise or perfectly linear reward signal in all scenarios. In this work, the Data Verifier serves as an efficient, learnable reward signal for optimizing data recipe generation. Furthermore, from an evaluation perspective, the Data Verifier Score (DVS) allows us to assess generation stability across multiple candidate recipes (analogous to a pass rate or success rate), providing a crucial supplementary metric to the heavy Downstream Benchmark Score (DBS). A promising direction for future work involves developing more precise and robust data evaluators. Such evaluators would provide higher-resolution reward signals and further alleviate the high costs associated with traditional model evaluation.

Prompt for Data Recipe Generation (Natural Language)

```
# Task Description
{{ task_description }}
# Benchmark
## {{ benchmark.name }}
{{ benchmark.description }}
# Available Hugging Face Training Datasets
{% for item in datasets -%}
## {{ item.dataset_id }}
{{ item.examples }}
{% endfor -%}
```

—

Based on the Task Description, the target Benchmark, and the Available Hugging Face Training Datasets, design a feasible **Data Processing Plan**. This plan must include: (1) **Data Selection**: Identify the most suitable raw datasets from the available list. (2) **Data Processing Workflow**: Define the pipeline to transform selected raw data into high-quality SFT training data.
The plan will serve as a blueprint for code generation and data production. Ensure it is comprehensive, actionable, and free from ambiguous or vague statements.
A high-quality SFT dataset must exhibit the following attributes:

- **High Quality**: Accurate samples free from noise.
- **Diversity**: Coverage of varied instructions and objectives.
- **Relevance**: Strong alignment with the target vertical domain.

Important Constraints & Guidelines:

- **Source Selection**: strictly select datasets provided in the context. Prioritize those with complete metadata and clear field definitions. **Strictly prohibit data contamination: DO NOT use benchmark data for training.** Do not hallucinate datasets, splits, or configurations not provided in the context.
- **Grounding**: Base the processing workflow solely on the actual fields and content of the selected datasets. Do not make assumptions about data fields that do not exist.
- **Context**: Ensure inputs and outputs form coherent Q&A turns. If the output relies on specific context (e.g., a document or snippet) within the data, design the workflow to explicitly embed this context into the input.
- **Format Alignment**: If the benchmark requires specific formats (e.g., multiple-choice, JSON output), ensure the constructed training data aligns with these requirements.
- **Final Output Format**: The pipeline must produce data in the following standard dialogue format:
  `{"dialogs": [{"role": "user", "content": "..."}, {"role": "assistant", "content": "..."}]}`
- **LLM Utilization**: Flexibly leverage LLM inference capabilities for tasks such as: extracting instructions from heterogeneous documents, data augmentation/synthesis, and quality verification. Clearly specify the prompt design strategies for these steps.

Output Format: Generate the plan strictly in the following format (do not include preamble or additional text):

```
## Training Data
[
{"dataset_id": "target_id", "split": "target_split", "name": "config_name", "sample_num": "int",
"reason": "justification"},
...
]
## Data Processing Workflow
[Detailed Data Processing Workflow description]
```

Figure 7: Prompt used for data recipe generation (Natural Language).

Prompt for Data Recipe Generation (Executable Code)

```
# Available Hugging Face Training Datasets
{% for item in datasets -%}
## {{ item.dataset_id }}
{{ item.examples }}
{% endfor -%}
# Data Processing Plan
{{ plan }}
# Tool Information
{{ tool_info }}
```

—

Based on the Available Hugging Face Training Datasets, the Data Processing Plan, and the Tool Information, generate the executable **Data Processing Script**.
To validate the correctness of the processing, also generate a corresponding **Verification Script**.
Important Notes & Constraints:

- **Implementation Logic**: The script must clearly implement the processing rationale and utilize tools correctly.
- **Verification Scope**: The verification script should validate that the generated data matches expectations and confirm the effectiveness of core processing steps.
- **Target Format**: The final data must follow the ShareGPT format: `{'dialogs': ['role': 'user', 'content': '...', 'role': 'assistant', 'content': '...']}`. Use the `format_to_sharegpt` utility (import from `aidp`).
- **Output Directory**: Save the processed data to the `data/processed/` directory.

Output Format: Generate exactly two Python code blocks in the following format, without any additional text:

```
“‘python
# data-processing code block
“‘
“‘python
# test code block
“‘
```

Figure 8: Prompt used for data recipe generation (Executable Code).

Prompt for Data Verifier

As a grading expert, your task is to determine whether the candidate's response matches the question and to assess the sample's usefulness for the specified task. Follow these evaluation guidelines precisely:

Evaluation Protocol:

1. Validity Check:

- **Reject questions** that are: INCOMPLETE (cut off), REPETITIVE (loops), or NOT_ENOUGH_INFO.
- **Reject answers** that are: INCOMPLETE, REPETITIVE, REFUSAL (e.g., "I cannot answer..."), or IRRELEVANT.
- **Action**: Classify as `\boxed{A}` and specify the reason (e.g., `\boxed{A} - INCOMPLETE`).

2. Format Check:

- Ensure the answer follows any explicit output-format requirements (e.g., single choice, JSON schema).
- If no explicit format is required, pass.
- **Action**: If violated, classify as `\boxed{B} - FORMAT_ERROR`.

3. Correctness Check:

- Re-generate a concise reference answer and compare it with the candidate's final answer.
- For multi-part questions, require all parts to be correct; partial correctness → Fail.
- **Action**: If mismatched, classify as `\boxed{C} - INCORRECT`.

4. Task-Alignment Check (Training-Suitability):

- Evaluate if the sample (Q+A) is useful for training the specified task.
- **Scope Fit**: Targets the capabilities the task trains.
- **I/O Contract Impact**:
  - *Beneficial/Benign (ALIGN)*: Mild deviations that still teach the target mapping (e.g., adding a brief rationale where not strictly forbidden).
  - *Fatal Mismatch (MISMATCH)*: Conflicts likely to cause inference failure (e.g., task requires JSON but sample invents schema; task requires single letter but sample gives long essay without choice).
- **Action**: If harmful noise or fatal I/O mismatch → `\boxed{D} - TASK_MISMATCH`. Else → `\boxed{E} - PASS`.

Grading Scale:

- `\boxed{A} - INVALID`: Fails validity criteria.
- `\boxed{B} - FORMAT_ERROR`: Fails format check.
- `\boxed{C} - INCORRECT`: Deviates from reference answer.
- `\boxed{D} - TASK_MISMATCH`: Fails task-alignment check.
- `\boxed{E} - PASS`: Passes all checks.

Execution Steps:

1. Thoroughly evaluate validity, format, correctness, and task-alignment step-by-step.
2. If any check fails, immediately assign the corresponding grade.

Input Data:

<Task Description Begin>
{{ task_description }}
<Task Description End>
<Original Question Begin>
{{ question }}
<Original Question End>
<Original Answer Begin>
{{ llm_response }}
<Original Answer End>

Output Format: `Analysis step by step: [...]`
`Final Judgment: \boxed{GRADE} - REASON`

Figure 9: Prompt used for Data Verifier.

Task	Data Sample	DataVerifier	Clarification
PHYSICS	{"dialogs": [{"content": "A high-energy electron travels towards a stationary proton with a speed of 0.95c. What is the proton's kinetic energy after the electron scatters off it at an angle of 30 degrees relative to the original path of the electron?", "role": "user"}, {"content": "To solve this problem, we will use the conservation of momentum and relativistic energy. Let's denote the initial momentum of the electron as p_ei and the final momentum of the electron as p_ef. Similarly, let's denote the initial momentum of the proton as p_pi and the final momentum of the proton as p_pf. The initial momentum of the proton is zero since it is stationary.\n\nFirst, we need to find the initial energy and momentum of the electron. The relativistic energy-momentum relation is given by:\n\nE^2 = (m*c^2)^2 + (p*c)^2\n\nwhere E is the total energy, m is the mass, c is the speed of light, and p is the momentum. For the electron, we have:\n\nm_e = 9.11 * 10^-31 kg (mass of electron)\nv_e = 0.95c (speed of electron)\nc = 3 * 10^8 m/s (speed of light)\n\nThe relativistic momentum of the electron can be calculated using the formula:\n\np = m*v / sqrt(1 - (v/c)^2)\n\np_ei = m_e * v_e / sqrt(1 - (v_e/c)^2)\n\nNow, let's consider the final momenta of the electron and proton after scattering. We know that the momentum is conserved, so:\n\np_ei = p_ef + p_pf\n\nThe angle between the final momentum of the electron and the initial momentum is 30 degrees. We can break the final momentum of the electron into components:\n\np_ef_x = p_ef * cos(30)\np_ef_y = p_ef * sin(30)\n\nThe final momentum of the proton will also have x and y components:\n\np_pf_x = p_pf * cos(theta)\np_pf_y = p_pf * sin(theta)\n\nFrom the conservation of momentum in the x and y directions, we have:\n\np_ei = p_ef_x + p_pf_x\n0 = p_ef_y + p_pf_y\n\nNow we have two equations with two unknowns (p_ef and p_pf). We can solve these equations simultaneously to find the final momentum of the proton. Once we have the final momentum of the proton, we can find its final kinetic energy using the relativistic energy-momentum relation:\n\nE_pf^2 = (m_p*c^2)^2 + (p_pf*c)^2\n\nwhere m_p is the mass of the proton (1.67 * 10^-27 kg). The kinetic energy of the proton is then:\n\nK_p = E_pf - m_p*c^2\n\nBy solving the system of equations and calculating the final kinetic energy of the proton, we can find the answer to the problem.", "role": "assistant"}]}	A – Incomplete	1. The given information alone does not uniquely determine the proton's final kinetic energy. Additional assumptions are required, such as that the scattering is elastic. 2. The proposed response does not actually calculate Kp.
LiveCodeBench	{"dialogs": [{"content": "Please draft a Python code snippet encompassing no more than five lines of code, that not only identifies but also exhibits the first eight prime numbers. Your code must adhere to the concept of prime numbers and provide the eight initial instances essentially.", "role": "user"}, {"content": "Here's a solution in Python:\n\n```python\nprimes = []\nn = 2\nwhile len(primes) < 8:\n if all(n % p > 0 for p in primes): primes.append(n)\n n += 1\nprint(primes)\n```", "role": "assistant"}]}	B – Format Error	The response did not strictly output only code.
OpenFinData	{"dialogs": [{"content": "You are a financial sentiment analysis expert. Your task is to analyze the sentiment expressed in the given financial text.Only reply with positive, neutral, or negative.", "role": "system"}, {"content": "Ripple Effects Intron Health analyst Naresh Chouhan sees several reasons for more upside.", "role": "user"}, {"content": "negative", "role": "assistant"}]}	C – Incorrect	"More upside" clearly indicates a positive sentiment, so labeling it as "negative" is incorrect.
CHID	{"dialogs": [{"content": "Hi can you describe the music type of Aurora song runaway ", "role": "user"}, {"content": "Aurora's song \"Runaway\" is a track from the Norwegian singer-songwriter's debut studio album \"All My Demons Greeting Me as a Friend,\" which was released in 2016. "role": "assistant"}]}	D – Mismatch	The task is related to Chinese idioms, so this data is a mismatch.
AIME	{"dialogs": [{"content": "Let $x,$ $y,$ z be real numbers, all greater than 3, so that\n\\[\\frac{(x + 2)^2}{y + z - 2} + \\frac{(y + 4)^2}{z + x - X + \\frac{(z + 6)^2}{x + y - 6} = 36.\\]Enter the ordered triple $(x,y,z).$\nIf we know the answer to the above question is (10,8,6), what is the value of unknown variable X?", "role": "user"}, {"content": "We are given an equation involving the variables $x,$ $y,$ and $z,$ and we need to find the value of X.\nThe equation is:\n\\[\\frac{(x + 2)^2}{y + z - 2} + \\frac{(y + 4)^2}{z + x - X} + \\frac{(z + 6)^2}{x + y - 6} = 36.\\]\nWe are given that the solution to this equation is $(x,y,z) = (10,8,6)$.\nWe can substitute these values into the equation and solve for X:\n\\[\\frac{(10 + 2)^2}{8 + 6 - 2} + \\frac{(8 + 4)^2}{6 + 10 - X} + \\frac{(6 + 6)^2}{10 + 8 - 6} = 36.\\]\nSimplifying, we have:\n\\[\\frac{12^2}{12} + \\frac{12^2}{16 - X} + \\frac{12^2}{12} = 36.\\]\nSimplifying further, we get:\n\\[12 + \\frac{144}{16 - X} + 12 = 36.\\]\nCombining like terms, we have:\n\\[\\frac{144}{16 - X} = 12.\\]\nTo solve for X, we can multiply both sides of the equation by $16 - X$:\n\\[144 = 12(16 - X).\\]\nSimplifying, we get:\n\\[144 = 192 - 12X.\\]\nSubtracting 192 from both sides of the equation, we find:\n\\[-48 = -12X.\\]\nDividing both sides of the equation by -12, we get:\n\\[4 = X.\\]\nThe value of X is 4.\nThe answer is: 4", "role": "assistant"}]}	E – Pass	It passes all checks.

Figure 10: Case study of the Data Verifier.

```python

# data-processing code block
...

# 1. Load stackexchange_earthscience
stack_ds = load_remote_dataset(
    "mlfoundations-dev/stackexchange_earthscience",
    name="default",
    split="train",
    shuffle=False
)

# 1.1 Use LLM to convert Open QA into 4-option Multiple Choice Questions
SYSTEM_PROMPT_STACK = """
You are a science tutor.
Convert the given open QA pair into a 4-option multiple-choice question.
Return JSON only:
{"question": "<original question>", "A": "<option A>", "B": "<option B>", "C": "<
    option C>", "D": "<option D>", "answer": "<correct letter>"}
The correct answer must be one of the four options. Distractors should be
    scientifically plausible but wrong.
"""

def stack_parser(response: str, raw: dict) -> list:
    parsed = extract_json(response)
    if not parsed or "answer" not in parsed:
        return []
    opts = f"A) {parsed['A']}\nB) {parsed['B']}\nC) {parsed['C']}\nD) {parsed['D']}"
    user = f"Question: {parsed['question']}\nOptions:\n{opts}"
    assistant = parsed["answer"].lower()
    return [{"user": user, "assistant": assistant}]

stack_mc = generate_dataset_with_llm(
    stack_ds,
    system_prompt=SYSTEM_PROMPT_STACK,
    response_parser=stack_parser
)
```

Generated Data Recipe Code (ClimaQA Case Study Part I)

```python

# 2. Load ScienceQA_text_only
sciqa_ds = load_remote_dataset(
    "tasksource/ScienceQA_text_only",
    name="default",
    split="train",
    shuffle=False
)

# 2.1 Filter for climate-related topics
def climate_filter(sample):
    txt = (sample.get("category", "") + " " + sample.get("topic", "")).lower()
    return any(k in txt for k in ("climate", "weather", "atmosphere", "ocean", "
    environment"))

sciqa_climate = select_by_filter(sciqa_ds, climate_filter)

# 2.2 Standardize format
def sciqa_map(sample):
    choices = sample["choices"]
    opts = "\n".join([f"{chr(65+i)}) {c}" for i, c in enumerate(choices)])
    user = f"Question: {sample['question']}\nOptions:\n{opts}"
    assistant = chr(65 + sample["answer"])
    return {"user": user, "assistant": assistant}

sciqa_mc = sciqa_climate.map(sciqa_map, remove_columns=sciqa_climate.column_names)

# 3. Merge & Deduplicate
merged = concatenate_datasets([stack_mc, sciqa_mc])
deduped = deduplicate_by_text_hash(
    merged,
    text_map=lambda x: x["user"],
    lowercase=True,
    ignore_non_character=True
)

# 4. Convert to ShareGPT format
sharegpt_ds = format_to_sharegpt(
    deduped,
    user_map=lambda x: x["user"],
    assistant_map=lambda x: x["assistant"]
)

# 5. Save output
dump_dataset(sharegpt_ds, "data/processed/train_climaqa_style.jsonl")
```

Generated Data Recipe Code (ClimaQA Case Study Part II)

Figure 11: Case study of data recipe generation.

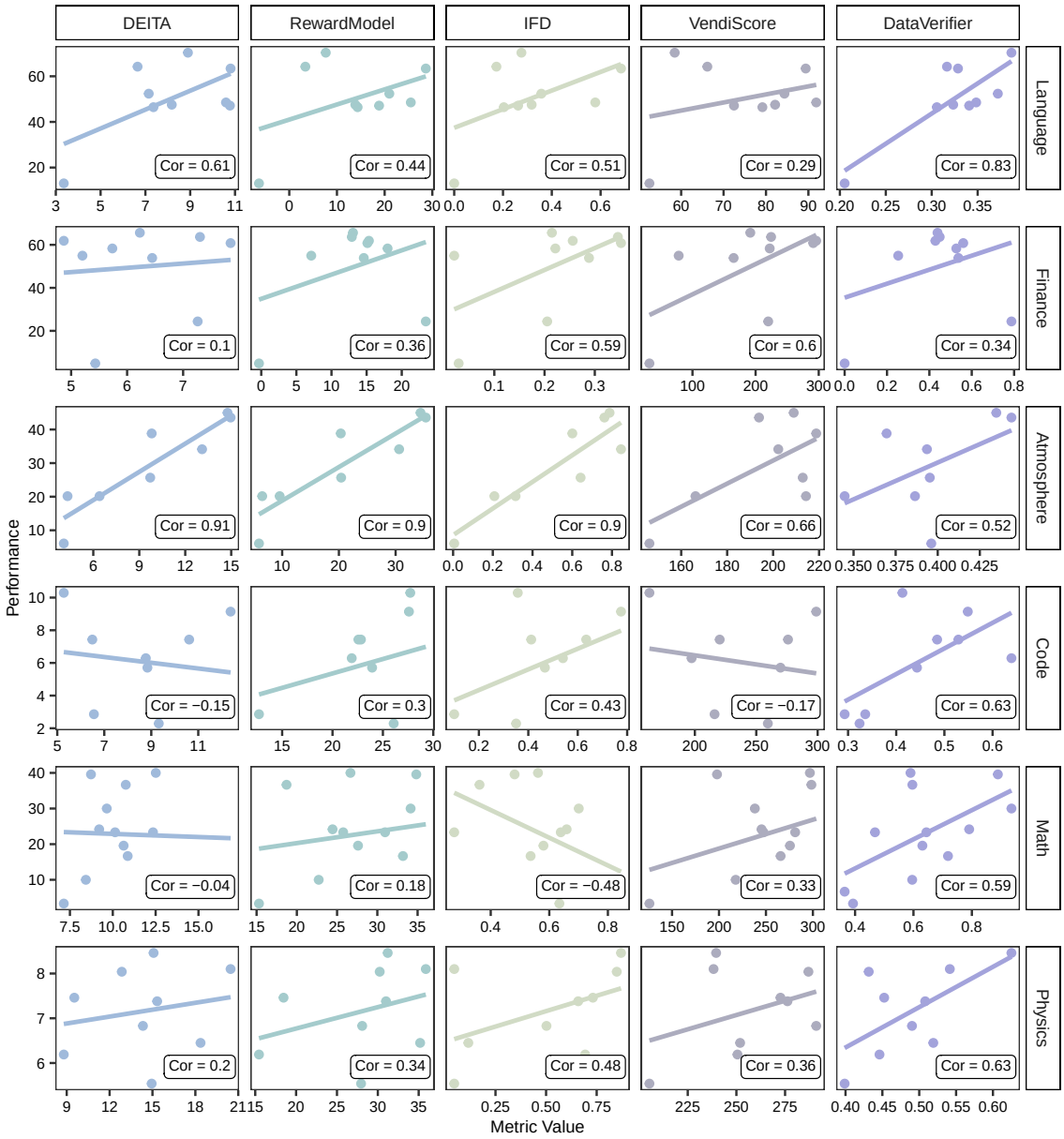

Figure 12: Complete results for correlation analysis.