CDR-Bench: Evaluating Faithful Execution of Compositional, Order-Sensitive Data Refinement Recipes

Anonymous ACL submission

Abstract

Data refinement involves executing multi-step recipes over evolving text states, where both composition and execution order of processing operators determine the outcome. While existing benchmarks either isolate text editing or entangle it with code and tool execution, it remains unclear whether LLMs can directly and faithfully execute these compositional, order-sensitive data refinement recipes. To fill this gap, we introduce CDR-Bench, a comprehensive benchmark featuring 3,319 high-quality tasks spanning four real-world data refinement domains and 29 distinct operators. Our benchmark evaluates models across atomic, order-agnostic, and order-sensitive settings, leveraging deterministic reference outputs to enable exact evaluation. Experiments on 10+ state-of-the-art LLMs reveal consistent failure patterns: performance degrades sharply in compositional settings, and order-sensitive recipe success collapses. These findings underline that current LLMs lack the procedural faithfulness required for reliable compositional data refinement.

1 Introduction

Data refinement, the process of transforming noisy and heterogeneous raw text into clean, task-ready data, is a core component of modern LLM pipelines. It underpins applications such as pre-training corpus construction (Chen et al., 2024a; Qin et al., 2025), reliable knowledge retrieval in RAG systems (Liu et al., 2026; Khan et al., 2024), and privacy-sensitive deployments (Garza et al., 2025; Pal et al., 2024). Traditionally, refinement pipelines have relied on heuristic rules and hand-crafted scripts (Lee et al., 2021; Li, 2019). Although reproducible, these pipelines become brittle as corpora, data policies, and downstream requirements evolve. The rise of LLMs enables a more flexible interface for data refinement, allowing users to specify goals in natural language across tasks such as text cleaning, quality filtering, information extraction, personally identifiable information (PII) redaction, and hallucination handling.

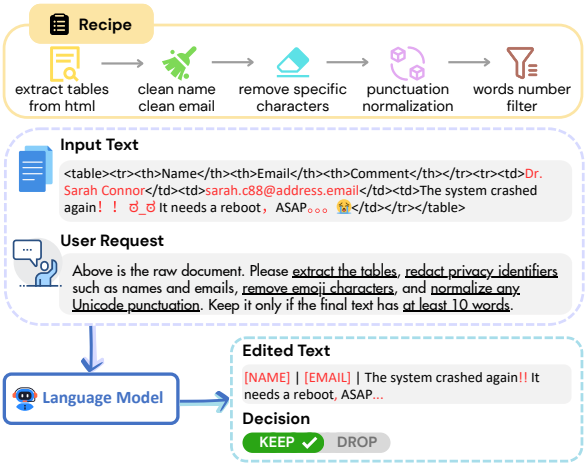

Figure 1: Illustration of a compositional data refinement process. The language model receives a raw input document alongside a user request specifying a multi-step recipe procedure, and directly outputs the processed target text together with a final execution judgment.

Despite this flexibility, these operations are rarely performed in isolation. Real-world data refinement requires executing a *compositional* and *order-sensitive* pipeline of interdependent operations over an evolving text state, where instructions dictate not only *what* to apply, but also *when*. Even if an LLM serves as a capable proxy for single, atomic operations, the compositional nature of refinement means that minor errors easily compound along the sequential execution, leading to eventual pipeline failure. Beyond mere composition, the order-sensitive requirement introduces an even more formidable challenge: Because each step during the pipeline execution can shift the context seen by subsequent steps, reordering operations can drastically alter intermediate states, the final edited text, and the ultimate keep/drop decision. Consequently, correctness in this setting is strictly procedural: the model must faithfully execute a

Benchmark	Input	Primary Output	Compositional Recipe	Order Sensitive	Evaluation Method
Task-Specific Editing Benchmarks					
TAB (Pilán et al., 2022)	Text	Edited Text	✗	✗	Human Annotation
EditEval (Dwivedi-Yu et al., 2022)	Text	Edited Text	✗	✗	Human Annotation
InstrEditBench (Zeng et al., 2026)	Text	Edited Text	✗	✗	Synthetic Annotation
OpenPII-1M (Ai4Privacy, 2026)	Text	Edited Text	✗	✗	Synthetic Annotation
Data Agent Benchmarks					
AutoDCWorkflow (Li et al., 2025)	Tables	Tables, Scripts	✗	✗	Execution-based
DataGovBench (Liu et al., 2025)	Tables, Text	Scripts	✓	✗	Execution-based
DCA-Bench (Huang et al., 2025)	Files	Reports	✗	✗	LLM-as-a-judge
KramaBench (Lai et al., 2026)	Files	Scripts	✓	✗	LLM, Execution-based
DAComp (Lei et al., 2025)	Files	Scripts, Reports	✓	✗	LLM, Execution-based
CDR-Bench (Ours)	Text	Decision, Edited Text	✓	✓	Deterministic Recipe Execution

Table 1: Scope comparison with representative task-specific editing benchmarks and data-centric agent benchmarks. CDR-Bench uniquely requires faithful, order-sensitive execution of compositional refinement recipes over raw text with deterministic evaluation.

latent sequence rather than merely generate a superficially plausible output.

However, as summarized in Table 1, existing benchmarks fail to evaluate this procedural faithfulness adequately. Standard instruction-driven text-editing benchmarks (Dwivedi-Yu et al., 2022; Zeng et al., 2026) focus on isolated edits, completely overlooking compositional and order-sensitive recipes. Conversely, while recent data-centric agents tackle multi-step pipelines (Liu et al., 2025; Lei et al., 2025), their end-to-end evaluations entangle the model's intrinsic procedural understanding with sandbox execution, debugging, and code generation. To untangle these factors and isolate the underlying model's genuine data refinement capability, *direct recipe execution* serves as a vital diagnostic primitive. As illustrated in Figure 1, it directly evaluates whether an LLM can independently apply the right operations in the right order to the right intermediate states without external scaffolding. This critical gap raises a question: can an LLM faithfully and directly execute compositional, order-sensitive data refinement recipes?

To answer this question, we introduce **CDR-Bench**, a comprehensive benchmark designed to evaluate LLMs on compositional, order-sensitive data refinement. CDR-Bench comprises 3,319 tasks spanning four real-world data refinement domains: Web Refinement, LaTeX Refinement, RAG Preparation, and Privacy Redaction, covering 29 diverse operators and 63 unique recipes. Crucially, all tasks carry deterministic reference outputs, enabling exact, objective evaluation without relying on LLM-as-a-judge paradigms. Our evaluation framework categorizes model capabilities into three distinct levels: atomic operator execution, order-agnostic recipe execution, and order-sensitive recipe execution. To rigorously assess performance across these levels, we further propose three complementary metrics: Recipe Success (RS) for exact match verification, Order-Consistent Success (OCS) to measure robustness against order permutations, and Refinement Gain (RG) for partial progress.

Through extensive experiments on over ten state-of-the-art LLMs, we find that compositional recipe execution exposes a fundamental limitation that single-operator performance does not predict. Models frequently produce outputs that look plausible on the surface while silently violating the intended execution order, and fewer than 5% of order-sensitive recipe groups are solved correctly across all tested orderings despite overall text quality remaining relatively stable. Deferred filtering decisions prove especially fragile, with exact recipe success falling by over 47 percentage points simply by moving a filter from before to after a sequence of transformations, a gap that neither chain-of-thought prompting nor few-shot demonstrations fully close. These findings suggest that progress in data refinement demands a shift from surface-level text editing to genuine procedural faithfulness, and CDR-Bench provides a reproducible testbed to measure and drive that progress. [1]

2 CDR-Bench

In this section, we introduce the definition, dataset construction pipeline, metrics and statistics.

[1]Anonymous code is available at https://anonymous.4open.science/r/cdrbench-review-80DF.

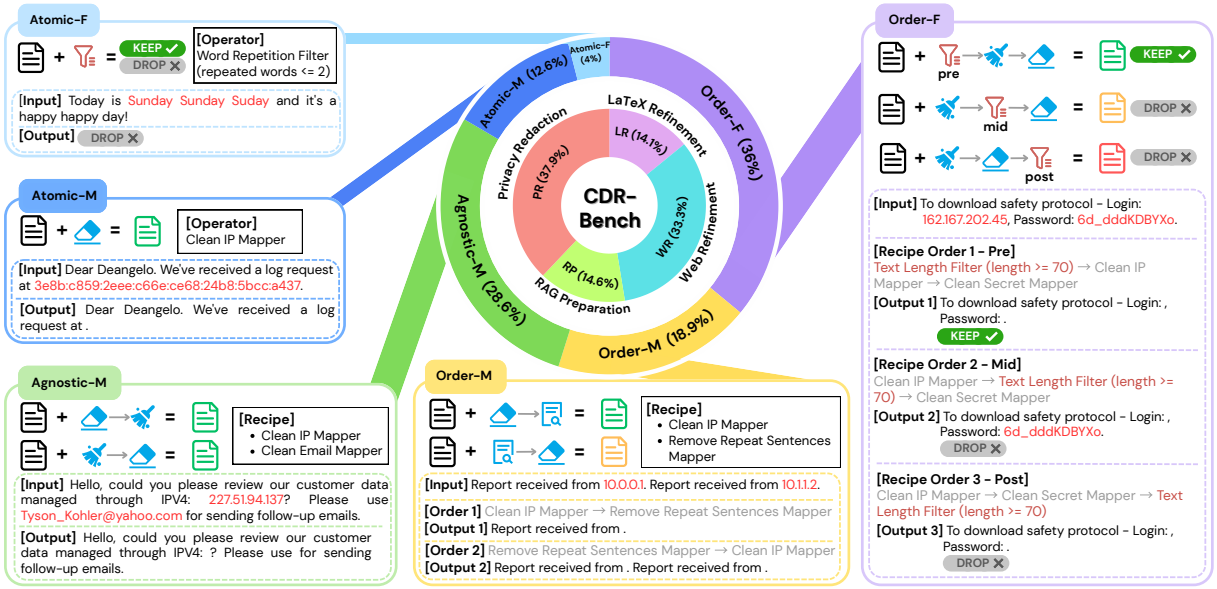

Figure 2: Overview of the CDR-Bench. The benchmark spans four data refinement domains (Privacy Redaction, Web Refinement, LaTeX Refinement, and RAG Preparation). The benchmark evaluates three levels of recipe execution: (i) single operators (**Atomic-M/F**); (ii) order-agnostic recipes (**Agnostic-M**); and (iii) order-sensitive recipes: mapper permutations (**Order-M**) and filter placements across Pre/Mid/Post positions (**Order-F**). Here, Mappers (M) and Filters (F) denote transformation and filtering operators, respectively.

2.1 Task Formulation

Compositional Data Refinement Our benchmark evaluates whether large language models can execute *compositional data-refinement recipes* described in natural language. A refinement recipe $r = (o_1, o_2, \ldots, o_n)$ is an ordered sequence of data-processing operators[2] applied to an input text t_0.

Formally, the deterministic reference execution of a recipe r on an input t_0 is denoted as $E(t_0, r) = (s^\star, t^\star)$. Here, $s^\star \in \{\texttt{KEEP}, \texttt{DROP}\}$ represents the ground-truth execution status, and $t^\star$ denotes the reference edited text. Specifically, $t^\star$ is the final text after executing all operators if $s^\star = \texttt{KEEP}$; or the last text state immediately before the first rejecting filter is applied if $s^\star = \texttt{DROP}$. Correspondingly, given the input text t_0 and a natural-language instruction $q(r)$ describing the recipe r, the execution by a language model M is defined as $M(t_0, q(r)) = (\hat{s}, \hat{t})$. The model predicts its own execution status $\hat{s}$ and the resulting text $\hat{t}$. The core task is to evaluate whether the LLM's execution $M(t_0, q(r))$ faithfully aligns with the deterministic reference $E(t_0, r)$.

[2]Each o_i denotes a data-processing operator. Following the taxonomy in Data-Juicer (Chen et al., 2024a, 2025), we consider two operator types: *mappers* (M), which rewrite text according to predefined rules, and *filters* (F), which return binary KEEP/DROP decisions.

Order-Sensitive Recipes Beyond basic composition, we further evaluate whether models correctly handle execution-order dependencies. We define a recipe group as *order-sensitive* if multiple recipes share the same operator set but produce different execution outcomes under different operator ordering. Formally, for a given input text t_0, two recipes r_i and r_j formed by different permutations of the same operators are order-sensitive if their reference executions diverge, i.e., $E(t_0, r_i) \neq E(t_0, r_j)$. For instance, a *text-length filter* placed before redaction mappers may yield KEEP, while the same filter placed after yields DROP as the cleaned text is shorter.

2.2 Benchmark Construction

Overview. Figure 2 shows an overview of the CDR-Bench dataset. CDR-Bench is designed to evaluate whether LLMs can faithfully execute compositional data-refinement recipes grounded in real data-processing needs. The benchmark construction pipeline proceeds in four stages: (1) collecting heterogeneous corpora and activating mapper operators on individual records; (2) mining frequent operator co-occurrence patterns to identify recipe candidates; (3) materializing tracks with deterministic references; and (4) verbalizing each recipe into diverse natural-language instructions.

Data Collection and Operator Activation We source data from Common Crawl 2026 snapshot, arXiv preprints (Wang et al., 2024), Wikipedia, GovReport (Huang et al., 2021), and multiple PII datasets (Nutrient.io, 2025; Ai4Privacy Community, 2024; Steier et al., 2025), organized into four functional domains: Web Refinement (WR), LaTeX Refinement (LR), RAG Preparation (RP), and Privacy Redaction (PR). For each domain, we define a set of candidate mapper and filter operators targeting common refinement requirements (detailed in Appendix A.1). We then perform an operator-level activation analysis on raw records to identify which mappers produce non-trivial changes on each sample, yielding a domain-specific pool of records annotated with their activated mapper operators. This annotated pool serves as the empirical foundation for the subsequent recipe mining step, ensuring that recipe construction is grounded in naturally occurring operator patterns rather than arbitrary author-defined combinations.

Recipe Mining Within each annotated pool, we identify representative mapper co-occurrence patterns through a greedy coverage procedure. Candidate operator combinations are ranked by the total number of records they cover, with ties broken in favor of longer combinations. This yields a compact set of recipe family anchors that capture diverse, real-world operator compositions. We then select the most frequent exact combinations within each retained family to serve as our final recipes. Pseudocode for the mining algorithm is provided in Appendix A.2.

Track Materialization Starting from the mined mapper recipes, we materialize three recipe-level evaluation tracks. For each recipe, we pair it with raw input samples that activate all of its constituent operators, executing the pipeline to obtain the deterministic reference output $t^\star$. *Agnostic-M* directly instantiates recipes from fixed mapper combinations to evaluate baseline sequential execution. *Order-M* constructs order-sensitive pairs by swapping operator positions, retaining only instances where the two orderings yield divergent outcomes on the same initial text t_0. *Order-F* dynamically inspects intermediate text states to instantiate valid filters with calibrated thresholds. It then inserts each filter at three distinct positions (*pre*, *mid*, and *post*) within a fixed mapper sequence, retaining only those groups where at least two placements produce different final outcomes (see Appendix A.3 for details). Additionally, *Atomic-M* and *Atomic-F* are included to evaluate single-operator tasks, serving as an isolated capability baseline.

Instruction Verbalization Once all tracks are materialized, we first build a prompt library by verbalizing each recipe into natural-language instructions across 11 prompt styles (Table 6), keeping every instruction aligned with the operator sequence, filter semantics, and threshold values defined during materialization. An LLM-based judge screens candidate instructions for functional equivalence to the target recipe, correct preservation of execution order, and natural expression of numeric constraints without exposing code-level identifiers (Figure 7). During evaluation, each instance is paired with three prompt variants sampled deterministically via a fixed random seed from distinct styles in the library, and wrapped in a fixed response-format template, separating phrasing variation from the standardized output contract.

2.3 Evaluation Metrics

We evaluate $M(t_0, q(r)) = (\hat{s}, \hat{t})$ against the deterministic reference $E(t_0, r) = (s^\star, t^\star)$. Standard metrics such as exact match (Rajpurkar et al., 2016), edit distance (Dwivedi-Yu et al., 2022; Zhang et al., 2024c; Levenshtein et al., 1966), and SARI (Xu et al., 2016) are insufficient here: exact match penalizes benign formatting variation, edit distance does not distinguish progress from regression, and SARI targets n-gram simplification rather than deterministic operator logic. We therefore propose three task-specific metrics: Recipe Success (RS), Order-Consistent Success (OCS), and Refinement Gain (RG).

Recipe Success (RS) measures exact recipe execution under a normalized text comparison. Let $f(\cdot)$ be a text normalization function that strips leading and trailing whitespace, removes empty lines, and unifies newline characters. We define

$$\mathrm{RS} = \mathbf{1}\big[\hat{s} = s^\star \wedge f(\hat{t}) = f(t^\star)\big]. \qquad (1)$$

A prediction is successful only if it correctly predicts the execution status and perfectly reconstructs the normalized reference text. We report RS@K throughout, where an instance is counted as solved if the model succeeds under at least one of K prompt styles.

Order-Consistent Success (OCS) is used to evaluate order-sensitive recipe groups. Each group $\mathcal{V}$ contains recipe variants that share the identical operator set but differ in execution sequence (e.g., permuted mapper orders or varying filter placements). OCS requires the model to correctly solve every variant within the group simultaneously:

$$\mathrm{OCS}(\mathcal{V}) = \mathbf{1}[\forall\, v \in \mathcal{V} :\ \mathrm{RS}_v = 1]\,, \qquad (2)$$

where RS_v denotes the Recipe Success for a specific variant v. For mapper-filter order groups, this necessitates succeeding on the *pre*, *mid*, and *post* variants simultaneously. We also report OCS@K, where a group is considered solved if every variant succeeds under at least one of its K prompt styles.

Refinement Gain (RG) captures partial editing progress not reflected in RS or OCS, measuring normalized edit-distance improvement toward the reference. Let $d(\cdot,\cdot)$ denote edit distance. We define

$$\mathrm{RG} = \max\left\{0,\ 1 - \frac{d(\hat{t}, t^\star)}{d(t_0, t^\star) + \epsilon}\right\}, \qquad (3)$$

where $\epsilon > 0$ is a small constant to prevent division by zero when the initial input already perfectly matches the reference. RG equals 1 when the prediction matches the reference, and 0 when it fails to reduce the edit distance relative to the original input.

2.4 Benchmark Statistics

CDR-Bench contains 3,145 high-quality compositional tasks and 174 atomic tasks spanning four data refinement domains. Detailed statistics are provided in Appendix A.5.

3 Experimental Results

3.1 Experiment Setup

Evaluated Models We evaluate more than ten state-of-the-art LLMs, spanning open-source models such as Qwen3.6 (Qwen Team, 2026a,b), DeepSeek-V4 (DeepSeek-AI, 2026), GLM (GLM-5-Team et al., 2026), Gemma-4 (Google DeepMind, 2026), Llama-4 (Meta AI, 2025), and Kimi-K2.6 (Moonshot AI, 2026), as well as proprietary model families such as Claude (Anthropic, 2025a,b) and GPT (OpenAI, 2026). Since data refinement is often applied at corpus scale, where low-latency and cost-efficient inference is preferred, we disable reasoning modes whenever explicit controls are available. Additional model setting details are provided in Appendix B.1.

Prompting Strategies We evaluate four prompting strategies. *Direct Mode* serves as the default, asking the model to produce the final output directly. *Few-Shot Mode* prepends two solved examples from the same track. *Plan-First Mode* asks the model to first restate the recipe as an ordered execution plan before producing the output. *State-Aware Mode* further encourages the model to identify intermediate text states and specify which operation or filter applies to each. The latter three strategies test whether demonstrations, explicit planning, or intermediate-state reasoning can mitigate recipe-execution failures. Prompt templates are provided in Appendix A.6.

3.2 Main Results

Performance across Different LLMs Table 2 reports the main results on CDR-Bench. Although proprietary models achieve stronger absolute performance, both open- and closed-source LLMs exhibit similar degradation patterns. We highlight three key findings:

(1) Atomic-to-recipe degradation. Models become substantially less reliable when refinement operations are composed into multi-step recipes rather than executed in isolation. In order-agnostic settings, GPT-5.4 drops from 30.30% RS@3 on Atomic-M to 22.74% on Agnostic-M, while Gemma-4-31B-IT shows a similar 4 percentage point(pp) decline. The degradation becomes even more severe in order-sensitive settings, where GPT-5.4 further falls to 13.99% on Order-M and Gemma-4-31B-IT drops from 25.55% to 8.04%. Similar trends hold across nearly all evaluated models, suggesting that procedural composition itself emerges as a major bottleneck for current LLMs.

(2) Order sensitivity exposes a gap between plausible refinement and faithful execution. Reordering transformations collapses exact recipe success even when the overall refinement gain remains stable. DeepSeek-V4-Flash drops 19.0 pp from Agnostic-M to Order-M in RS@3 while RG changes much less, suggesting models can produce directionally correct refinements yet fail to follow the intended execution order. This pattern further manifests within Order-M as a consistent gap between recipe success and group-level order consistency, where GPT-5.4 reaches 13.99% RS@3 but only 4.90% OCS@3. Models can occasionally produce correct individual recipes yet consistently fail to maintain order across a full operator group.

(3) Deferred decisions become increasingly un-

Model	Atomic-M	Atomic-F	Agnostic-M		Order-M			Order-F				
	RS@3↑	RS@3↑	RS@3↑	RG↑	RS@3↑	RG↑	OCS@3↑	RS_{pre}@3↑	RS_{mid}@3↑	RS_{post}@3↑	RG↑	OCS@3↑
Closed-Source Models												
Claude Opus 4.5	27.27	71.43	22.13	49.39	11.54	51.00	2.80	42.51	18.56	13.05	41.17	8.14
Claude Opus 4.6	29.55	73.81	22.33	48.06	12.24	50.42	2.80	57.96	**33.17**	**23.95**	47.03	**18.68**
Qwen3.6-max	26.15	76.19	23.94	50.81	7.34	48.90	0.70	46.88	21.82	17.03	39.44	8.27
GPT-5.4	**30.30**	**88.10**	22.74	53.77	**13.99**	52.58	**4.90**	**62.04**	25.03	14.97	**48.47**	9.46
+ Few-Shot	$31.82_{+1.52}$	$88.10_{+0.00}$	$23.38_{+0.64}$	$52.98_{-0.79}$	$13.99_{+0.00}$	$52.46_{-0.12}$	$4.90_{+0.00}$	$63.37_{+1.33}$	$25.72_{+0.69}$	$15.18_{+0.21}$	$48.46_{-0.01}$	$9.99_{+0.53}$
+ Plan-First	–	–	$23.74_{+1.00}$	$54.45_{+0.68}$	$18.18_{+4.19}$	$55.37_{+2.79}$	$7.69_{+2.79}$	$62.42_{+0.38}$	$29.38_{+4.35}$	$17.01_{+2.04}$	$53.43_{+4.96}$	$13.06_{+3.60}$
+ State-Aware	–	–	$23.84_{+1.10}$	$54.17_{+0.40}$	$16.43_{+2.44}$	$53.24_{+0.66}$	$7.69_{+2.79}$	$75.15_{+13.11}$	$29.52_{+4.49}$	$17.07_{+2.10}$	$51.18_{+2.71}$	$13.45_{+3.99}$
Open-Source Models												
DeepSeek-V4-Flash	24.62	76.19	23.94	44.41	4.90	41.27	0.00	39.57	11.75	13.67	33.43	2.40
Kimi-K2.6	29.23	76.19	24.14	46.36	9.44	47.77	0.70	35.25	16.55	14.39	35.39	5.88
Llama-4-Scout-17B	16.92	62.50	13.59	35.95	3.87	26.66	0.00	22.41	4.47	3.99	26.98	0.86
GLM-5	26.15	73.81	24.80	49.42	10.49	52.05	2.10	33.93	22.78	17.75	37.92	6.47
Gemma-4-31B-IT	29.55	80.95	**25.55**	**56.54**	8.04	**59.32**	0.70	32.81	19.16	14.01	43.19	6.11
Qwen3.6-27B	24.24	80.95	23.73	42.39	8.42	41.88	2.11	46.95	19.09	17.51	33.29	7.09
Qwen3.6-35B-A3B	18.18	78.05	20.40	42.03	7.02	41.31	0.00	50.98	12.17	12.89	29.95	3.70
+ Few-Shot	$24.24_{+6.06}$	$83.33_{+5.28}$	$19.19_{-1.21}$	$39.99_{-2.04}$	$6.32_{-0.70}$	$40.41_{-0.90}$	$0.00_{+0.00}$	$52.20_{+1.22}$	$12.11_{-0.06}$	$11.42_{-1.47}$	$30.41_{+0.46}$	$2.40_{-1.30}$
+ Plan-First	–	–	$20.52_{+0.12}$	$42.25_{+0.22}$	$5.94_{-1.08}$	$40.03_{-1.28}$	$0.00_{+0.00}$	$41.16_{-9.82}$	$10.84_{-1.33}$	$12.98_{+0.09}$	$31.39_{+1.44}$	$3.55_{-0.15}$
+ State-Aware	–	–	$23.94_{+3.54}$	$47.70_{+5.67}$	$9.44_{+2.42}$	$48.52_{+7.21}$	$2.10_{+2.10}$	$59.98_{+9.00}$	$25.18_{+13.01}$	$19.35_{+6.46}$	$38.61_{+8.66}$	$12.33_{+8.63}$

Table 2: Main results on CDR-Bench under models' non-thinking mode, evaluating baseline performances and the effects of prompt-level enhancement strategies. We report recipe success (RS@3) and refinement gain (RG) across tasks, with group-level order consistency (OCS@3) additionally reported for order-sensitive recipes. Note that Plan-First and State-Aware strategies target recipe-level execution rather than single atomic tasks (denoted by "–"). Best and second-best results among the original baseline models are **bolded** and underlined.

reliable. Filtering decisions degrade substantially when applied after preceding transformations. GPT-5.4 declines from 62.04% RS@3 in pre-filter settings to 25.03% and 14.97% in mid- and post-filter settings, respectively. Qwen3.6-35B-A3B exhibits a similar pattern, indicating that downstream decisions become unreliable once they depend on transformed intermediate states.

Performance over Different Recipe Length Figure 3 shows that performance consistently decreases as recipes grow longer across all three compositional tracks. On Agnostic-M and Order-M, all three models nearly collapse to zero RS@3 beyond five operations. Order-M degrades more rapidly than Agnostic-M, indicating that tracking evolving intermediate states under operator reordering compounds the difficulty introduced by length alone. Order-F shows a more gradual decline, as its transformation order is fixed and only the filter placement varies, requiring less fine-grained state discrimination than Order-M.

Performance across Domains Figure 4 shows performance and compositional degradation across domains. PR achieves the highest atomic RS@3, as its operators target localized patterns such as emails and URLs. WR and LR score lower atomically, since their operators require reasoning over broader document structure. Despite this, PR suffers the steepest compositional drops (-29.1 pp for GPT-5.4, -38.1 pp for Gemma-4), suggesting that atomic familiarity does not confer compositional robustness and the bottleneck shifts to multi-step state tracking regardless of operator simplicity.

Effect of Different Prompting Strategies Table 2 compares prompt-level strategies against Direct Mode. *Few-Shot* improves atomic execution by 5–6 pp for Qwen3.6-35B-A3B but offers little benefit at the recipe level. *Plan-First* brings moderate gains on order-sensitive settings for GPT-5.4 but remains inconsistent across models. *State-Aware* is the most effective strategy: by explicitly identifying intermediate text states and specifying which operation or filter applies to each, it consistently improves order-sensitive performance, increasing Order-F OCS@3 by 4 pp for GPT-5.4 and 9 pp for Qwen3.6-35B-A3B, while also benefiting Agnostic-M, suggesting that explicit state tracking helps beyond strictly order-sensitive settings. Nevertheless, absolute OCS@3 scores remain low across all strategies, indicating that prompt engineering alone is insufficient to fully resolve recipe execution failures.

Effect of Instruction Styles Figure 12 (left) shows that RS@K increases steadily with the number of styles attempted, as different phrasings provide complementary benefits, though the Atomic–Compositional gap persists across all K. The right panel shows that operation-enumerating styles such as *Step-by-Step* and *Brief* outperform outcome-oriented styles such as *Goal-First* and *Scenario Story* on transformation-heavy tracks, as the latter require models to infer intermediate procedures

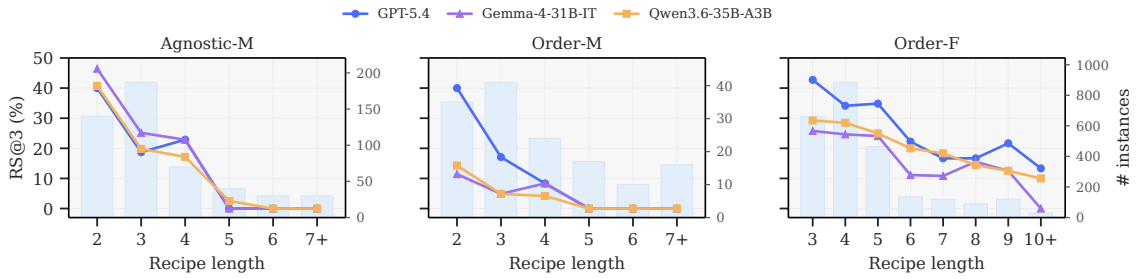

Figure 3: Performance over different recipe lengths on three compositional recipe tracks. Lines show RS@3 of representative LLMs, and light-blue bars indicate the number of instances in each recipe-length bucket.

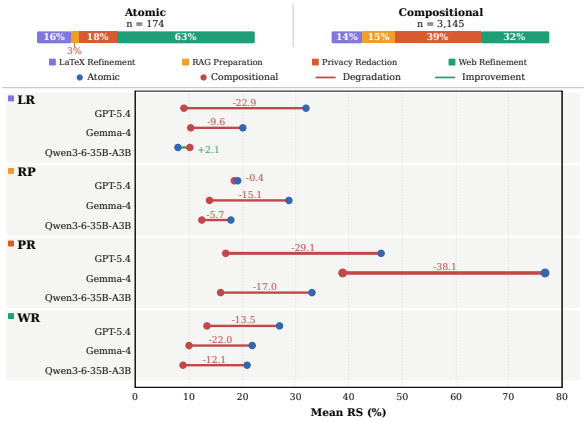

Figure 4: Atomic and compositional RS across domains, with per-model degradation gaps (pp).

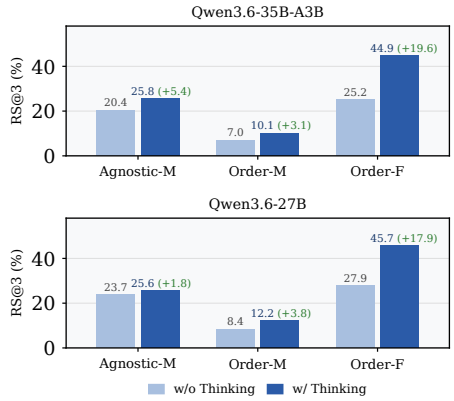

Figure 5: Effect of thinking mode across tracks.

Model	n	Closer to canonical	Closer to swapped	Tie
Gemma-4	409	68.0%	29.8%	2.2%
GPT-5.4	383	55.4%	43.1%	1.6%
Qwen3.6-35B-A3B	416	54.6%	43.0%	2.4%
Overall	1208	**59.4%**	**38.6%**	**2.1%**

Table 3: Analysis of ORDER-M swapped-condition errors. Most failures are closer to the canonical ordering than to the perturbed gold, suggesting systematic resistance to instruction rather than random output.

from the desired end state. On Order-M, all styles collapse into a narrow performance range, confirming that instruction phrasing alone cannot compensate for the fine-grained state tracking required for order-sensitive execution.

Effect of Thinking Mode Figure 5 shows that thinking mode substantially improves RS@3 across all compositional tracks, with the largest gains on Order-F (+19.6 pp for Qwen3.6-35B-A3B and +17.9 pp for Qwen3.6-27B), where explicit multi-step reasoning most directly benefits filter evaluation over transformed text states. Gains on Order-M are comparatively modest, as mapper ordering errors are more implicit and harder to surface even with extended reasoning.

3.3 Error Analysis

Figure 13 summarizes the failure distribution and full definitions and breakdowns are in Appendix B.3. Three patterns stand out. *(1) Filter-threshold errors* account for 39.3% of failures, where models produce plausible cleaned text but assign the wrong KEEP/DROP status. *(2) Deterministic execution remains brittle*: mapper tracks are dominated by formatting drift (28.4%), missed operators (11.7%), and under-application (9.5%), indicating that models approximate rather than faithfully reproduce operator behavior. *(3) Rejection does not reliably halt execution*: in Order-F DROP cases, models sometimes continue past a rejecting filter. As shown in Fig. 6, RS in Pre-DROP cases closely tracks preference for t_{stop} over t_{full}, while Mid-DROP cases further require correct intermediate-state rewriting before termination.

3.4 Real-World Scenario Evaluation

To verify that the observed compositional difficulties are not confined to rule-based operators, we further evaluate models on two semantic tracks with human-annotated references (PII redaction and hallucination processing), summarized in Ta-

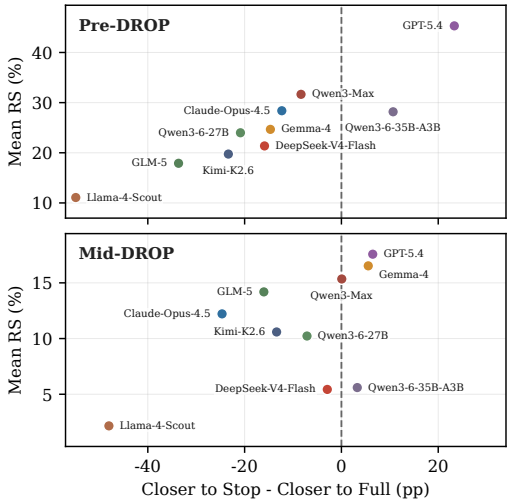

Figure 6: Failure to stop after a rejecting filter in Order-F DROP cases. Predictions are compared with the correct stopping state t_{stop} and a continued-execution state t_{full}. The x-axis shows the closer-rate difference between t_{stop} and t_{full}, and the y-axis reports mean RS.

ble 4. PII redaction confirms that the composition bottleneck generalizes to semantic settings, as all models drop substantially when multiple redaction requirements must be satisfied simultaneously. Hallucination processing reveals a different structure where difficulty is dominated by precise correction rather than composition, and models consistently perform better on hallucination-free texts than on those requiring targeted edits. These contrasting patterns suggest that compositional challenge is task-structure-dependent rather than a universal constant. Full details are in Appendix C.

	PII Redaction		Hallucination	
Model	Atomic	Comp	Atomic	Comp
Claude Opus 4.5	66.9	**45.0**	19.1	22.2
Claude Opus 4.6	66.7	44.4	16.7	20.0
Qwen3.6-Max	**67.4**	43.0	**21.7**	21.4
GPT-5.4	57.5	33.2	15.5	15.0
DeepSeek-V4-Flash	55.6	38.0	21.1	20.0
Kimi-K2.6	58.8	28.0	19.7	**31.1**
GLM-5	66.0	41.0	15.3	14.3

Table 4: Real-scenario evaluation (RS %). Atomic: mean across atomic subtasks; Comp: compositional.

4 Related Work

Data Curation and Data-Centric Agent Benchmarks Recent work has introduced benchmarks for LLM-based data curation and data-centric agents. Some evaluate workflow construction or executable data processing over structured data, including AutoDCWorkflow and DataGovBench (Li et al., 2025; Liu et al., 2025), while broader benchmarks such as DAComp and KramaBench cover data-intelligence workflows involving SQL/Python coding, multi-source integration, data cleaning, reasoning, and report generation (Lei et al., 2025; Lai et al., 2026). Other benchmarks target content-level curation tasks, including PII detection and anonymization (Ai4Privacy, 2026; Pilán et al., 2022), hallucination detection or correction (Niu et al., 2024; Dziri et al., 2022), and semantic tagging of entities and relations (Wang et al., 2022).

LLM-Driven Data Curation and Preparation Recent studies have explored LLMs for data curation, cleaning, and preparation. Prior work evaluates or instruction-tunes LLMs for preprocessing operators including error detection, data imputation, schema matching, and entity matching (Zhang et al., 2024b,a). Other methods address retrieval-based repair, dependency induction for tabular cleaning, and formula synthesis for imputation (Naeem et al., 2024; Biester et al., 2024; Zhang et al., 2024d). Systems including ChatPipe, SEED, and AutoPrep study pipeline orchestration, domain-specific curation, multi-agent table preparation, and automatic cleaning workflow generation (Chen et al., 2023, 2024b; Fan et al., 2025).

5 Conclusion

In this paper, we introduced CDR-Bench to evaluate the faithful execution of compositional, order-sensitive data refinement recipes. CDR-Bench reveals a consistent gap between what LLMs appear to do and what they are actually instructed to do. Across all evaluated models, composing operators into multi-step recipes exposes failure patterns that single-operator performance does not predict: models frequently produce plausible-looking outputs while silently violating execution order, and fewer than 5% of order-sensitive recipe groups are solved correctly across all tested orderings. Real-world semantic tracks further confirm that compositional difficulty is not an artifact of rule-based operator design, though its magnitude varies with task structure. Taken together, the results point to procedural faithfulness as a distinct capability dimension that is poorly captured by existing benchmarks, and one that will need to be explicitly targeted as LLMs take on greater roles in data preparation pipelines.

Limitations

CDR-Bench has several limitations. First, it is grounded in the semantics and coverage of the Data-Juicer operator library. To keep evaluation deterministic, we exclude subjective refinement recipes and do not use LLM judges as final evaluators, which limits coverage of broader real-world refinement practices. Second, prompt verbalization partly relies on LLM assistance. Although we average over multiple prompt styles and evaluate several prompting strategies, this process may introduce two gaps: the generated instructions may differ from how users naturally describe refinement needs, and they may imperfectly verbalize the precise execution semantics of the underlying deterministic recipes, despite validation. Third, CDR-Bench focuses on direct text-level execution and does not cover tool use, code generation, or interactive refinement in agentic settings. Finally, multilingual and multimodal refinement remain outside the current scope. We leave these directions for future work.

Ethics Statement

CDR-Bench includes tasks involving privacy redaction and sensitive text cleanup. To mitigate potential risks, we rely on public, licensed, or synthetic data sources where appropriate. The released benchmark will avoid exposing raw private credentials, personal identifiers, or other unsafe content beyond what is already controlled in the source data. The goal of CDR-Bench is to improve the reliability of safe data processing and privacy-preserving refinement, rather than to facilitate misuse.

Reproducibility Statement

To support reproducibility, we will release the data mining code, selected benchmark instances, recipe metadata, prompt templates across different verbalization styles, and evaluation code. Since gold references are derived from deterministic operator execution, future work can reproduce the evaluation without relying on subjective model-based judging.

References

Ai4Privacy. 2026. Openpii 1m — multilingual pii masking dataset (19 labels, 23 languages).

Ai4Privacy Community. 2023. ai4privacy/pii-masking-400k. Hugging Face Dataset. A synthetic multilingual dataset for training and evaluating PII detection and masking models with annotated span.

Ai4Privacy Community. 2024. ai4privacy/pii-masking-200k dataset. Hugging Face Dataset. A synthetic multilingual dataset for training and evaluating PII detection and masking models with annotated span.

Anthropic. 2025a. Introducing claude opus 4.5. https://www.anthropic.com/news/claude-opus-4-5. Accessed: 2025-11-24.

Anthropic. 2025b. Introducing claude opus 4.6. https://www.anthropic.com/news/claude-opus-4-6. Accessed: 2026-02-05.

Fabian Biester, Mohamed Abdelaal, and Daniel Del Gaudio. 2024. Llmclean: Context-aware tabular data cleaning via llm-generated ofds. *Preprint*, arXiv:2404.18681.

Daoyuan Chen, Yilun Huang, Zhijian Ma, Hesen Chen, Xuchen Pan, Ce Ge, Dawei Gao, Yuexiang Xie, Zhaoyang Liu, Jinyang Gao, Yaliang Li, Bolin Ding, and Jingren Zhou. 2024a. Data-juicer: A one-stop data processing system for large language models. In *SIGMOD*.

Daoyuan Chen, Yilun Huang, Xuchen Pan, Nana Jiang, Haibin Wang, Yilei Zhang, Ce Ge, Yushuo Chen, Wenhao Zhang, Zhijian Ma, Jun Huang, Wei Lin, Yaliang Li, Bolin Ding, and Jingren Zhou. 2025. Data-juicer 2.0: Cloud-scale adaptive data processing for and with foundation models. *NeurIPS*.

Sibei Chen, Hanbing Liu, Weiting Jin, Xiangyu Sun, Xiaoyao Feng, Ju Fan, Xiaoyong Du, and Nan Tang. 2023. Chatpipe: Orchestrating data preparation program by optimizing human-chatgpt interactions. *Preprint*, arXiv:2304.03540.

Zui Chen, Lei Cao, Sam Madden, Tim Kraska, Zeyuan Shang, Ju Fan, Nan Tang, Zihui Gu, Chunwei Liu, and Michael Cafarella. 2024b. Seed: Domain-specific data curation with large language models. *Preprint*, arXiv:2310.00749.

DeepSeek-AI. 2026. Deepseek-v4: Towards highly efficient million-token context intelligence.

Jane Dwivedi-Yu, Timo Schick, Zhengbao Jiang, Maria Lomeli, Patrick Lewis, Gautier Izacard, Edouard Grave, Sebastian Riedel, and Fabio Petroni. 2022. Editeval: An instruction-based benchmark for text improvements. *Preprint*, arXiv:2209.13331.

Nouha Dziri, Ehsan Kamalloo, Sivan Milton, Osmar Zaiane, Mo Yu, Edoardo M. Ponti, and Siva Reddy. 2022. FaithDial: A faithful benchmark for information-seeking dialogue. *Transactions of the Association for Computational Linguistics*, 10:1473–1490.

Meihao Fan, Ju Fan, Nan Tang, Lei Cao, Guoliang Li, and Xiaoyong Du. 2025. Autoprep: Natural language question-aware data preparation with a multi-agent framework. *Preprint*, arXiv:2412.10422.

Leon Garza, Anantaa Kotal, Aritran Piplai, Lavanya Elluri, Prajit Das, and Aman Chadha. 2025. Prvl: Quantifying the capabilities and risks of large language models for pii redaction. *Preprint*, arXiv:2508.05545.

GLM-5-Team, :, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, Chenzheng Zhu, Congfeng Yin, Cunxiang Wang, Gengzheng Pan, Hao Zeng, Haoke Zhang, Haoran Wang, and 168 others. 2026. Glm-5: from vibe coding to agentic engineering. *Preprint*, arXiv:2602.15763.

Google DeepMind. 2026. Our most intelligent open models, built from gemini 3 research and technology to maximize intelligence-per-parameter. https://deepmind.google/models/gemma/gemma-4/. Accessed: 2026-05-18.

Benhao Huang, Yingzhuo Yu, Jin Huang, Xingjian Zhang, and Jiaqi Ma. 2025. Dca-bench: A benchmark for dataset curation agents. *Preprint*, arXiv:2406.07275.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. 2021. Efficient attentions for long document summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1419–1436, Online. Association for Computational Linguistics.

Ayman Asad Khan, Md Toufique Hasan, Kai Kristian Kemell, Jussi Rasku, and Pekka Abrahamsson. 2024. Developing retrieval augmented generation (rag) based llm systems from pdfs: An experience report. *Preprint*, arXiv:2410.15944.

Eugenie Lai, Gerardo Vitagliano, Ziyu Zhang, Om Chabra, Sivaprasad Sudhir, Anna Zeng, Anton A. Zabreyko, Chenning Li, Ferdi Kossmann, Jialin Ding, Jun Chen, Markos Markakis, Matthew Russo, Weiyang Wang, Ziniu Wu, Michael J. Cafarella, Lei Cao, Samuel Madden, and Tim Kraska. 2026. Kramabench: A benchmark for ai systems on data-to-insight pipelines over data lakes. *Preprint*, arXiv:2506.06541.

Ga Young Lee, Lubna Alzamil, Bakhtiyar Doskenov, and Arash Termehchy. 2021. A survey on data cleaning methods for improved machine learning model performance. *Preprint*, arXiv:2109.07127.

Fangyu Lei, Jinxiang Meng, Yiming Huang, Junjie Zhao, Yitong Zhang, Jianwen Luo, Xin Zou, Ruiyi Yang, Wenbo Shi, Yan Gao, Shizhu He, Zuo Wang, Qian Liu, Yang Wang, Ke Wang, Jun Zhao, and Kang Liu. 2025. Dacomp: Benchmarking data agents across the full data intelligence lifecycle. *Preprint*, arXiv:2512.04324.

Vladimir I Levenshtein and 1 others. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710. Soviet Union.

Canchen Li. 2019. Preprocessing methods and pipelines of data mining: An overview. *Preprint*, arXiv:1906.08510.

Lan Li, Liri Fang, Bertram Ludäscher, and Vetle I. Torvik. 2025. Autodcworkflow: Llm-based data cleaning workflow auto-generation and benchmark. *Preprint*, arXiv:2412.06724.

Jingyu Liu, Jiaen Lin, and Yong Liu. 2026. Tackling the inherent difficulty of noise filtering in rag. *Preprint*, arXiv:2601.01896.

Zhou Liu, Zhaoyang Han, Guochen Yan, Hao Liang, Bohan Zeng, Xing Chen, Yuanfeng Song, and Wentao Zhang. 2025. Datagovbench: Benchmarking llm agents for real-world data governance workflows. *Preprint*, arXiv:2512.04416.

Meta AI. 2025. The llama 4 herd: The beginning of a new era of natively multimodal intelligence. https://ai.meta.com/blog/llama-4-multimodal-intelligence/. Accessed: 2026-05-18.

Abhika Mishra, Akari Asai, Vidhisha Balachandran, Yizhong Wang, Graham Neubig, Yulia Tsvetkov, and Hannaneh Hajishirzi. 2024. Fine-grained hallucination detection and editing for language models. *Preprint*, arXiv:2401.06855.

Moonshot AI. 2026. Kimi k2.6 tech blog: Advancing open-source coding. https://www.kimi.com/blog/kimi-k2-6. Accessed: 2026-05-18.

Zan Ahmad Naeem, Mohammad Shahmeer Ahmad, Mohamed Eltabakh, Mourad Ouzzani, and Nan Tang. 2024. Retclean: Retrieval-based data cleaning using foundation models and data lakes. *Preprint*, arXiv:2303.16909.

Cheng Niu, Yuanhao Wu, Juno Zhu, Siliang Xu, KaShun Shum, Randy Zhong, Juntong Song, and Tong Zhang. 2024. RAGTruth: A hallucination corpus for developing trustworthy retrieval-augmented language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10862–10878, Bangkok, Thailand. Association for Computational Linguistics.

Nutrient.io. 2025. Docpii: Contextual redaction benchmark dataset. https://huggingface.co/datasets/nutrientdocs/synthetic_labeled_redaction_instruction_en_v1. Synthetic dataset for document-contextual PII redaction evaluation, based on gretelai/gretel-pii-masking-en-v1.

OpenAI. 2026. Gpt-5.4 model documentation. https://developers.openai.com/api/docs/models/gpt-5.4. Accessed: 2026-05-18.

Anwesan Pal, Radhika Bhargava, Kyle Hinsz, Jacques Esterhuizen, and Sudipta Bhattacharya. 2024. The empirical impact of data sanitization on language models. *Preprint*, arXiv:2411.05978.

Ildikó Pilán, Pierre Lison, Lilja Øvrelid, Anthi Papadopoulou, David Sánchez, and Montserrat Batet. 2022. The text anonymization benchmark (TAB): A dedicated corpus and evaluation framework for text anonymization. *Computational Linguistics*, 48(4):1053–1101.

Zeyu Qin, Qingxiu Dong, Xingxing Zhang, Li Dong, Xiaolong Huang, Ziyi Yang, Mahmoud Khademi, Dongdong Zhang, Hany Hassan Awadalla, Yi R. Fung, Weizhu Chen, Minhao Cheng, and Furu Wei. 2025. Scaling laws of synthetic data for language models. *Preprint*, arXiv:2503.19551.

Qwen Team. 2026a. Qwen3.6-27B: Flagship-level coding in a 27B dense model.

Qwen Team. 2026b. Qwen3.6-35B-A3B: Agentic coding power, now open to all.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *Preprint*, arXiv:1606.05250.

Amy Steier, Andre Manoel, Alexa Haushalter, and Maarten Van Segbroeck. 2025. Nemotron-pii: Synthesized data for privacy-preserving ai.

Chong Wang, Jian Zhang, Yiling Lou, Mingwei Liu, Weisong Sun, Yang Liu, and Xin Peng. 2024. Tiger: A generating-then-ranking framework for practical python type inference. *Preprint*, arXiv:2407.02095.

Xiaozhi Wang, Yulin Chen, Ning Ding, Hao Peng, Zimu Wang, Yankai Lin, Xu Han, Lei Hou, Juanzi Li, Zhiyuan Liu, Peng Li, and Jie Zhou. 2022. Maven-ere: A unified large-scale dataset for event coreference, temporal, causal, and subevent relation extraction. *Preprint*, arXiv:2211.07342.

Wei Xu, Courtney Napoles, Ellie Pavlick, Quanze Chen, and Chris Callison-Burch. 2016. Optimizing statistical machine translation for text simplification. *Transactions of the Association for Computational Linguistics*, 4:401–415.

Yiming Zeng, Wanhao Yu, Zexin Li, Tao Ren, Yu Ma, Jinghan Cao, Xiyan Chen, and Tingting Yu. 2026. Bridging the editing gap in llms: Fineedit for precise and targeted text modifications. *Preprint*, arXiv:2502.13358.

Haochen Zhang, Yuyang Dong, Chuan Xiao, and Masafumi Oyamada. 2024a. Jellyfish: Instruction-tuning local large language models for data preprocessing. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 8754–8782, Miami, Florida, USA. Association for Computational Linguistics.

Haochen Zhang, Yuyang Dong, Chuan Xiao, and Masafumi Oyamada. 2024b. Large language models as data preprocessors. *Preprint*, arXiv:2308.16361.

Haopeng Zhang, Hayate Iso, Sairam Gurajada, and Nikita Bhutani. 2024c. Xatu: A fine-grained instruction-based benchmark for explainable text updates. *Preprint*, arXiv:2309.11063.

Yunfan Zhang, Changlun Li, Yuyu Luo, and Nan Tang. 2024d. Sketchfill: Sketch-guided code generation for imputing derived missing values. *Preprint*, arXiv:2412.19113.

A Benchmark Construction Details

A.1 Domain-Specific Operator Inventory

Table 5 summarizes the rule-based operator inventory used in CDR-Bench. These operators include shared statistical filters and domain-specific mappers for Web Refinement (WR), LaTeX Refinement (LR), RAG Preparation (RP), and Privacy Redaction (PR). Shared filters implement cross-domain keep/drop decisions based on deterministic heuristic signals, such as text length, line length, and repetition ratios. Domain-specific mappers cover common refinement operations for crawled web text, scientific LaTeX sources, retrieval preparation, and privacy redaction.

A.2 Recipe Mining

We first extract the exact mapper signature from each record and count the frequency of all observed signatures. Candidate recipe families are constructed by enumerating mapper subsets within a predefined size range and computing their coverage over exact signatures. Specifically, the coverage of a subset is defined as the total frequency of signatures containing that subset. We retain only candidates whose coverage exceeds a minimum support threshold and rank them by coverage, subset size, and subset frequency. Family anchors are then selected greedily to maximize coverage diversity, where a candidate is kept only if it covers previously uncovered signatures. Finally, each exact signature is assigned to its best matching family anchor, and the most frequent assigned signatures are retained as recipe sources.

A.3 Filter Calibration and Insertion

For each mapper recipe, we replay execution and record intermediate text states $S_0, \ldots, S_n$, where S_0 denotes the raw input and S_k denotes the state after the k-th mapper. Each domain filter is evaluated at every checkpoint, and an initial threshold

Operator	Behavior
Shared Operators	
alphanumeric_filter	Keeps text only when its alphanumeric-content ratio falls within a configured range.
average_line_length_filter	Keeps text only when the average line length falls within a configured range.
character_repetition_filter	Keeps text only when character-level repetition remains within a configured range.
maximum_line_length_filter	Keeps text only when the longest line falls within a configured range.
text_length_filter	Keeps text only when total text length falls within a configured range.
word_repetition_filter	Keeps text only when word-level repetition remains within a configured range.
words_num_filter	Keeps text only when the total word count falls within a configured range.
Web Refinement (WR)	
clean_copyright_mapper	Removes copyright notices or related boilerplate from the beginning of a web text sample.
clean_html_mapper	Cleans HTML markup and converts HTML content into readable plain text.
clean_links_mapper	Removes HTTP, HTTPS, or FTP links from a crawled web page.
extract_tables_from_html_mapper	Extracts table content from HTML into structured text.
fix_unicode_mapper	Repairs malformed Unicode text, e.g., fixes mojibake or broken encoding.
punctuation_normalization_mapper	Normalizes non-standard Unicode punctuation into standard English forms.
remove_specific_chars_mapper	Removes configured noisy symbols or visually irrelevant special characters.
remove_words_with_incorrect_substrings_mapper	Drops tokens containing URL-like substrings such as `http`, `www`, or `.com`.
whitespace_normalization_mapper	Normalizes irregular tabs, repeated spaces, or broken spacing patterns.
LaTeX Refinement (LR)	
clean_copyright_mapper	Removes copyright-related boilerplate from LaTeX source text.
expand_macro_mapper	Expands macro definitions used in LaTeX source into their literal expansions.
fix_unicode_mapper	Repairs malformed Unicode text in extracted LaTeX content.
latex_figure_context_extractor_mapper	Extracts figure-related text, captions, and nearby citing context from LaTeX source.
punctuation_normalization_mapper	Normalizes irregular Unicode punctuation into common English punctuation.
remove_bibliography_mapper	Removes bibliography sections at the end of LaTeX documents.
remove_comments_mapper	Removes comment lines or inline comments beginning with `%`.
remove_header_mapper	Removes LaTeX preamble or header material before the main document body.
RAG Preparation (RP)	
fix_unicode_mapper	Repairs corrupted characters that reduce document readability and retrievability.
remove_long_words_mapper	Removes abnormally long tokens that are unlikely to be useful for indexing.
remove_repeat_sentences_mapper	Removes repeated sentences while preserving the remaining text order.
Privacy Redaction (PR)	
clean_email_mapper	Removes email addresses from the text.
clean_ip_mapper	Removes IPv4 and IPv6 addresses from logs.
clean_links_mapper	Removes web or file links that may expose sensitive destinations.
clean_mac_mapper	Removes MAC addresses from device records.
clean_path_mapper	Removes file-system paths, including Unix, Windows, and UNC paths.
clean_phone_mapper	Removes phone numbers from the content.
clean_secret_mapper	Removes API keys, tokens, passwords, and authorization credentials.
remove_words_with_incorrect_substrings_mapper	Drops words containing configured substrings like `href` or `.com`.

Table 5: Operator inventory used in CDR-Bench. Operators are grouped by domain and functional role. Shared operators are deterministic statistical filters used across domains, while the remaining operators are domain-specific mappers for Web Refinement (WR), LaTeX Refinement (LR), RAG Preparation (RP), and Privacy Redaction (PR).

Algorithm 1 Recipe Mining

Require: Records $\mathcal{D}$, threshold τ, max families B, max recipes M
Ensure: Family anchors $\mathcal{F}$, recipe sources $\mathcal{R}$

Extract exact mapper signatures S_d from each record
Build exact-signature counter C_{exact}
for $l \in [L_{\min}, L_{\max}]$ **do**
  Enumerate mapper subsets of size l
  Update subset counter C_{sub}
end for
for each subset $A \in C_{\text{sub}}$ **do**
  Compute coverage

$$\text{cov}(A) = \sum_{S \supseteq A} C_{\text{exact}}(S)$$

end for
Keep candidates $\mathcal{C} = \{A : \text{cov}(A) \geq \tau\}$
Sort $\mathcal{C}$ by coverage, subset size, and subset frequency
Initialize $\mathcal{F} \leftarrow \emptyset$
for each candidate $A \in \mathcal{C}$ **do**
  if A covers previously uncovered signatures **then**
    Add A to $\mathcal{F}$
  end if
  if $|\mathcal{F}| = B$ **then**
    break
  end if
end for
for each exact signature S **do**
  Assign S to its best matching family anchor
end for
for each family $A \in \mathcal{F}$ **do**
  Select top-M assigned signatures ranked by C_{exact}
end for
return $\mathcal{F}, \mathcal{R}$

Algorithm 2 Filter Calibration and Insertion

Require: Mapper recipe $\mathcal{M} = [m_1, \ldots, m_n]$, domain filters $\mathcal{F}$, support records $\mathcal{D}$, target drop rate τ
Ensure: Order-F instance groups

for each record $x \in \mathcal{D}$ **do**
  Replay $\mathcal{M}$ on x, recording checkpoints $S_0, S_1, \ldots, S_n$
end for
for each filter $f \in \mathcal{F}$ **do**
  for each checkpoint S_k **do**
    Collect filter statistic $v_k = \text{stat}(f, S_k)$ across all records
    Calibrate initial threshold via default percentile
  end for
  Select *front* candidate at $k = 0$, *end* candidate at $k = n$
  Select *mid* candidate at $k^* = \arg\max_{0<k<n} |\Delta \bar{v}_k|$
  if all three positions exist **then**
    Form order family $\{r_{\text{front}}, r_{\text{mid}}, r_{\text{end}}\}$
  end if
end for
for each order family **do**
  Pool statistic values V across all slots and records
  if f is **min**-type **then**
    $\theta \leftarrow \text{percentile}(V, \tau)$
  else
    $\theta \leftarrow \text{percentile}(V, 1 - \tau)$
  end if
  for each record x **do**
    $O \leftarrow [\text{execute}(x, r_s, \theta) \text{ for } s \in \{\text{front}, \text{mid}, \text{end}\}]$
    if $|\text{unique}(O)| \geq 2$ **then**
      Retain group
    end if
  end for
  if retained groups < 5 **then**
    Discard family
  end if
  Keep at most 10 retained groups per family
end for

is calibrated using the 20th percentile for min-type filters and the 80th percentile for max-type filters. Rather than using a fixed midpoint, the *mid* position is selected as the checkpoint whose filter statistic exhibits the largest deviation from the preceding checkpoint mean, i.e., $k^\star = \arg\max_{0<k<n} |\Delta \bar{v}_k|$. During benchmark instantiation, a unified threshold θ is recalibrated by pooling statistic values across all three positions and selecting the percentile corresponding to the target drop rate ($\tau = 0.5$ by default). An instance group is retained only if at least two of the three recipe variants produce different execution outcomes. Families with fewer than five retained groups are discarded.

A.4 Recipe Verbalization Styles

To evaluate whether models execute the underlying recipe rather than overfitting to a single instruction wording, we verbalize each recipe using multiple user-facing styles. Table 6 defines each style and provides representative examples. The styles vary the discourse form of the request, such as concise commands, casual requests, policy-like rules, and downstream use-case framing, while preserving the same operator sequence, execution order, and filtering semantics.

A.5 Benchmark Statistics

We provide detailed statistics of CDR-Bench below. Table 7 summarizes the overall task composition, input length distribution, recipe coverage, and instruction statistics. Table 8 breaks down task and recipe coverage by domain, using the same domain names as in the main text: Web Refinement (WR), LaTeX Refinement (LR), RAG Preparation (RP), and Privacy Redaction (PR).

A.6 Prompt Templates

We present the LLM judge prompt used for instruction validation (Figure 7) and the four evaluation prompt templates (Figures 8-11).

Style	Definition	Example
Brief	A short, compact instruction that keeps all necessary behavior while minimizing wording.	Remove private identifiers and unsafe links, normalize the remaining text, then keep only sanitized samples whose special-character share is below 15%.
Casual Chat	A casual chat-style request that sounds like a real user asking for help while still being complete enough to execute.	Hey, could you help me clean up these source files? I need the LaTeX comments and references removed, the macros expanded, and then please keep only sources with at least 300 words left.
Data-First	The raw data appears before the concrete instruction, so the task request is weighted toward the end of the prompt.	Above is the raw document. Please remove links and contact information, normalize spacing, and then keep the document only if its final text has at least 500 characters.
Exclusion-Based	A request centered on what must not remain in the output, useful for cleaning and filtering recipes.	When processing this report, make sure the final result contains no disclaimers, table residue, or lines longer than 180 characters. After that, judge whether it is suitable for retrieval.
Goal-First	A prose description that emphasizes the intended final state rather than a numbered procedure.	The goal is to make these help documents clean enough for a support index, with links removed, repeated template sentences cleaned up, spacing made consistent, and only documents with no more than 20% repeated words retained.
Peer Handoff	A natural teammate-to-teammate handoff written in practical workplace language.	Could you take these LaTeX sources, remove the comments and bibliography, expand the simple macros, and then keep only sources whose final text length is at least 1,000 characters?
Policy Rule	A formal processing requirement that reads like a data handling policy, not code.	Before release, the text must not contain emails, IP addresses, file paths, credentials, or long secret-like tokens; after sanitization, retain it only if the result still has at least 100 words.
QC Filter	A request that sounds like data quality screening, emphasizing retention criteria and rejection conditions.	Quality-check this page after cleaning away links and noisy HTML. Keep it only if the remaining text has at least 50 words and looks usable as corpus content.
Scenario Story	A scenario-style request that first explains the messy data situation and then asks for the needed processing.	I have raw crawl pages with markup, navigation links, and messy whitespace. Please turn them into clean readable text and keep only pages whose alphanumeric content makes up at least 60% of the final text.
Step-by-Step	A direct command that lists the required operations explicitly and in order.	Please clean this web page by removing the HTML, stripping links, normalizing the whitespace, and then keep it only if the cleaned text has at least 100 characters.
Use-Case Framing	A task framed around a downstream use case such as retrieval, indexing, release, compliance, or corpus construction.	For downstream retrieval, process these reports so that disclaimers, table residue, and abnormal long lines are removed, then keep only reports whose longest remaining line is no more than 200 characters.

Table 6: Prompt styles for recipe verbalization. After a data refinement recipe and its deterministic Data-Juicer reference are fixed, we instantiate the same recipe under multiple user-facing styles. These styles vary the surface form and discourse context of the request while preserving the operator sequence, execution order, and filtering semantics.

Judge Prompt for Instruction Validation

```
You are a strict benchmark prompt judge.
You will be given:
- The internal recipe definition and filter parameters.
- The operator code/doc evidence.
- One candidate user-facing prompt.
Judge whether the prompt is faithful to what the recipe actually does.
Mandatory keep conditions:
1. Functional equivalence: the prompt requests the same transformation and filtering behavior.
2. Order correctness: the requested order matches the internal recipe order.
3. No code leakage: the prompt does not mention operator names, parameter names, YAML, Python,
   hidden code, or implementation internals.
4. Threshold grounding: if the recipe has filter parameters, all active numeric thresholds
     must
   appear as natural user-facing constraints, not vague words like "long enough" or "too
     repetitive".
5. Wrapper compatibility: the user requirement can be safely combined with a fixed benchmark
   wrapper that separately supplies raw input text and the JSON output contract.
Also score:
- user_naturalness: does it sound like a plausible user request?
- threshold_grounding: are thresholds/conditions expressed naturally and correctly?
- clarity: is the request clear and executable?
- style_distinctiveness: does this candidate sound distinct from a generic template for the
     same style?
Return JSON only:
{
  "verdict": "keep" or "reject",
  "must_pass": {
    "functional_equivalence": true,
    "order_correct": true,
    "no_code_leakage": true,
    "thresholds_grounded": true,
    "wrapper_compatible": true
  },
  "scores": {
    "user_naturalness": 1-5,
    "threshold_grounding": 1-5,
    "clarity": 1-5,
    "style_distinctiveness": 1-5
  },
  "issues": ["..."],
  "summary": "one short sentence"
}
```

Figure 7: The LLM judge prompt used to validate candidate user instructions before they enter the final recipe-level prompt pool.

Direct Execution Prompt (shared wrapper)

```
Task:
{user_requirement}
Raw input text: <input>{input_text}</input>
Return tagged output only.
Use exactly this format: {output_hint}
Rules:
- status must be KEEP or DROP inside <status>...</status>.
- Put the output text inside <clean_text>...</clean_text>.
- If status is KEEP, clean_text must be the final refined text.
- If status is DROP, clean_text must be the text state at the point where the sample is
     rejected.
- Preserve backslashes exactly as plain text; do not JSON-escape them.
- Do not output markdown, code fences, or explanations.
```

Figure 8: Direct Mode prompt, which also serves as the shared wrapper for Few-Shot, Plan-First, and State-Aware modes (Figures 9–11).

Few-Shot — examples prepended before the shared wrapper (Figure 8)

```
Example 1
Task:
{example_1_user_requirement}
Raw input text: <input>{example_1_input_text}</input>
Correct output:
<status>{example_1_reference_status}</status><clean_text>{example_1_reference_text}</
    clean_text>

Example 2
Task:
{example_2_user_requirement}
Raw input text: <input>{example_2_input_text}</input>
Correct output:
<status>{example_2_reference_status}</status><clean_text>{example_2_reference_text}</
    clean_text>
```

Figure 9: Few-Shot Mode: two solved examples from the same track are inserted in the shared wrapper. The shared wrapper (Figure 8) follows unchanged.

Plan-First — analysis instruction appended to `{user_requirement}`

```
Before cleaning the text, first write a short analysis block in <analyze>...</analyze>.
Use that block to restate the requested procedure as a concise standardized execution recipe.
List the steps in order. For each step, say what to do and include the relevant rule or
threshold when there is one.

Example analysis format:
<analyze>
Step 1: Remove repeated sentences.
Step 2: Normalize whitespace.
Step 3: Apply the word repetition filter with ratio <= 0.2, and drop the text if it fails.
</analyze>

Then execute the procedure on the text.
If a filter rejects the sample, stop at the rejection point and return DROP.
```

Figure 10: Plan-First Mode: this analysis instruction is appended to `{user_requirement}` in the shared wrapper (Figure 8), asking the model to restate the recipe as an ordered execution plan before producing output.

State-Aware — analysis instruction appended to `{user_requirement}`

```
Before cleaning the text, first write a short analysis block in <analyze>...</analyze>.
Use that block to identify the intermediate text states that matter for correctness.
You may refer to them with names such as S0, S1, and S2.
Only text-changing steps should create a new state, while filter steps should be described
as operating on the relevant existing state.
Focus on which operation or filter should be applied to which state, especially when order
matters.

Example analysis format:
<analyze>
A useful state view is S0 = raw text and S1 = text after repeated-sentence removal.
The key risk is applying the repetition filter on S0 instead of S1; this filter should be
evaluated on S1 because the cleanup step changes the statistics.
</analyze>

In that analysis:
- refer only to the states or transitions most likely to change the result if used incorrectly
    ;
- emphasize the specific state where an important filter or operation must be applied;
- do not analyze every step if most steps are unambiguous.

Then apply the intended procedure to the text.
If a filter rejects the sample, stop at the last valid state and return DROP.
```

Figure 11: State-Aware Mode: this analysis instruction is appended to `{user_requirement}` in the shared wrapper (Figure 8), asking the model to identify intermediate text states before executing the recipe.

Statistics	Value
Total Tasks	
Total Tasks	3319 (100%)
Atomic Mapper Tasks	132 (4.0%)
Atomic Filter Tasks	42 (1.3%)
Order-Agnostic Mapper Tasks	497 (15.0%)
Mapper-Order Sensitivity Tasks	143 (4.3%)
Filter-Order Sensitivity Tasks	2505 (75.5%)
Input Text	
Avg. Input Length (chars)	2638.9
Median Input Length (chars)	638.0
Min / Max Input Length (chars)	27 / 9997
Short / Medium / Long Inputs	2322 / 632 / 365
Recipe Coverage	
Active Operators Covered	29
Unique Recipe Templates Mined	112
Unique Executable Recipe Variants	596
Avg. Recipe Length Per Instance	4.25
Task Instruction	
Avg. Prompt Variants Per Sample	10.4
Median Prompt Variants Per Sample	11.0
Avg. Instruction Words	45.5
Median Instruction Words	44.0

Table 7: Overall statistics of CDR-Bench.

Domain	Tasks	Recipes	Order-F	Order-M
LaTeX Refinement (LR)	468	9	24	13
RAG Preparation (RP)	486	10	25	11
Privacy Redaction (PR)	1259	23	65	12
Web Refinement (WR)	1106	21	53	9
Total	3319	63	167	45

Table 8: Domain-level data and recipe coverage of CDR-Bench.

A.7 Our Position

Existing works primarily evaluate either code-driven data workflows over structured tables or isolated content-level curation tasks. CDR-Bench instead targets direct recipe execution for compositional, order-sensitive refinement over unstructured text with deterministic references, isolating recipe execution from coding, tool-call, and environment interaction.

B Experiment Details

B.1 Hyperparameters

All models are evaluated with temperature set to 0 and a maximum output length of 32,768 tokens in non-thinking mode. We use a unified output schema across all models, requiring a `status` field in `{KEEP, DROP}` and a `clean_text` field. For RS@3 and OCS@3, each recipe is evaluated un-

der three verbalization styles, and an instance is considered solved if any of the three prompts succeeds. Closed-source models are accessed through the DashScope API, while open-source models are deployed on 8 NVIDIA A100 GPUs.

B.2 Additional Experimental Results

Figure 12 and Figure 5 show the impact of instruction styles and thinking mode on recipe execution performance across tracks.

Effect of Different Prompting Strategies Table 2 compares prompt-level strategies against Direct Mode. *Few-Shot* improves atomic execution by 5–6 pp for Qwen3.6-35B-A3B but offers little benefit at the recipe level. *Plan-First* brings moderate gains on order-sensitive settings for GPT-5.4 but remains inconsistent across models. *State-Aware* is the most effective strategy: by explicitly identifying intermediate text states and specifying which operation or filter applies to each, it consistently improves order-sensitive performance, increasing Order-F OCS@3 by 4 pp for GPT-5.4 and 9 pp for Qwen3.6-35B-A3B, while also benefiting Agnostic-M, suggesting that explicit state tracking helps beyond strictly order-sensitive settings. Nevertheless, absolute OCS@3 scores remain low across all strategies, indicating that prompt engineering alone is insufficient to fully resolve recipe execution failures.

Effect of Instruction Styles Figure 12 (left) shows that RS@K increases steadily with the number of styles attempted, as different phrasings provide complementary benefits, though the Atomic–Compositional gap persists across all K. The right panel shows that operation-enumerating styles such as *Step-by-Step* and *Brief* outperform outcome-oriented styles such as *Goal-First* and *Scenario Story* on transformation-heavy tracks, as the latter require models to infer intermediate procedures from the desired end state. On Order-M, all styles collapse into a narrow performance range, confirming that instruction phrasing alone cannot compensate for the fine-grained state tracking required for order-sensitive execution.

Effect of Thinking Mode Figure 5 shows that thinking mode substantially improves RS@3 across all compositional tracks, with the largest gains on Order-F (+19.6 pp for Qwen3.6-35B-A3B and +17.9 pp for Qwen3.6-27B), where explicit multi-step reasoning most directly benefits filter evaluation over transformed text states. Gains on Order-M are comparatively modest, as mapper ordering errors are more implicit and harder to surface even with extended reasoning.

B.3 Failure Mode

Failure Mode	Definition
Filter Threshold Error	Incorrect KEEP/DROP decision under a filter criterion.
Missed Operator	One or more required operators are skipped; output remains close to input.
Under-application	Cleanup is partial; residual artifacts remain.
Over-application	Valid content is removed or modified beyond the recipe.
Wrong Order	Operators are applied in an order inconsistent with the recipe.
Formatting Drift	Output is close to reference but differs in deterministic surface formatting.
Semantic Rewrite	Model paraphrases or rewrites content instead of executing the recipe.
Instruction Misread	Model misunderstands or ignores a recipe constraint.
Format / Parse Error	Output cannot be parsed into the required schema.

Table 9: Failure-mode taxonomy and definitions.

To better understand where models fail beyond aggregate recipe-success scores, we conduct an error analysis over representative model outputs. We assign each failed prediction to one primary failure mode according to the taxonomy in Table 9, separating status-level failures such as incorrect KEEP/DROP decisions from text-level execution failures such as skipped operators, incomplete cleanup, excessive deletion, order violations, formatting drift, and semantic rewriting. When multiple symptoms appear in the same prediction, we assign the most causally dominant error type, prioritizing status errors before text-level deviations. Figure 13 summarizes the resulting distribution across models and tracks, with the left panel reporting track-level failure rates and the right panels showing failure mode composition per model.

Filter-threshold errors. The dominant failure type (39.3%) across all models. Models frequently produce well-formed cleaned text but misjudge the KEEP/DROP outcome, suggesting that exact numerical or statistical filtering criteria are harder to apply

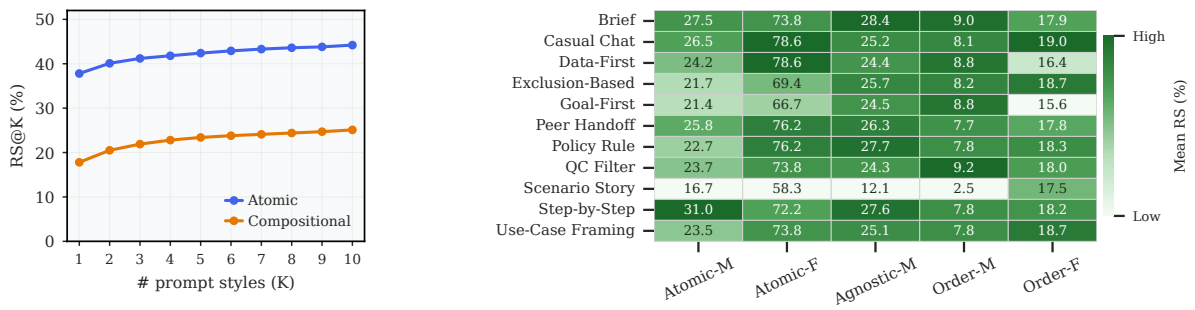

Figure 12: RS@K curves by task family (left) and mean RS across tracks and prompt styles (right).

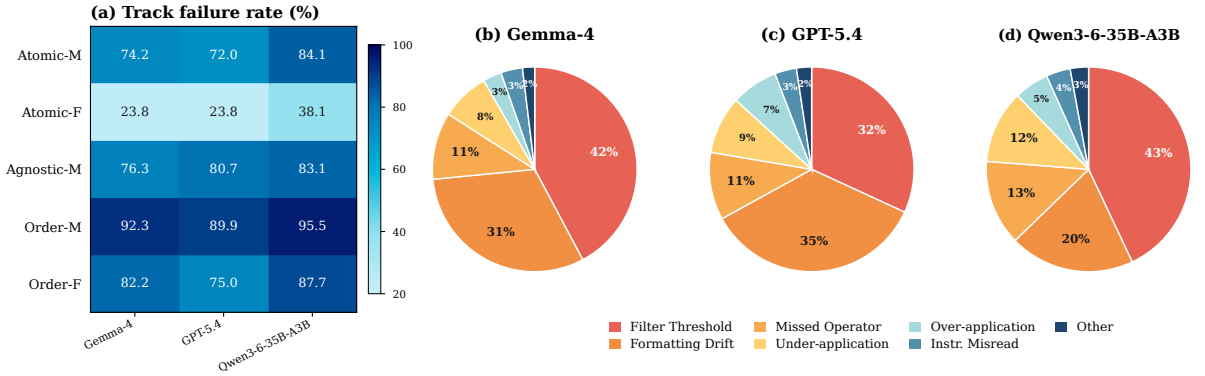

Figure 13: Failure mode analysis across representative models. (a) Track-level failure rates for Gemma-4, GPT-5.4, and Qwen3.6-35B-A3B. (b)–(d) Failure mode distributions.

than the text transformations themselves.

Mapper execution failures. On transformation-heavy tracks, formatting drift (28.4%) is the leading error, followed by missed operators (11.7%) and under-application (9.5%). Together these indicate that models tend to approximate the requested cleanup rather than reproduce deterministic operator behavior exactly. Over-application and semantic rewrites appear less frequently but reflect a distinct failure mode where models modify content beyond what the recipe specifies. We further analyze ORDER-M errors by measuring edit distance from each incorrect model output to both the canonical (original) gold and the perturbed (swapped) gold, to determine whether failures reflect random generation or systematic order confusion.

As shown in Table 3, 59.4% of errors are closer to the canonical gold than to the required swapped ordering, suggesting that models revert to a familiar argument structure rather than following the specified permutation. The effect is strongest in Gemma-4 (68.0%), while GPT-5.4 and Qwen3.6-35B-A3B are more balanced, indicating partial instruction-following that nonetheless fails to align all relational slots correctly.

Stopping behavior in DROP cases. Figure 6 analyzes Order-F DROP cases, where a rejecting filter should halt execution. In Pre-DROP cases, mean RS is nearly monotonically aligned with whether predictions are closer to t_{stop} than t_{full}, confirming that the primary error is continuing execution after rejection. GPT-5.4 achieves the highest RS together with the strongest preference for t_{stop}. Smaller models such as Qwen3.6-35B-A3B obtain lower overall RS but remain relatively close to t_{stop}, showing stronger stopping behavior than their overall execution performance would suggest. Mid-DROP cases impose an additional requirement: models must correctly rewrite the intermediate text state before terminating, making rewriting quality a further bottleneck for smaller open models. Gemma-4 stands out among similarly sized models with relatively high RS and stronger stopping behavior, reflecting both better intermediate-state rewriting and more reliable early termination.

C Real-Scenario Evaluation Details

C.1 Motivation and Design

The main evaluation tracks rely on rule-based Data-Juicer operators, which provide deterministic refer-

ences but naturally invite the question of whether the observed trends extend beyond operator replay. To complement the main benchmark, we therefore evaluate two semantic tracks grounded in realistic data-processing scenarios. The goal is not to recreate CDR-Bench in a second form, but to test whether its central observations remain relevant in settings with human annotations, higher-level semantic judgments, and less explicitly procedural task definitions.

These tracks differ from the main evaluation in three key respects. (1) Their operations correspond to semantic tasks such as entity recognition and hallucination processing rather than deterministic rule execution. (2) Ground truth comes from human-annotated datasets rather than operator replay. (3) The tasks involve higher-level semantic reasoning that is only weakly exercised in rule-based tracks. We therefore treat this section as a targeted real-world scenario study that complements, rather than replaces, the main benchmark.

C.2 PII Semantic Redaction

We use the ai4privacy PII masking dataset (Ai4Privacy Community, 2023), selecting 500 English samples with at least two distinct PII category groups via stratified sampling. PII labels are aggregated into five semantic groups: *person* (names, usernames), *location* (cities, streets, zip codes), *contact* (emails, phone numbers), *identification* (ID cards, bank accounts, tax IDs), and *temporal* (dates of birth and other time-related sensitive fields in the dataset grouping). The *Atomic* track evaluates each group in isolation using `[LABEL_N]` placeholders; the *Compositional* track requires redacting all present groups simultaneously. Ground truth is the programmatic redaction output derived from dataset annotations. Evaluation uses normalized text exact match (Recipe Success, RS) and edit-distance-based Refinement Gain (RG).

Use of the AI4Privacy dataset. We use AI4Privacy only as an evaluation benchmark for non-commercial academic research. We sample a subset from the publicly available Hugging Face release and use it exclusively for experiments in this paper. We do not redistribute the original dataset or any derivative data release.

C.3 Hallucination Detection and Correction

We use the FAVA dataset (Mishra et al., 2024) (460 samples), which provides LLM-generated text annotated with hallucination spans and corrections across several types, including entity errors, relation errors, subjective injections, contradictions, inventions, and unverifiable claims.

The *Atomic* track decomposes hallucination processing into four sub-tasks: (1) *Detection*—determine whether the text contains hallucinations (JSON with boolean flag and count); (2) *Span Extraction*—identify hallucinated text spans and types (JSON); (3) *Type Classification*—classify hallucination types present (JSON); and (4) *Correction*—produce corrected text with hallucinated content removed or replaced. The *Compositional* track requires the full pipeline in a single pass. Structured outputs are evaluated via field-level exact match, while text outputs are evaluated via normalized exact match.

C.4 Composition Gap Analysis

In the main text, Table 4 reports *Atomic* as the mean accuracy across atomic sub-tasks so that it can be directly compared with the compositional setting at a high level. Here we further analyze composition effects with a more task-specific definition.

For both tracks, we define the *composition gap* at the instance level by matching atomic and compositional predictions from the same source sample. For PII, atomic is the mean accuracy across all present semantic groups, while compositional is the single full-redaction task. For hallucination, the gap analysis focuses specifically on the final correction output, comparing atomic correction against compositional correction, since correction is the text-producing stage most directly comparable to the compositional endpoint.

These two tracks exhibit different composition mechanisms. In PII redaction, compositional difficulty arises from a multiplicative requirement: if a sample contains k PII groups, all k groups must be correctly redacted in the final output, so failure probability compounds with group count. In hallucination processing, by contrast, the dominant difficulty lies in the correction operation itself. As a result, compositional execution adds relatively little additional penalty beyond the already difficult core editing step.

Model	Contact	Location	Temporal	Ident	Person	Comp	Gap
Claude 4.5 Opus	87.0	65.7	76.2	55.4	59.5	45.0	23.7
Claude 4.6 Opus	89.3	63.7	76.2	53.9	59.7	44.4	24.2
Qwen3.6-Max	90.5	62.7	75.6	60.0	58.0	43.0	26.4
GPT5.4	81.0	58.8	71.2	53.2	39.0	33.2	27.5
DeepSeek-V4-Flash	81.4	55.9	70.6	52.9	34.7	38.0	21.1
Kimi-K2.6	87.7	55.6	70.6	48.2	45.3	28.0	33.5
GLM-5	90.1	59.8	75.6	57.9	57.2	41.0	27.1

Table 10: PII Semantic Track full results (RS %). Atomic subtasks: Contact, Location, Temporal, Ident(ification), Person. Avg: mean across all present atomic subtasks. Gap: Avg − Comp.

Model	Detect	Span	Type	Correct	Comp	Gap
Claude 4.5 Opus	28.0	15.2	11.3	22.2	22.2	0.0
Claude 4.6 Opus	26.7	10.4	8.3	21.7	20.0	1.7
Qwen3.6-Max	30.4	16.7	21.5	18.3	21.4	−3.0
GPT-5.4	28.0	10.0	10.2	13.8	15.0	−1.3
DeepSeek-V4-Flash	31.5	13.3	20.0	20.0	20.0	0.0
Kimi-K2.6	28.5	6.3	12.2	32.0	31.1	0.9
GLM-5	27.8	9.6	10.2	13.7	14.3	−0.7

Table 11: Hallucination Semantic Track full results (RS %). Atomic subtasks: Detect(ion), Span (extraction), Type (classification), Correct(ion). Comp: compositional. Gap: Correct − Comp, measuring the additional difficulty of composing detection with correction.

C.5 Results and Key Differences

Table 10 reports PII results. The composition gap ranges from 21.1 to 33.5 percentage points across models, showing that the atomic-to-compositional drop remains substantial in this semantic setting. Refinement Gain is relatively high (0.64–0.76), suggesting that many failed compositional predictions still remain meaningfully closer to the reference than exact-match accuracy alone would indicate.

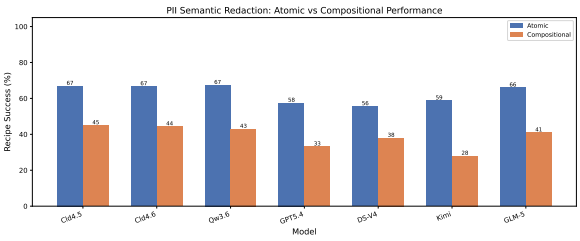

Figure 14: PII Semantic Track: atomic vs. compositional Recipe Success. The consistent gap (21–34pp) shows that multi-group redaction remains substantially harder than single-group redaction in this semantic setting.

Table 11 reports hallucination results. Unlike PII, the atomic-to-compositional difference on the final correction output is minimal, ranging from -3.0pp to +1.7pp across models. This does not make the task easy; rather, it suggests that hallucination performance is dominated by the difficulty of the core correction operation itself, with limited additional penalty from composition. Low RG values (0.17–0.33) further indicate that hallucination correction remains intrinsically challenging.

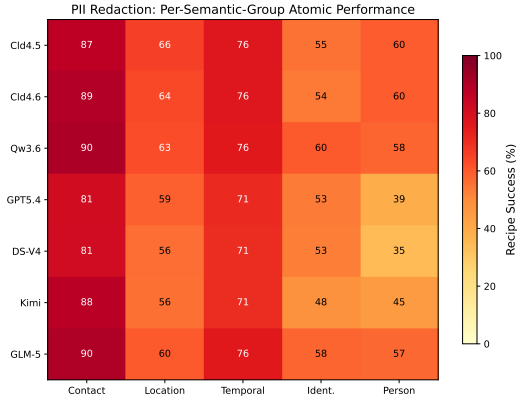

Figure 15: PII per-group atomic Recipe Success. Contact (email, phone) is easiest; person names and identification-related fields are hardest, reflecting the degree of contextual reasoning required.

Distinct Task Behaviors. The contrasting PII and hallucination patterns suggest that semantic tasks do not exhibit a single uniform form of compositional degradation. In PII, the need to satisfy multiple independent redaction requirements simultaneously produces a large atomic-to-compositional drop. In hallucination processing, the main difficulty already lies in precise correction, so the additional burden introduced by composition is comparatively limited.

Coarse-to-fine hierarchy. Figure 16 shows that binary detection (`has_hallucination`: 55–71%) substantially outperforms precise counting (`hallucination_count`: 27–32%), span localization (`spans`: 6–17%), and type classification (`types`: 8–22%). This indicates that models can often sense the presence of factual errors without being able to characterize them precisely, revealing a coarse-to-fine capability hierarchy.

Correction Paradox and Model Behavior. Across models, accuracy on hallucination-free texts is much higher than on texts that require actual correction—a phenomenon we refer to as the "correction paradox." Models are comparatively reliable at preserving already-correct text, but much less reliable at making precise targeted edits, underscoring the difficulty of faithful modification relative to content preservation.

Kimi-K2.6 exhibits a particularly sharp version of this pattern in our evaluation: it achieves 98.7% accuracy on hallucination-free texts, but its perfor-

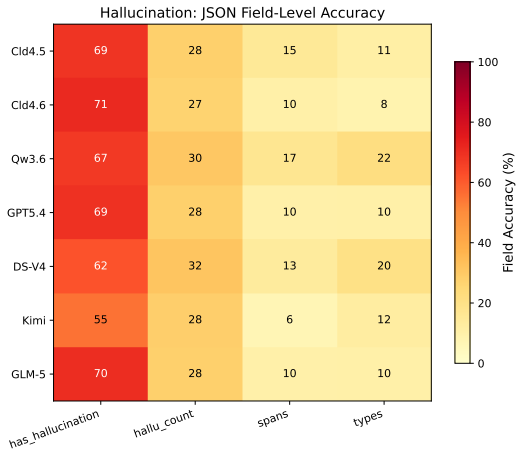

Figure 16: Hallucination JSON field-level accuracy. Binary detection (`has_hallucination`) substantially outperforms precise counting (`hallucination_count`), span localization (`spans`), and type classification (`types`), revealing a coarse-to-fine capability hierarchy.

mance drops dramatically once targeted edits are required. This behavior is consistent with strong conditional instruction following on "leave unchanged if no hallucination is present," together with difficulty on the modification step itself.

C.6 Key Findings

- **Some compositional difficulties extend to semantic settings, but not uniformly.** PII semantic redaction shows a large atomic-to-compositional drop (21–34pp), indicating that the composition bottleneck observed in the main benchmark is also relevant in semantic redaction tasks. Hallucination tasks, however, display a much smaller difference, showing that semantic composition behavior depends strongly on task structure.

- **Different composition mechanisms create different challenges.** PII compositional difficulty arises from compounded failure across multiple required redaction groups, while hallucination difficulty is dominated by the correction operation itself.

- **Difficulty reflects contextual reasoning demands.** PII group difficulty follows a stable gradient from format-driven categories (contact) to more context-dependent ones (person and identification-related fields), consistent across models.

- **Real-world semantic tasks show heterogeneous failure patterns.** The contrast between PII and hallucination suggests that the main benchmark is most directly predictive for scenarios that require multiple precise local edits to be coordinated within one final output.

C.7 Prompt Templates

We use three prompt styles per task (*direct*, *imperative checklist*, *application context*), with the direct style as default. All prompts share a common system prompt and output format. For text tasks (PII redaction, hallucination correction), the model returns `<status>` and `<clean_text>`; for JSON tasks (hallucination detection, span extraction, type classification), the model returns a structured JSON object.

System Prompt (shared)

```
You are a careful data refinement engine. Follow the user request exactly.
Return only the required output. Do not explain your reasoning.
```

User Prompt Wrapper (text tasks: PII redaction, Hallu correction)

```
Task:
{user_requirement}

Raw input text:
<input>
{input_text}
</input>

Return tagged output only.
Use exactly this format: <status>KEEP</status><clean_text>...</clean_text>
Rules:
- status must be KEEP.
- Put the processed text inside <clean_text>...</clean_text>.
- Do not output markdown, code fences, or explanations.
```

User Prompt Wrapper (JSON tasks: Hallu detection, span, type)

```
Task:
{user_requirement}

Input text:
<input>
{input_text}
</input>

Return only the JSON output. Do not wrap in code fences or add explanations.
```

Figure 17: Shared system prompt and user prompt wrappers for all semantic track evaluations.

PII Atomic — `user_requirement` (example: contact group, direct style)

```
You are a data refinement engine. Apply the following PII redaction steps to
the input text in order.

Step 1: Identify and replace all contact information (email addresses,
telephone numbers) with placeholders. Use the format [LABEL_N] where LABEL is
the original entity type (one of: EMAIL, TELEPHONENUM) and N is a counter
starting from 1 for each label type. Leave all other text unchanged.

Return the result in the format: <status>KEEP</status><clean_text>...</clean_text>
```

PII Compositional — `user_requirement` (example: 4 groups, direct style)

```
You are a data refinement engine. Apply the following PII redaction steps to
the input text in order.

Step 1: Identify and replace all contact information (email addresses,
telephone numbers) with placeholders. Use the format [LABEL_N] where LABEL is
the original entity type (one of: EMAIL, TELEPHONENUM) and N is a counter
starting from 1 for each label type. Leave all other text unchanged.
Step 2: Identify and replace all identification numbers (ID cards, bank
accounts, social security, tax IDs, driver licenses, credit cards) with
placeholders. Use the format [LABEL_N] where LABEL is the original entity
type (one of: IDCARDNUM, ACCOUNTNUM, SOCIALNUM, TAXNUM, DRIVERLICENSENUM,
CREDITCARDNUMBER) and N is a counter starting from 1 for each label type.
Leave all other text unchanged.
Step 3: Identify and replace all location information (cities, street
addresses, building numbers, zip codes) with placeholders. ...
Step 4: Identify and replace all person names and usernames (given names,
surnames, usernames) with placeholders. ...

Return the result in the format: <status>KEEP</status><clean_text>...</clean_text>
```

Figure 18: PII redaction prompt templates. Atomic shows a single-group instruction; compositional enumerates all present groups as ordered steps. The `direct` style is used as default.

Hallu Atomic: Detection — `user_requirement` (direct style)

```
Inspect the given text and determine whether it contains any hallucinated,
fabricated, or factually unsupported content. Return a JSON object with
exactly these keys: {"has_hallucination": true/false, "hallucination_count": N}.
```

Hallu Atomic: Span Extraction — `user_requirement` (direct style)

```
Identify all hallucinated text spans in the given text. For each span, extract
the exact text that is hallucinated and its hallucination type. Return a JSON
object: {"spans": [{"text": "...", "type": "..."}]}.
```

Hallu Atomic: Type Classification — `user_requirement` (direct style)

```
Classify the hallucination types present in the given text. Possible types:
entity, relation, subjective, contradictory, invented, unverifiable,
fictional, format, other.
Return a JSON object: {"types": ["type1", "type2", ...]}.
```

Hallu Atomic: Correction — `user_requirement` (direct style)

```
Correct all hallucinated content in the given text. If the text contains NO
hallucinations, return it EXACTLY as-is without any changes  do not rephrase,
reword, or reformat it. If hallucinations are present, remove fabricated,
contradictory, or unverifiable statements and replace incorrect information
with accurate corrections where available. Preserve all factual content
unchanged.
Return: <status>KEEP</status><clean_text>corrected text</clean_text>
```

Hallu Compositional — `user_requirement` (direct style)

```
You are a hallucination detection and correction engine. Analyze the given
text and perform the following steps:
1. Detect whether the text contains hallucinated content.
2. If NO hallucinations are found, return the text EXACTLY as-is without any
   changes.
3. If hallucinations exist, identify all hallucinated spans and their types
   (entity/relation/subjective/contradictory/invented/unverifiable).
4. Correct all hallucinated content by removing fabrications and applying
   known corrections. Preserve all factual content unchanged.
Return the corrected text: <status>KEEP</status><clean_text>corrected text</clean_text>
```

Figure 19: Hallucination track prompt templates. Atomic decomposes into four sub-tasks (detection, span extraction, type classification, correction), each evaluated independently. Compositional requires the full pipeline in a single pass. The `direct` style is used as default.