PROCEDURAL FAITHFULNESS IN DATA REFINEMENT: CAN LLMS EXECUTE COMPOSITIONAL, ORDER-SENSITIVE RECIPES?

Anonymous authors
Paper under double-blind review

ABSTRACT

Executing instructions that combine multiple operations requires following not only what to apply but also when. We study this procedural faithfulness in data refinement, where operation order and intermediate text states determine the correct output. Existing evaluations focus on individual tasks or end-to-end agent workflows, leaving the model's own ability to execute compositional, order-sensitive recipes insufficiently characterized. We introduce **CDR-Bench**, a diagnostic benchmark with 3,462 deterministic tasks across four domains and 29 operators, featuring paired execution-order and filter-placement variants. Across more than ten general-purpose LLMs, we find substantial gaps between individual-operation competence and complete recipe success in both rule-based and semantic refinement. Oracle interventions reveal difficulties in intermediate-text construction and filter-statistic computation, while code-assisted execution improves performance but leaves failures in recipe interpretation and order preservation. We develop training-free *state-aware prompting* that elicits explicit intermediate-state tracking and introduce **Juicer**, a specialized refinement model trained through broad operator learning followed by recipe-oriented adaptation. State-aware prompting improves compositional recipe success, while Juicer raises Order-M RS@3 from 7.02% to 31.82%. Overall, our study identifies procedural faithfulness as a distinct evaluation target in data refinement and provides a controlled framework for examining how models handle execution order, intermediate states, and stopping conditions.

1 INTRODUCTION

LLMs are increasingly asked to execute instructions that combine multiple operations in a specified order. Since each operation may change the input to subsequent steps, successful execution requires not only competence in individual operations but also adherence to *what* should be applied and *when*. These requirements arise naturally in data refinement, where raw text undergoes a sequence of transformations and filtering decisions to support applications such as pretraining corpus construction (Chen et al., 2024a; Qin et al., 2025), retrieval in RAG systems (Liu et al., 2026; Khan et al., 2024), and privacy-sensitive deployments (Garza et al., 2025; Pal et al., 2024). Using LLMs to execute refinement recipes expressed in natural language requires preserving these procedural requirements. For example, checking document length before removing private information can yield a different retention decision from checking it afterward. Different workflows may require early filtering for computational efficiency or filtering after refinement to determine whether the processed content meets retention criteria. Correctness therefore requires applying the right operations in the right order to the appropriate intermediate states, and stopping when instructed, rather than merely producing a superficially plausible output. We refer to this capability as *procedural faithfulness*.

As summarized in Table 1, existing benchmarks evaluate multi-requirement compliance (Zhou et al., 2023; Wen et al., 2024; Purpura et al., 2026), individual text transformations (Dwivedi-Yu et al., 2022; Zeng et al., 2026), or symbolic and logical procedures (Fujisawa et al., 2024; Zhang et al., 2026). More closely related are data-centric agent benchmarks (Liu et al., 2025; Lei et al., 2025), which evaluate LLM orchestration of data-processing workflows through code and tools. However, their end-to-end results reflect the combined performance of the model and its execution infrastructure, rather than isolating the model's ability to execute compositional, order-sensitive recipes directly.

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

Table 1: Comparison with representative benchmarks. ✓ denotes explicit coverage, △ partial coverage, and ✗ no explicit coverage. Definitions and the full comparison appear in Appendix B.

Benchmark	Evaluation setting	Sequential state updates	Paired order variants	Intermediate references	Deterministic oracle
Complex Instruction Following					
IFEval (Zhou et al., 2023)	Response requirements	✗	✗	✗	✓
ComplexBench (Wen et al., 2024)	Constraint composition	✗	✗	✗	△
Task-Specific Data Processing					
EditEval (Dwivedi-Yu et al., 2022)	Individual text edits	✗	✗	✗	✗
InstrEditBench (Zeng et al., 2026)	Individual text edits	✗	✗	✗	✗
Data-Centric Agents					
DataGovBench (Liu et al., 2025)	Code/tool workflows	✓	✗	✗	△
DAComp (Lei et al., 2025)	Code/tool workflows	✓	✗	✗	△
Procedure Following					
ProcBench (Fujisawa et al., 2024)	Symbolic procedures	✓	✗	✓	✓
LogicIFEval (Zhang et al., 2026)	Logic simulation	✓	✗	✗	✓
CDR-Bench (ours)	Direct recipe execution	✓	✓	✓	✓

To bridge this gap, we introduce **CDR-Bench**, a benchmark for compositional, order-sensitive data refinement. Given an input text and a natural-language recipe, the model directly returns the refined text and a KEEP/DROP decision. The benchmark pairs recipes that share the same input and operators but differ in execution order or filter placement, testing whether model outputs follow the specified procedural requirements. Executable reference procedures provide deterministic intermediate states and final outputs, supporting both objective evaluation and controlled diagnosis. Its deterministic core comprises 3,462 tasks across four domains and 29 operators, with atomic tasks providing a reference for assessing compositional execution.

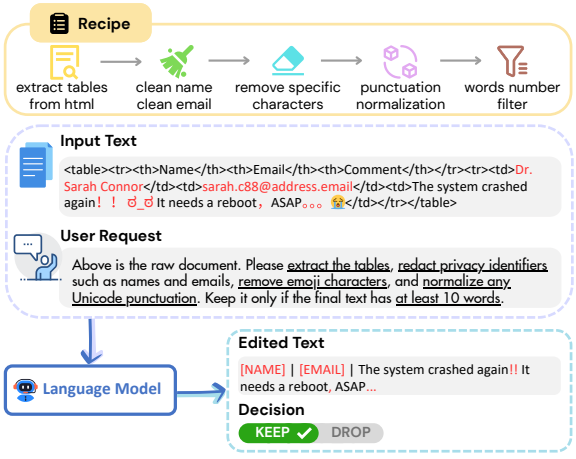

Figure 1: Data refinement recipe execution.

Through experiments on more than ten general-purpose LLMs, we find substantial gaps between individual-operation competence and faithful execution of complete recipes across both rule-based and semantic refinement. Oracle interventions reveal difficulties in constructing intermediate text and computing filter statistics, while code-assisted execution substantially improves performance but leaves failures in recipe interpretation and order preservation. Motivated by these findings, we develop a training-free *state-aware prompting* strategy that encourages models to track intermediate text states, improving recipe execution accuracy. We further introduce **Juicer**, a specialized data-refinement model trained through two phases of supervised fine-tuning, progressing from broad operator learning to recipe-oriented adaptation, followed by instruction-following reinforcement learning. Juicer improves all five core-track aggregate scores over its base model, increasing Order-M RS@3 from 7.02% to 31.82%. Together, these findings establish CDR-Bench as a reproducible testbed for measuring and diagnosing procedural faithfulness, while our prompting strategy and Juicer demonstrate progress toward more faithful execution of compositional, order-sensitive recipes.

2 CDR-Bench

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

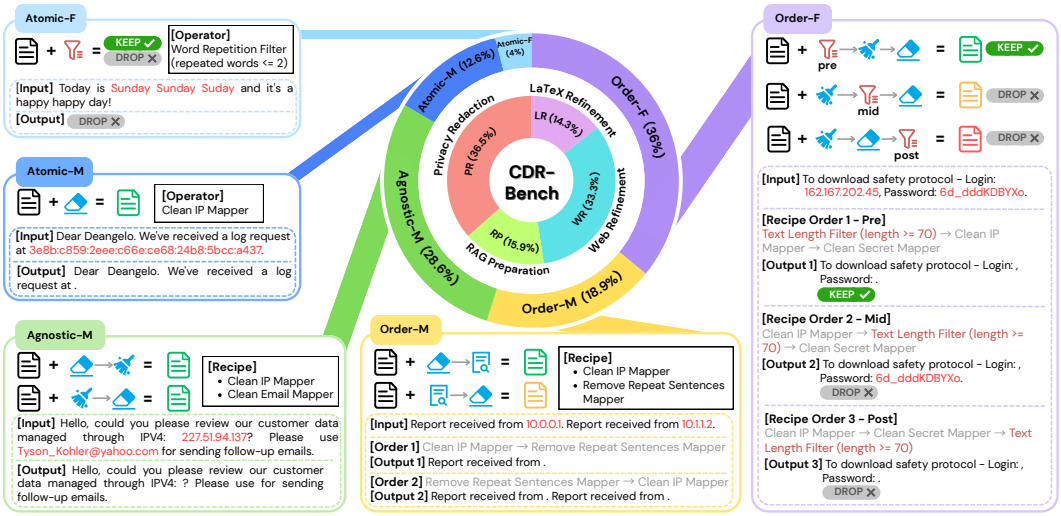

Figure 2: Overview of the CDR-Bench. The benchmark spans four data refinement domains (Privacy Redaction, Web Refinement, LaTeX Refinement, and RAG Preparation). The benchmark evaluates three levels of recipe execution: (i) single operators (**Atomic-M/F**); (ii) order-agnostic recipes (**Agnostic-M**); 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. Percentages are computed over unique recipe templates across domains and tracks.

$r = (o_1, o_2, \ldots, o_n)$ is an ordered sequence of data-processing operators[1] 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 all operators if $s^\star = \texttt{KEEP}$, or the text state before the first rejecting filter 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 $E(t_0, r)$.

Order-Sensitive Recipes Beyond 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 outcomes under different 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. 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.

[1]Each o_i denotes a data-processing operator. Following Data-Juicer (Chen et al., 2024a; 2025), we consider two types of operators: *mappers* (M), which rewrite text, and *filters* (F), which output binary KEEP/DROP decisions.

Data Collection and Operator Activation We source data from Common Crawl snapshot, arXiv preprints (TIGER-Lab, 2025), Wikipedia, GovReport (Huang et al., 2021), and multiple PII datasets (Nutrient, 2025; Ai4Privacy Community, 2023a; Steier et al., 2025), organized into four 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 C.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 combinations within each retained family to serve as final recipes. Pseudocode is provided in Appendix C.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 mapper recipes from fixed mapper combinations without selecting for order sensitivity, providing a baseline for compositional 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 C.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 9), 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 12). 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, 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 use Recipe Success (RS) as our primary metric for faithful execution and Refinement Gain (RG) as a complementary measure of progress toward the reference output.

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

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

A prediction is successful only if it correctly predicts the execution status and reconstructs the reference text. We report RS@K, where an instance is solved if the model succeeds under at least one of K instruction styles, reducing sensitivity to how the recipe is worded.

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

$$\mathrm{RG} = \max\left\{0,\ 1 - \frac{d(\hat{t}, t^\star)}{d(t_0, t^\star) + \epsilon}\right\}, \tag{2}$$

where $\epsilon > 0$ is a small constant to prevent division by zero when the initial input already perfectly matches the reference. $\mathrm{RG} = 1$ 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

The core of CDR-Bench contains 3,462 tasks spanning four domains, including 3,288 compositional tasks and 174 atomic tasks. We report detailed statistics in Appendix C.5.

3 Experimental Results

3.1 Experiment Setup

Evaluated Models We evaluate a range of state-of-the-art LLMs, spanning open-source models such as Qwen3.6-35b-a3b (Qwen Team, 2026a;b), DeepSeek-V4 (DeepSeek-AI, 2026), GLM-5 (GLM-5-Team, 2026), Gemma-4 (Google DeepMind, 2026), Llama-4 (Meta AI, 2025), and Kimi-K3 (Kimi Team et al., 2026), as well as proprietary model families such as Claude (Anthropic, 2025; 2026a;c;b) and GPT (OpenAI, 2026a;b). We additionally assess Jev (Almeida, 2026), a specialized decision model, on selected CDR-Bench tasks and compare its performance with the evaluated LLMs in Appendix G.

Evaluation Protocol All models are evaluated in Direct Mode by default, without demonstrations, planning instructions, or explicit intermediate-state guidance. Each recipe is verbalized using the 11 instruction styles summarized in Table 9. For each instance, we deterministically sample three distinct styles and report RS@3 and RG. Since data refinement is often performed at corpus scale, where inference efficiency is important, we disable reasoning modes whenever explicit controls are available. Additional model and inference settings are provided in Appendix D.1.

3.2 RQ1: How Reliably Do LLMs Execute Refinement Recipes?

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.

Atomic competence does not ensure recipe success. 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 (Gemma-4-31B-IT) shows a similar 4 pp decline. The degradation becomes even more severe in order-sensitive settings. These results show that performance on isolated operations does not ensure reliable execution of compositional recipes, particularly when operation order affects the required output.

Order-sensitive composition is harder to execute. Beyond combining multiple operations, Order-M requires respecting an execution sequence that changes the result. We construct these cases by swapping operations in compositional recipes and retaining pairs with different reference outputs. Every evaluated model achieves lower RS@3 on Order-M than on Agnostic-M. DeepSeek-V4-Flash drops by 19.0 pp, while Gemma-4 decreases from 25.55% to 8.04%. These results show that models are less reliable on compositional recipes whose outputs depend on operation order, highlighting the importance of procedural faithfulness beyond completing the individual transformations.

Deferred filtering reduces recipe success. Unlike filtering the original input, later filtering requires models to correctly execute preceding transformations and evaluate the filtering criterion on the

Table 2: Main results on CDR-Bench, measured by recipe success (RS@3, %; higher is better). Models use non-thinking mode unless marked with †, which denotes low reasoning effort. Order-M reports the original and swapped orders; Order-F reports Pre, Mid, and Post filter placements. Best and second-best results in each column are **bolded** and underlined. Full results, refinement gain, and 95% confidence intervals are reported in Tables 13–16.

Model	Atomic (Single Operator)		Compositional (Multi-Operator Recipe)							
	Atomic-M	Atomic-F	Agnostic-M	Order-M			Order-F			
				Original	Swapped	Avg.	Pre	Mid	Post	Avg.
Closed-Source Models										
Qwen3.8-Max	28.79	**92.86**	21.93	10.49	12.59	11.54	55.33	23.11	15.33	31.26
Qwen3.6-Max	26.15	76.19	23.94	9.09	5.59	7.34	48.14	21.32	15.81	28.42
Claude Opus 4.6†	29.55	73.81	22.33	6.99	**17.48**	12.24	57.96	**33.17**	**23.95**	**38.36**
GPT-5.4	**30.30**	88.10	22.74	11.89	16.08	**13.99**	**62.04**	25.03	14.97	34.01
Open-Source Models										
Qwen3.6-27B	24.24	80.95	23.73	11.89	4.93	8.41	46.95	19.09	17.51	27.85
Qwen3.6-35B-A3B	18.18	78.05	20.40	8.45	5.59	7.02	50.98	12.17	12.89	25.35
DeepSeek-V4-Flash	24.62	76.19	23.94	6.99	2.80	4.90	39.57	11.75	13.67	21.66
DeepSeek-V4-Pro	22.73	76.19	24.55	8.39	6.29	7.34	31.38	15.09	14.61	20.36
Kimi-K3	**30.30**	80.95	24.55	12.59	5.59	9.09	43.47	20.96	15.33	26.59
GLM-5.2	25.00	80.95	24.95	**13.99**	4.20	9.09	32.10	19.64	17.37	23.03
Gemma-4-31B-IT	29.55	80.95	**25.55**	10.49	5.59	8.04	32.81	19.16	14.01	22.00
Llama-4-Scout-17B	16.92	62.50	13.59	2.82	4.93	3.87	22.41	4.47	3.99	10.29

resulting intermediate state. 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, suggesting that end-to-end execution becomes less reliable when filtering depends on earlier transformations. This difficulty in executing successive operations is also reflected in the length analysis, where the evaluated models approach zero RS@3 beyond five operations on Agnostic-M and Order-M. Instruction reformulation provides limited relief on Order-M, where different styles yield similarly low success (Appendix D.5).

3.3 RQ2: How Much Do Oracle Guidance and Code Execution Help?

We further compare two forms of assistance. On a paired Order-F diagnostic subset, oracle inputs supply either the correct intermediate text at the filter checkpoint, obtained by executing the preceding recipe operations, or its true statistic. Models predict KEEP/DROP at Mid and Post placements, with mean status accuracy reported over three formulations. Code-run executes model-generated Python programs across all five tracks and is evaluated using RS@3, which requires both the correct status and output text.

Filtering failures reflect errors in text construction and statistic extraction. Figure 3 reveals two difficulties in deferred filtering. First, supplying the correct intermediate text improves Mid/Post accuracy by 8.2/4.9 pp for Gemma-4 and 10.6/15.0 pp for Qwen3.6-35B-A3B, indicating difficulty constructing the text required for filtering. Second, even with correct text, accuracy remains at 62.9–90.8%. Supplying the true statistic raises accuracy to 99.8–100.0%, exposing substantial errors in computing statistics from text. Together, these results indicate that models struggle

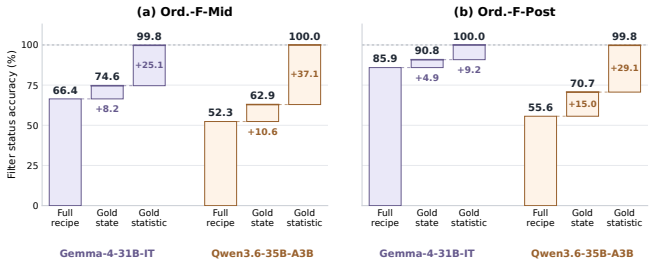

Figure 3: Oracle interventions on Ord.-F-Mid and Ord.-F-Post. Values show mean status accuracy (%) and successive gains (pp). Gold State supplies the correct intermediate text at the filter checkpoint, and Gold Statistic supplies its true statistic.

Table 3: Direct vs. Code-run RS@3 (%).

Model	Mode	A-M	A-F	Agn.-M	Ord.-M	Ord.-F
Qwen3.6-35B-A3B	Direct	18.18	78.05	20.40	7.02	25.35
	Code-run	**36.21**	**97.62**	**22.57**	**15.65**	**41.02**
Gemma-4-31B-IT	Direct	29.55	80.95	**25.55**	8.04	22.00
	Code-run	**38.14**	**97.62**	21.00	**20.37**	**50.29**

Table 4: Order-M error proximity (%).

Model	n	Orig.	Swap	Tie
Gemma-4-31B-IT	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**

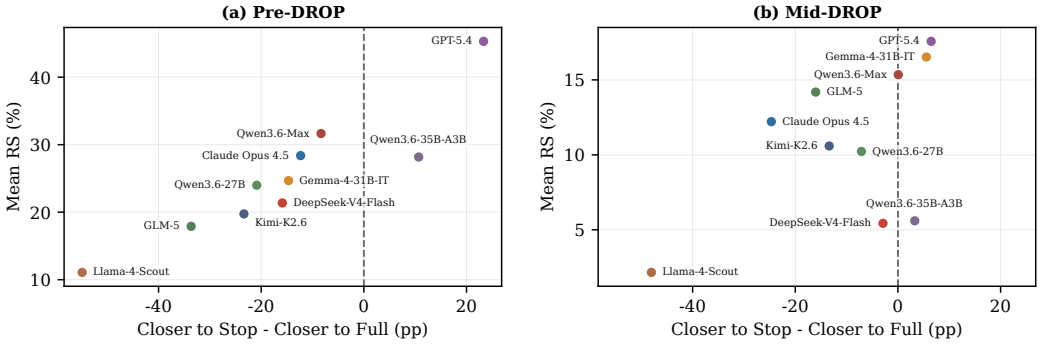

Figure 4: Stopping-state proximity and recipe success on Order-F. The horizontal axis measures the difference between the fractions of eligible DROP outputs closer to t_{stop} and t_{full}. The vertical axis reports mean RS over all evaluated outputs at the corresponding filter placement.

both to produce the correct intermediate text and to measure its properties, while applying the filtering rule to an accurate statistic is largely reliable.

Code assistance improves but does not close the recipe-execution gap. Table 3 shows that Code-run raises Atomic-F RS@3 to 97.62% for both models. Order-F also improves by 15.67 and 28.30 pp for Qwen and Gemma-4. Together with the oracle results, these gains suggest that code execution helps models obtain accurate statistics from text. However, Order-M success remains below 21% for both models, and Gemma-4 declines on Agnostic-M, suggesting that interpreting user instructions and generating code that faithfully implements compositional, order-sensitive recipes remains challenging even with a deterministic interpreter.

3.4 RQ3: How Does Execution Fail and Can Prompting Help?

We analyze execution failures (Figure 9) and whether prompting strategies can mitigate them. Among the analyzed failures, 39.3% involve incorrect KEEP/DROP decisions. Others change formatting inconsistently with the reference (28.4%), skip required operations (11.7%), or apply an operation to only part of the text that requires it (9.5%). Beyond these errors, we examine whether outputs follow the requested operation order and stop at the specified filter checkpoint. Full definitions and per-model distributions appear in Appendix D.8.

Models may favor the original order over the requested sequence. We construct swapped recipes by reordering operations in the original recipe and retaining pairs with different reference outputs. Table 4 shows that, when instructed to execute the swapped recipe, 59.4% of failed predictions are closer to the original-order reference, compared with 38.6% closer to the requested reference. This tendency is strongest for Gemma-4 at 68.0%, versus 55.4% for GPT-5.4 and 54.6% for Qwen. The asymmetry suggests that models may follow a default processing order instead of faithfully applying the requested permutation.

Models may complete transformations before applying the filter. Figure 4 compares outputs on ground-truth DROP cases with the correct stopping state t_{stop} and the full-execution state t_{full}, obtained by continuing all remaining operations despite rejection. In Pre-DROP, many models produce outputs closer to t_{full} even though the input should be rejected without any transformation.

Table 5: Training-free prompting strategies on compositional tracks. We report recipe success (RS@3) and refinement gain (RG). Colored values denote changes from Direct. Best and second-best results in each column are **bolded** and underlined.

Prompting Strategy	Agnostic-M		Order-M		Order-F			
	RS@3↑	RG↑	RS@3↑	RG↑	RS_{pre}@3↑	RS_{mid}@3↑	RS_{post}@3↑	RG↑
GPT-5.4								
Direct Prompting	22.74	53.77	13.99	52.58	62.04	25.03	14.97	48.47
+ Few-Shot	23.38+0.64	52.98-0.79	13.99+0.00	52.46-0.12	63.37+1.33	25.72+0.69	15.18+0.21	48.46-0.01
+ Plan-First	23.74+1.00	**54.45**+0.68	**18.18**+4.19	**55.37**+2.79	62.42+0.38	29.38+4.35	17.01+2.04	**53.43**+4.96
+ State-Aware	**23.84**+1.10	54.17+0.40	16.43+2.44	53.24+0.66	**75.15**+13.11	**29.52**+4.49	**17.07**+2.10	51.18+2.71
Qwen3.6-35B-A3B								
Direct Prompting	20.40	42.03	7.02	41.31	50.98	12.17	12.89	29.95
+ Few-Shot	19.19-1.21	39.99-2.04	6.32-0.70	40.41-0.90	52.20+1.22	12.11-0.06	11.42-1.47	30.41+0.46
+ Plan-First	20.52+0.12	42.25+0.22	5.94-1.08	40.03-1.28	41.16-9.82	10.84-1.33	12.98+0.09	31.39+1.44
+ State-Aware	**23.94**+3.54	**47.70**+5.67	**9.44**+2.42	**48.52**+7.21	**59.98**+9.00	**25.18**+13.01	**19.35**+6.46	**38.61**+8.66

In Mid-DROP, this tendency persists for several models, whose outputs favor completing the recipe over retaining the intermediate text at the rejection point. Together, these patterns suggest a gap in procedural faithfulness, with models potentially completing transformations before making the filtering decision rather than applying the filter at its prescribed position and stopping upon rejection.

State-aware prompting yields consistent recipe-level gains. We compare three training-free strategies with Direct prompting. *Few-Shot* adds two solved examples from the same track, *Plan-First* asks the model to generate an ordered execution plan before answering, and *State-Aware* further requires tracking intermediate text states and their corresponding operations or filters. Full prompts are provided in Appendix H.1. Table 5 shows that *Few-Shot* provides limited recipe-level gains and can even hurt Qwen3.6-35B-A3B. *Plan-First* improves several GPT-5.4 settings, especially on Order-M, but transfers inconsistently across models. *State-Aware* yields more consistent gains over Direct, improving RS@3 across all five compositional settings for both models, with particularly large gains on Order-F. Although substantial errors remain, these results suggest that explicit intermediate-state tracking is a useful prompting strategy for improving recipe execution.

3.5 RQ4: Can LLMs Execute Compositional Semantic Refinement Recipes?

We further examine compositional refinement in four semantic domains with annotated references, covering PII redaction (Ai4Privacy Community, 2023b), hallucination processing (Mishra et al., 2024), safety tagging (Ghosh et al., 2025), and rubric scoring (Wang et al., 2024). This complementary study compares individual operations with their combined requirements, extending the composition component of CDR-Bench to tasks requiring semantic judgments. Figure 11 reports an average RS@3 of 58.8% on atomic tasks and 30.8% on compositional tasks. The gaps are 51.1 pp for rubric scoring, 29.1 pp for PII redaction, 27.6 pp for safety tagging, and 4.0 pp for hallucination processing. These differences quantify the difficulty of complete outputs under the specified scoring rules. They do not by themselves isolate interference between operations, since atomic averages and joint success impose different correctness requirements. Full results appear in Appendix E.

3.6 RQ5: Can Targeted Post-Training Improve Recipe Execution?

We investigate targeted post-training through **Juicer**, a specialized refinement model initialized from Qwen3.6-35B-A3B. Its supervised fine-tuning curriculum progresses from broad operator learning, using approximately 0.5M Data-Juicer-style examples covering roughly 70 operators, to recipe-oriented adaptation emphasizing composition, execution order, filtering, and output-format compliance. Instruction-

Table 6: RS@3 (%) of Juicer vs. Qwen3.6-35B-A3B.

Model	Atomic-M	Atomic-F	Agnostic-M	Order-M	Order-F	Avg.
Base	18.18	78.05	20.40	7.02	25.35	29.80
Juicer	**51.52**	**90.48**	**26.56**	**31.82**	**42.08**	**48.49**

following reinforcement learning then produces the final model. As shown in Table 6, Juicer improves all five core-track aggregate scores under direct, non-thinking evaluation. Order-M RS@3 increases from 7.02% to 31.82%, and the five-track macro-average rises from 29.80% to 48.49%. However, Order-F Post reaches only 21.56%, and substantial semantic composition difficulties remain. Since benchmark feedback informed training-data design, we interpret these gains as in-domain development evidence. Training details and complete results appear in Appendix F.

4 Related Work

Complex Instruction Following Instruction-following benchmarks have progressed from checking individual requirements to studying their composition. IFEval (Zhou et al., 2023) provides programmatically verifiable constraints, while FollowBench (Jiang et al., 2024) and InFoBench (Qin et al., 2024) support fine-grained evaluation through incremental constraints or requirement decomposition. ComplexBench (Wen et al., 2024) models logical composition among constraints, and MOSAIC (Purpura et al., 2026) measures how their interaction, quantity, and prompt position affect compliance. These benchmarks primarily ask whether the final response jointly satisfies the requested constraints. Their ordering effects concern how constraints are presented or logically composed. CDR-Bench instead studies execution order. Each operation changes the state consumed by later operations, so satisfying all named requirements does not guarantee a procedurally correct output.

LLM-Based Data Processing LLMs have also been evaluated as data-processing components and agents. Prior work studies individual operations such as text editing, anonymization, repair, imputation, and matching (Dwivedi-Yu et al., 2022; Zeng et al., 2026; Pilán et al., 2022; Zhang et al., 2024b;a). Workflow systems and data-agent benchmarks extend this setting to pipeline construction, code generation, tool use, and execution over files or tables (Li et al., 2025; Liu et al., 2025; Lei et al., 2025; Lai et al., 2026; Chen et al., 2023). The former tests whether a model can perform an operation in isolation. The latter tests whether a complete system can accomplish a workflow. CDR-Bench targets the missing middle layer. It supplies the policy and removes code, tools, and environment interaction, allowing an error to be attributed to recipe interpretation and execution rather than to the surrounding agent stack.

Procedure Following A closely related line of work evaluates whether models can follow explicit procedures and structural dependencies. SIFo (Chen et al., 2024b) evaluates sequentially dependent instructions across text modification, question answering, mathematics, and security rule following. ProcBench (Fujisawa et al., 2024) supplies multi-step inference rules and evaluates final and step-level correctness. LogicIFEval (Zhang et al., 2026) derives verifiable instructions from code with conditions, loops, nesting, and function calls. CDR-Bench focuses on compositional data refinement with non-commutative text transformations and state-dependent filtering. It combines paired execution-order and filter-placement variants with deterministic intermediate references to evaluate order adherence and diagnose filtering and stopping errors.

5 Conclusion

We introduced CDR-Bench, a controlled benchmark for evaluating procedural faithfulness in compositional, order-sensitive data refinement. Across rule-based and semantic tasks, strong performance on individual operations does not ensure successful execution of complete recipes. Oracle interventions reveal difficulties in constructing intermediate text and computing filter statistics, while failure analyses suggest deviations from the requested execution order and stopping conditions. State-aware prompting and code-assisted execution offer partial improvements, and Juicer demonstrates the potential of targeted post-training, improving all five core-track aggregate scores over its base model. These benchmark-informed gains nevertheless coexist with persistent difficulties in deferred filtering and trade-offs across semantic tasks. Our findings highlight procedural faithfulness as a key requirement for reliable data refinement. Together, CDR-Bench, our diagnostic analyses, and prompting and post-training interventions provide a reproducible framework for measuring this capability, understanding failures in execution order and state-dependent filtering, and improving adherence to the requested procedure.

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.

Generative AI Use Disclosure

We used generative AI tools to assist in verbalizing executable recipes into natural-language instructions and to screen candidate instructions for fidelity, as described in Section 2.2. We also used generative AI tools for language editing to improve the grammar, clarity, and readability of the manuscript. All AI-generated and AI-edited content was reviewed and validated by the authors. The authors take full responsibility for the final content of this work.

Reproducibility Statement

To support reproducibility, we release benchmark instances, recipe metadata, prompt variants, evaluation code, and scripts for API and vLLM inference. The evaluation code and execution instructions are available anonymously at `https://anonymous.4open.science/r/cdrbench-6122/`. Since gold references are derived from deterministic operator execution, future work can reproduce the evaluation without relying on subjective model-based judging.

References

Ai4Privacy. Openpii 1m: Multilingual pii masking dataset (19 labels, 23 languages). `https://huggingface.co/datasets/ai4privacy/pii-masking-openpii-1m`, 2026. Hugging Face Dataset.

Ai4Privacy Community. ai4privacy/pii-masking-200k dataset. `https://huggingface.co/datasets/ai4privacy/pii-masking-200k`, 2023a. A synthetic multilingual dataset for training and evaluating PII detection and masking models with annotated spans.

Ai4Privacy Community. ai4privacy/pii-masking-400k dataset. `https://huggingface.co/datasets/ai4privacy/pii-masking-400k`, 2023b. A synthetic multilingual dataset for training and evaluating PII detection and masking models with annotated spans.

Diogo Almeida. Introducing System One Models & Jev. TypeSafe AI Blog, September 2026. URL `https://typesafe.ai/blog/introducing-system-one-models-and-jev`.

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

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

Anthropic. Introducing claude sonnet 4.6. `https://www.anthropic.com/news/claude-sonnet-4-6`, 2026b. Accessed: 2026-06-25.

Anthropic. Introducing claude opus 4.7. `https://www.anthropic.com/news/claude-opus-4-7`, 2026c. Accessed: 2026-06-25.

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. Data-juicer: A one-stop data processing system for large language models. In *SIGMOD*, 2024a.

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. Data-juicer 2.0: Cloud-scale adaptive data processing for and with foundation models. *NeurIPS*, 2025.

Sibei Chen, Hanbing Liu, Weiting Jin, Xiangyu Sun, Xiaoyao Feng, Ju Fan, Xiaoyong Du, and Nan Tang. Chatpipe: Orchestrating data preparation program by optimizing human-chatgpt interactions, 2023. URL https://arxiv.org/abs/2304.03540.

Xinyi Chen, Baohao Liao, Jirui Qi, Panagiotis Eustratiadis, Christof Monz, Arianna Bisazza, and Maarten de Rijke. The sifo benchmark: Investigating the sequential instruction following ability of large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 1691–1706. Association for Computational Linguistics, 2024b. doi: 10.18653/v1/2024.findings-emnlp.92. URL http://dx.doi.org/10.18653/v1/2024.findings-emnlp.92.

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

Jane Dwivedi-Yu, Timo Schick, Zhengbao Jiang, Maria Lomeli, Patrick Lewis, Gautier Izacard, Edouard Grave, Sebastian Riedel, and Fabio Petroni. Editeval: An instruction-based benchmark for text improvements, 2022. URL https://arxiv.org/abs/2209.13331.

Ippei Fujisawa, Sensho Nobe, Hiroki Seto, Rina Onda, Yoshiaki Uchida, Hiroki Ikoma, Pei-Chun Chien, and Ryota Kanai. Procbench: Benchmark for multi-step reasoning and following procedure, 2024. URL https://arxiv.org/abs/2410.03117.

Leon Garza, Anantaa Kotal, Aritran Piplai, Lavanya Elluri, Prajit Das, and Aman Chadha. Prvl: Quantifying the capabilities and risks of large language models for pii redaction, 2025. URL https://arxiv.org/abs/2508.05545.

Shaona Ghosh, Prasoon Varshney, Makesh Narsimhan Sreedhar, Aishwarya Padmakumar, Traian Rebedea, Jibin Rajan Varghese, and Christopher Parisien. AEGIS2.0: A diverse AI safety dataset and risks taxonomy for alignment of LLM guardrails. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 5992–6026, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.306. URL https://aclanthology.org/2025.naacl-long.306/.

GLM-5-Team. GLM-5: from vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

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

Benhao Huang, Yingzhuo Yu, Jin Huang, Xingjian Zhang, and Jiaqi Ma. Dca-bench: A benchmark for dataset curation agents, 2025. URL https://arxiv.org/abs/2406.07275.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. 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*, pp. 1419–1436, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.112. URL https://aclanthology.org/2021.naacl-main.112.

Yuxin Jiang, Yufei Wang, Xingshan Zeng, Wanjun Zhong, Liangyou Li, Fei Mi, Lifeng Shang, Xin Jiang, Qun Liu, and Wei Wang. Followbench: A multi-level fine-grained constraints following benchmark for large language models, 2024. URL https://arxiv.org/abs/2310.20410.

Ayman Asad Khan, Md Toufique Hasan, Kai Kristian Kemell, Jussi Rasku, and Pekka Abrahamsson. Developing retrieval augmented generation (rag) based llm systems from pdfs: An experience report, 2024. URL https://arxiv.org/abs/2410.15944.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, M. C., Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y. Charles, H. S. Che, Guanduo Chen, Guangyu Chen, Guanzheng Chen, Huarong Chen, Jia Chen, Jianlong Chen, Jun Chen, Kexin Chen, Peng Chen, Ruijue Chen, Wentao Chen, Xin Chen, Yang Chen, Yanru Chen, Yifei Chen, Yingjiang Chen, Yuankun Chen, Yujie Chen, Yutian Chen, Zhirong Chen, Dazhi Cheng, Yean Cheng, Jialei Cui, Jingbing Cui, Anqi Dai, Jiaqi Deng, Hao Ding, Rui Ding, Shaofeng Ding, Mengfan Dong, Mengnan Dong, Yuhao Dong,

Yuxin Dong, Angang Du, Chenzhuang Du, Dikang Du, Jusen Du, Yulun Du, Yu Fan, Jing Feng, Qiulin Feng, Yichen Feng, Kelin Fu, Qiang Fu, Fuxuan Gao, Hongcheng Gao, Jingyue Gao, Tong Gao, Weijia Gao, Shangyi Geng, Jie Gong, Linhu Gong, Shengao Gong, Xiaochen Gong, Qizheng Gu, Yicheng Gu, Shuhao Guan, Haiqing Guo, Shiqi Guo, Xiang Guo, Zhengyan Guo, Beixi Hao, Wenxin Hao, Xiaoru Hao, Dailan He, Haotian He, Lehan He, Qi He, Weiran He, Xinran He, Xinyi He, Yibo He, Yunjia He, Chao Hong, Tiange Hong, Hao Hu, Jiaxi Hu, Ruikun Hu, Weiming Hu, Yangyang Hu, Zhenxing Hu, Liang Hua, Jinbin Huang, Ke Huang, Ruiyuan Huang, Siying Huang, Weixiao Huang, Yan Huang, Zhengjie Huang, Zhiqi Huang, Yulong Hui, Chaobo Jia, Yutong Jiang, Zhejun Jiang, Zuoyou Jiang, Wenyi Jin, Xinyi Jin, Yu Jing, Huanjun Kong, Guokun Lai, Aidi Li, Cheng Li, Chengyuan Li, Cong Li, Fang Li, Guanyu Li, Haoyang Li, Jia Li, Junxiong Li, Lei Li, Letian Li, Lincan Li, Weihong Li, Wentao Li, Xintong Li, Yang Li, Yishen Li, Yiwei Li, Yuxiao Li, Zhaowei Li, Zhaoxi Li, Zheming Li, Zhengxiao Li, Zhiyuan Li, Jiawei Lin, Xiaohan Lin, Yibo Lin, Zichao Lin, Ziyan Lin, Bill Liu, Boxiao Liu, Chuan Liu, Liang Liu, Shaowei Liu, Shudong Liu, Shuran Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yanming Liu, Yibo Liu, Yipeng Liu, Zhengying Liu, Zhiheng Liu, Enzhe Lu, Haoyu Lu, Linqiang Lu, Tingzhan Lu, Zhiyuan Lu, Aotian Luo, G. Luo, Junyu Luo, Yifan Luo, B. Lyu, Wenzhou Lyu, Shaoguang Mao, Yuan Mei, Xin Men, Minqing Ni, Yixuan Niu, Siyuan Pan, Shujun Peng, Zhangyang Qi, Ruoyu Qin, ZeChao Qin, Zeyu Qin, Haiquan Qiu, Jianxin Qiu, Jiezhong Qiu, Bowen Qu, Yuhao Qu, Zeyu Shang, Youbo Shao, Han Shen, Jincheng Shi, Juanfeng Shi, Lidong Shi, Shengyuan Shi, Wingchun Siu, Pengwei Song, Xiaoxi Song, Jianlin Su, Yunfeng Su, Zhaochen Su, Lin Sui, Jingsong Sun, Junyao Sun, Shaoning Sun, Shuzhe Sun, Tongyu Sun, Yujun Sun, Yunpeng Tai, Chuning Tang, Heyi Tang, Sirui Tang, Zecheng Tang, Chaoran Tian, Rongpeng Tian, Yu Tian, Wei Tu, Chensi Wang, Chuang Wang, Chunjie Wang, Dinglu Wang, Feng Wang, Hailong Wang, Haiming Wang, Hao Wang, Hao Wang, Huaqing Wang, Hui Wang, Jiayi Wang, Jinglong Wang, Jinhong Wang, Jiuzheng Wang, Linian Wang, Shaobo Wang, Shenzhi Wang, Shuyi Wang, Si Wang, Siyuan Wang, Tianfu Wang, Wenjue Wang, Xingran Wang, Xinmei Wang, Xinyuan Wang, Xusheng Wang, Yalin Wang, Yangkun Wang, Yao Wang, Yaoyu Wang, Yejie Wang, Yiqin Wang, Yucheng Wang, Yuzhi Wang, Zhaoji Wang, Zhaowei Wang, Zhengtao Wang, Zhenhao Wang, Zhongsheng Wang, Zifan Wang, Chu Wei, Ming Wei, Shouxin Wei, Zichen Wen, Fan Wu, Haoning Wu, Rucong Wu, Wenhao Wu, Xiaoxue Wu, Yingcong Wu, Yongqi Wu, Yuxin Wu, Zijian Wu, Xinglang Xian, Chenxuan Xiang, Yuye Xiang, Bocheng Xiao, Chenjun Xiao, Xin Xiao, Jin Xie, Xiaotong Xie, Yifeng Xie, Zhe Xie, Bowei Xing, Yiming Xiong, Baosheng Xu, Boyu Xu, Jiale Xu, Jianfan Xu, Jing Xu, Jinjing Xu, L. H. Xu, Qingtao Xu, Shuyao Xu, Suting Xu, Tiantian Xu, Tianxiang Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ye Xu, Yueni Xu, Ziyao Xu, Haonan Xue, Junjie Yan, Yaoyao Yan, Fan Yang, Guangyao Yang, Hao Yang, Junwei Yang, Ruoyu Yang, Wenjie Yang, Xiaofei Yang, Xinyu Yang, Yi Yang, Yiling Yang, Ying Yang, Yuchen Yang, Zhen Yang, Zhilin Yang, Zian Yang, Zuhao Yang, Haotian Yao, Dan Ye, Haoran Ye, Wenjie Ye, Zhanbo Ye, Bohong Yin, Haoxiang Yin, Xietong Yin, Chengzhen Yu, Haozhen Yu, Longhui Yu, Shengnan Yu, Shuying Yu, Tianxiang Yu, Enming Yuan, Mengjie Yuan, Tongtian Yue, Wei Yue, Yang Yue, Dunyuan Zha, Haobing Zhan, B. H. Zhang, Dehao Zhang, Fei Zhang, Hao Zhang, Haoyuan Zhang, Huanyu Zhang, Jiapei Zhang, Jiaxuan Zhang, Jin Zhang, Kaiyi Zhang, Miaozhen Zhang, Puqi Zhang, Qinglei Zhang, Rong Zhang, Rui Zhang, Shaoshuai Zhang, Shiyi Zhang, Xiaobin Zhang, Xiaoyun Zhang, Y. Zhang, Yangkun Zhang, Ye Zhang, Yichi Zhang, Yikun Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Zijing Zhang, Bin Zhao, Chenguang Zhao, Feifan Zhao, Jinglun Zhao, Jinxiang Zhao, Shuai Zhao, Wenshuo Zhao, Xiangyu Zhao, Xuanle Zhao, Yikai Zhao, Zijia Zhao, Haozhi Zheng, Huabin Zheng, Ruihan Zheng, Shaojie Zheng, Tengyang Zheng, Haofeng Zhong, Lei Zhong, Longguang Zhong, M. Zhou, Qiankang Zhou, Runjie Zhou, Ruozhang Zhou, Xinyu Zhou, Yiqiao Zhou, Zaida Zhou, Jinguo Zhu, Liya Zhu, Xinhao Zhu, Yangjunfeng Zhu, Yuxuan Zhu, Zhen Zhu, Chen Zhuang, Weiyu Zhuang, and Xinxing Zu. Kimi k3: Open frontier intelligence, 2026. URL https://arxiv.org/abs/2607.24653.

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. Kramabench: A benchmark for ai systems on data-to-insight pipelines over data lakes, 2026. URL https://arxiv.org/abs/2506.06541.

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. Dacomp: Benchmarking data agents across the full data intelligence lifecycle, 2025. URL https://arxiv.org/abs/2512.04324.

Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

Lan Li, Liri Fang, Bertram Ludäscher, and Vetle I. Torvik. Autodcworkflow: Llm-based data cleaning workflow auto-generation and benchmark, 2025. URL https://arxiv.org/abs/2412.06724.

Jingyu Liu, Jiaen Lin, and Yong Liu. Tackling the inherent difficulty of noise filtering in rag, 2026. URL https://arxiv.org/abs/2601.01896.

Zhou Liu, Zhaoyang Han, Guochen Yan, Hao Liang, Bohan Zeng, Xing Chen, Yuanfeng Song, and Wentao Zhang. Datagovbench: Benchmarking llm agents for real-world data governance workflows, 2025. URL https://arxiv.org/abs/2512.04416.

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

Abhika Mishra, Akari Asai, Vidhisha Balachandran, Yizhong Wang, Graham Neubig, Yulia Tsvetkov, and Hannaneh Hajishirzi. Fine-grained hallucination detection and editing for language models, 2024. URL https://arxiv.org/abs/2401.06855.

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

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

OpenAI. Gpt-5.5 model documentation. https://developers.openai.com/api/docs/models/gpt-5.5, 2026b. Accessed: 2026-06-25.

Anwesan Pal, Radhika Bhargava, Kyle Hinsz, Jacques Esterhuizen, and Sudipta Bhattacharya. The empirical impact of data sanitization on language models, 2024. URL https://arxiv.org/abs/2411.05978.

Ildikó Pilán, Pierre Lison, Lilja Øvrelid, Anthi Papadopoulou, David Sánchez, and Montserrat Batet. The text anonymization benchmark (TAB): A dedicated corpus and evaluation framework for text anonymization. *Computational Linguistics*, 48(4):1053–1101, December 2022. doi: 10.1162/coli_a_00458. URL https://aclanthology.org/2022.cl-4.19/.

Alberto Purpura, Li Wang, Sahil Badyal, Gene Beaufrand, and Adam Faulkner. Deconstructing instruction-following: A new benchmark for granular evaluation of large language model instruction compliance abilities. In Vera Demberg, Kentaro Inui, and Lluís Marquez (eds.), *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1334–1349, Rabat, Morocco, March 2026. Association for Computational Linguistics. ISBN 979-8-89176-380-7. doi: 10.18653/v1/2026.eacl-long.62. URL https://aclanthology.org/2026.eacl-long.62/.

Yiwei Qin, Kaiqiang Song, Yebowen Hu, Wenlin Yao, Sangwoo Cho, Xiaoyang Wang, Xuansheng Wu, Fei Liu, Pengfei Liu, and Dong Yu. Infobench: Evaluating instruction following ability in large language models, 2024. URL https://arxiv.org/abs/2401.03601.

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. Scaling laws of synthetic data for language models, 2025. URL https://arxiv.org/abs/2503.19551.

Qwen Team. Qwen3.6-27B: Flagship-level coding in a 27B dense model, April 2026a. URL https://qwen.ai/blog?id=qwen3.6-27b.

Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all, April 2026b. URL https://qwen.ai/blog?id=qwen3.6-35b-a3b.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016. URL https://arxiv.org/abs/1606.05250.

Amy Steier, Andre Manoel, Alexa Haushalter, and Maarten Van Segbroeck. Nemotron-pii: Synthesized data for privacy-preserving ai. https://huggingface.co/datasets/nvidia/Nemotron-PII, 2025. Hugging Face Dataset.

TIGER-Lab. arxiv-latex-5t. https://huggingface.co/datasets/TIGER-Lab/arxiv-latex-5T, 2025. Hugging Face Dataset.

Zhilin Wang, Yi Dong, Olivier Delalleau, Jiaqi Zeng, Gerald Shen, Daniel Egert, Jimmy J. Zhang, Makesh Narsimhan Sreedhar, and Oleksii Kuchaiev. Helpsteer2: Open-source dataset for training top-performing reward models, 2024.

Bosi Wen, Pei Ke, Xiaotao Gu, Lindong Wu, Hao Huang, Jinfeng Zhou, Wenchuang Li, Binxin Hu, Wendy Gao, Jiaxin Xu, Yiming Liu, Jie Tang, Hongning Wang, and Minlie Huang. Benchmarking complex instruction-following with multiple constraints composition, 2024. URL https://arxiv.org/abs/2407.03978.

Wei Xu, Courtney Napoles, Ellie Pavlick, Quanze Chen, and Chris Callison-Burch. Optimizing statistical machine translation for text simplification. *Transactions of the Association for Computational Linguistics*, 4:401–415, 2016. doi: 10.1162/tacl_a_00107. URL https://aclanthology.org/Q16-1029/.

Yiming Zeng, Wanhao Yu, Zexin Li, Tao Ren, Yu Ma, Jinghan Cao, Xiyan Chen, and Tingting Yu. Bridging the editing gap in llms: Fineedit for precise and targeted text modifications, 2026. URL https://arxiv.org/abs/2502.13358.

Haochen Zhang, Yuyang Dong, Chuan Xiao, and Masafumi Oyamada. Jellyfish: Instruction-tuning local large language models for data preprocessing. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 8754–8782, Miami, Florida, USA, November 2024a. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.497. URL https://aclanthology.org/2024.emnlp-main.497/.

Haochen Zhang, Yuyang Dong, Chuan Xiao, and Masafumi Oyamada. Large language models as data preprocessors, 2024b. URL https://arxiv.org/abs/2308.16361.

Haopeng Zhang, Hayate Iso, Sairam Gurajada, and Nikita Bhutani. Xatu: A fine-grained instruction-based benchmark for explainable text updates, 2024c. URL https://arxiv.org/abs/2309.11063.

Mian Zhang, Shujian Liu, Sixun Dong, Ming Yin, Yebowen Hu, Xun Wang, Simin Ma, Song Wang, Sathish Reddy Indurthi, Haoyun Deng, Zhiyu Zoey Chen, and Kaiqiang Song. Logicif: Towards complex logic instruction following, 2026. URL https://arxiv.org/abs/2508.09125.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models, 2023. URL https://arxiv.org/abs/2311.07911.

A LIMITATIONS

CDR-Bench is designed as a controlled diagnostic of the LLM-side procedural executor rather than an end-to-end evaluation of complete data-refinement systems. Its deterministic core is grounded in the Data-Juicer operator library, which provides reproducible execution traces and objective scoring but does not represent every refinement policy or implementation. We partly address this scope boundary through four semantic-extension domains with human-annotated references and through a code-generation-and-execution baseline. We do not comprehensively evaluate iterative tool selection, code

debugging, external verification, or runtime recovery. The natural-language instructions are generated across multiple styles with LLM assistance and screened for fidelity to the executable recipes. This design reduces dependence on a single prompt formulation, but the resulting instruction distribution may not capture the full diversity or ambiguity of requests written by real users. In addition, the balanced `KEEP`/ `DROP` construction is intended to remove majority-class shortcuts and should not be interpreted as modeling the naturally skewed retention rates of every deployment pipeline. The current evaluation also focuses on English text. Multilingual, multimodal, and interactive refinement remain important directions for future work. Finally, evaluating models across multiple tracks and instruction formulations entails substantial computational and API costs. Within our resource budget, we therefore cover a limited set of models and do not include all recent closed-source releases.

B Additional Background

Table 7 expands the comparison in Table 1. We distinguish the composition of multiple requirements from sequential execution in which operations consume the outputs of preceding steps. Paired order variants further require keeping the input and operator set fixed while varying the execution order. Intermediate references denote gold states at operation boundaries. A deterministic oracle denotes an executable reference procedure or rule-based correctness verifier, rather than merely a deterministic similarity metric. The symbols describe explicit benchmark coverage, not whether the underlying tasks could involve the corresponding capability.

Table 7: Full scope comparison with related benchmarks. ✓ denotes explicit coverage, △ partial coverage, and ✗ no explicit coverage under the definitions in Appendix B.

Benchmark	Input	Evaluation target	Compositional instructions	Sequential state updates	Paired order variants	Intermediate references	Deterministic oracle
Complex Instruction Following							
IFEval (Zhou et al., 2023)	Prompt	Output compliance	✓	✗	✗	✗	✓
FollowBench (Jiang et al., 2024)	Prompt	Output compliance	✓	✗	✗	✗	✗
InFoBench (Qin et al., 2024)	Prompt	Requirement compliance	✓	✗	✗	✗	✗
ComplexBench (Wen et al., 2024)	Prompt	Constraint composition	✓	✗	✗	✗	△
MOSAIC (Purpura et al., 2026)	Prompt	Constraint interaction	✓	✗	✗	✗	△
Task-Specific Data Processing							
TAB (Pilán et al., 2022)	Unstructured text	Anonymization	✗	✗	✗	✗	✗
EditEval (Dwivedi-Yu et al., 2022)	Unstructured text	Text editing	✗	✗	✗	✗	✗
InstrEditBench (Zeng et al., 2026)	Unstructured text	Text editing	✗	✗	✗	✗	✗
OpenPII-1M (Ai4Privacy, 2026)	Unstructured text	PII masking	✗	✗	✗	✗	✗
Data-Centric Agents							
AutoDCWorkflow (Li et al., 2025)	Tables	Workflow generation	✗	✗	✗	✗	△
DataGovBench (Liu et al., 2025)	Tables, text	Governance workflow	✓	✓	✗	✗	△
DCA-Bench (Huang et al., 2025)	Files	Dataset curation	✗	✗	✗	✗	✗
KramaBench (Lai et al., 2026)	Files	Data-to-insight workflow	✓	✓	✗	✗	△
DAComp (Lei et al., 2025)	Files	Data workflow	✓	✓	✗	✗	△
Procedure Following							
ProcBench (Fujisawa et al., 2024)	Symbols	Procedure execution	✓	✓	✗	✓	✓
LogicIFEval (Zhang et al., 2026)	Prompt	Logic simulation	✓	✓	✗	✗	✓
CDR-Bench (ours)	Unstructured text	Direct recipe execution	✓	✓	✓	✓	✓

C Benchmark Construction Details

C.1 Domain-Specific Operator Inventory

Table 8 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.

C.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

Table 8: 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).

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 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`.

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 $\mathcal{F}, \mathcal{R}$

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.

C.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 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.

C.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 9 defines each style and provides representative examples. The styles vary the discourse form of the request, such as concise

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 $v_k = \mathrm{stat}(f, S_k)$ 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**
    $\theta \leftarrow \mathrm{percentile}(V, \tau)$
  else
    $\theta \leftarrow \mathrm{percentile}(V, 1 - \tau)$
  end if
  for each record x **do**
    $O \leftarrow [\mathrm{execute}(x, r_s, \theta)$ for $s \in \{\text{front}, \text{mid}, \text{end}\}]$
    if $|\mathrm{unique}(O)| \geq 2$ **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

commands, casual requests, policy-like rules, and downstream use-case framing, while preserving the same operator sequence, execution order, and filtering semantics.

C.5 Benchmark Statistics

We provide detailed statistics of CDR-Bench below. Table 10 summarizes the overall task composition, input length distribution, recipe coverage, and instruction statistics. Table 11 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).

C.6 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.

D Experiment Details

D.1 Hyperparameters

We use non-thinking mode whenever supported by the model and serving interface. Claude Opus models are evaluated with low reasoning effort, as indicated by † in the result tables. All models are evaluated with temperature set to 0 and a maximum output length of 32,768 tokens. We use a

Table 9: 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.

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.

unified output schema across all models, requiring a `status` field in {`KEEP`, `DROP`} and a `clean_text` field. For RS@3, each recipe is evaluated under 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.

D.2 Full Results

Table 13 extends the main results with more models under the same evaluation protocol. The additional models exhibit similar execution gaps, with low Order-M success and substantially lower Order-F success at Mid and Post than at Pre.

D.3 Refinement Gain

Low exact recipe success leaves open how much useful transformation models achieve before falling short of the reference. Table 14 addresses this question using Refinement Gain (RG), which measures edit-distance reduction toward the reference relative to the original input. Gemma-4 achieves the highest RG on Agnostic-M and Order-M, while GPT-5.4 leads on Order-F. On Order-M, Gemma-4 attains higher RG than GPT-5.4 (59.32 vs. 52.58), despite lower RS@3 (8.04 vs. 13.99). Thus, the model that more often completes a recipe exactly is not necessarily the one that achieves greater average text improvement. RG provides a complementary view of transformation quality beyond binary completion.

Table 10: Overall statistics of CDR-Bench, including task composition, input characteristics, recipe coverage, and instruction diversity.

Statistics	Value
Total Tasks	
Total Tasks	3462 (100%)
Atomic Mapper Tasks	132 (3.8%)
Atomic Filter Tasks	42 (1.2%)
Order-Agnostic Mapper Tasks	497 (14.4%)
Mapper-Order Sensitivity Tasks	286 (8.3%)
Filter-Order Sensitivity Tasks	2505 (72.4%)
Input Text	
Avg. Input Length (chars)	2704.0
Median Input Length (chars)	650.0
Min / Max Input Length	27 / 9997
Short / Medium / Long	2398 / 664 / 400
Recipe Coverage	
Active Operators Covered	29
Family Anchors	16
Executable Recipe Templates	63
Materialized Variants	616
Avg. Recipe Length	4.23
Instruction	
Avg. Prompt Variants	10.4
Median Prompt Variants	11.0
Avg. Instruction Words	46.1

Table 11: Domain-level task and recipe coverage of CDR-Bench. Family anchors are mined from operator co-occurrence patterns, and executable recipe templates are selected from these families before materialization.

Domain	Tasks	Ratio	Anchors	Recipes
LaTeX Refinement (LR)	500	14.4%	2	9
RAG Preparation (RP)	521	15.1%	3	10
Privacy Redaction (PR)	1304	37.7%	5	23
Web Refinement (WR)	1137	32.8%	6	21
Total	3462	100%	16	63

Table 12: Failure-mode taxonomy used for analyzing model errors on CDR-Bench.

Failure Mode	Definition
Filter Error	Incorrect `KEEP`/`DROP` decision.
Missed Operator	Required operators are skipped.
Under-application	Residual artifacts remain after refinement.
Over-application	Valid content is removed or modified.
Wrong Order	Operators violate recipe ordering.
Formatting Drift	Surface formatting differs from reference.
Semantic Rewrite	Content is rewritten instead of refined.
Instruction Misread	Recipe constraint is misunderstood.
Parse Error	Output violates required schema.

Table 13: Full results on CDR-Bench, measured by recipe success (RS@3, %; higher is better). Models use non-thinking mode unless marked with †, which denotes low reasoning effort. Order-M reports the original and swapped orders. Order-F reports Pre, Mid, and Post filter placements. Best and second-best results in each column are **bolded** and underlined.

Model	Atomic (Single Operator)		Compositional (Multi-Operator Recipe)							
	Atomic-M	Atomic-F	Agnostic-M	Order-M			Order-F			
				Original	Swapped	Avg.	Pre	Mid	Post	Avg.
Closed-Source Models										
Qwen3.8-Max	28.79	**92.86**	21.93	10.49	12.59	11.54	55.33	23.11	15.33	31.26
Qwen3.6-Max	26.15	76.19	23.94	9.09	5.59	7.34	48.14	21.32	15.81	28.42
Claude Opus 4.6†	29.55	73.81	22.33	6.99	**17.48**	12.24	57.96	**33.17**	**23.95**	**38.36**
Claude Opus 4.5†	27.27	71.43	22.13	6.99	16.08	11.54	42.51	18.56	13.05	24.71
GPT-5.4	**30.30**	88.10	22.74	11.89	16.08	**13.99**	**62.04**	25.03	14.97	34.01
Open-Source Models										
Qwen3.6-27B	24.24	80.95	23.73	11.89	4.93	8.41	46.95	19.09	17.51	27.85
Qwen3.6-35B-A3B	18.18	78.05	20.40	8.45	5.59	7.02	50.98	12.17	12.89	25.35
DeepSeek-V4-Flash	24.62	76.19	23.94	6.99	2.80	4.90	39.57	11.75	13.67	21.66
DeepSeek-V4-Pro	22.73	76.19	24.55	8.39	6.29	7.34	31.38	15.09	14.61	20.36
Kimi-K3	**30.30**	80.95	24.55	12.59	5.59	9.09	43.47	20.96	15.33	26.59
Kimi-K2.6	29.23	76.19	24.14	**14.69**	4.20	9.44	35.25	16.55	14.39	22.06
GLM-5.2	25.00	80.95	24.95	13.99	4.20	9.09	32.10	19.64	17.37	23.03
GLM-5	26.15	73.81	24.80	11.89	9.09	10.49	33.93	22.78	17.75	24.82
Gemma-4-31B-IT	29.55	80.95	**25.55**	10.49	5.59	8.04	32.81	19.16	14.01	22.00
Llama-4-Scout-17B	16.92	62.50	13.59	2.82	4.93	3.87	22.41	4.47	3.99	10.29

D.4 Confidence Intervals

We report 95% confidence intervals for RS@3. We use 10,000 bootstrap resamples at the instance level for atomic and order-agnostic tasks, and at the group level for order-sensitive tasks.

Table 14: Refinement gain (RG; higher is better) on CDR-Bench, complementing the RS@3 results in Table 13. Best and second-best results in each column are **bolded** and underlined. † denotes low reasoning effort.

Model	Agnostic-M	Order-M	Order-F
Closed-Source Models			
Qwen3.8-Max	52.74	53.19	44.66
Qwen3.6-Max	50.81	48.90	40.38
Claude Opus 4.6†	48.06	50.42	47.03
Claude Opus 4.5†	49.39	51.00	41.17
GPT-5.4	53.77	52.58	**48.47**
Open-Source Models			
Qwen3.6-27B	42.39	41.88	33.29
Qwen3.6-35B-A3B	42.03	41.31	29.95
DeepSeek-V4-Flash	44.41	41.27	33.43
DeepSeek-V4-Pro	48.80	46.56	36.48
Kimi-K3	53.83	53.05	44.31
Kimi-K2.6	46.36	47.77	35.39
GLM-5.2	51.26	53.66	38.99
GLM-5	49.42	52.05	37.92
Gemma-4-31B-IT	**56.54**	**59.32**	43.19
Llama-4-Scout-17B	35.95	26.66	26.98

Table 15: RS@3 (%) with 95% percentile bootstrap confidence intervals for Atomic-M, Atomic-F, and Agnostic-M. Each cell reports estimate [lower, upper]. † denotes low reasoning effort.

Model	Atomic-M	Atomic-F	Agnostic-M
Closed-Source Models			
Qwen3.8-Max	28.79 [21.21, 36.36]	92.86 [83.33, 100.00]	21.93 [18.31, 25.55]
Qwen3.6-Max	26.15 [18.46, 33.85]	76.19 [61.90, 88.10]	23.94 [20.32, 27.77]
Claude Opus 4.6†	29.55 [21.97, 37.12]	73.81 [59.52, 85.71]	22.33 [18.71, 25.96]
Claude Opus 4.5†	27.27 [19.70, 34.85]	71.43 [57.14, 83.33]	22.13 [18.71, 25.96]
GPT-5.4	30.30 [22.73, 37.88]	88.10 [78.57, 97.62]	22.74 [19.11, 26.36]
Open-Source Models			
Qwen3.6-27B	24.24 [17.42, 31.82]	80.95 [69.05, 92.86]	23.73 [20.08, 27.59]
Qwen3.6-35B-A3B	18.18 [12.12, 25.00]	78.05 [65.85, 90.24]	20.40 [16.97, 24.04]
DeepSeek-V4-Flash	24.62 [17.69, 32.31]	76.19 [61.90, 88.10]	23.94 [20.12, 27.77]
DeepSeek-V4-Pro	22.73 [15.91, 30.30]	76.19 [61.90, 88.10]	24.55 [20.72, 28.37]
Kimi-K3	30.30 [22.73, 37.88]	80.95 [69.05, 92.86]	24.55 [20.93, 28.37]
Kimi-K2.6	29.23 [21.54, 36.92]	76.19 [61.90, 88.10]	24.14 [20.52, 27.97]
GLM-5.2	25.00 [18.18, 32.58]	80.95 [69.05, 92.86]	24.95 [21.13, 28.77]
GLM-5	26.15 [18.46, 33.85]	73.81 [59.52, 85.71]	24.80 [21.17, 28.63]
Gemma-4-31B-IT	29.55 [21.97, 37.12]	80.95 [69.05, 92.86]	25.55 [21.73, 29.58]
Llama-4-Scout-17B	16.92 [10.77, 23.08]	62.50 [47.50, 77.50]	13.59 [10.55, 16.63]

D.5 Additional Analysis

We further examine how execution reliability varies with recipe length, instruction style, thinking mode, and refinement domain.

Performance across Recipe Lengths Figure 5 shows that execution reliability generally decreases as recipes grow longer across all three compositional tracks. On Agnostic-M and Order-M, RS@3 approaches zero for all three evaluated models beyond five operations. Order-M exhibits a sharper decline, while Order-F degrades more gradually. Long recipes therefore remain difficult even when their operations are order-agnostic, with particularly low success when execution must also preserve a consequential operation order.

Effect of Instruction Styles Figure 6 compares the 11 instruction styles defined in Table 9. RS@K increases as more formulations are attempted, but the gap between atomic and compositional execution persists. Styles that explicitly enumerate operations, including *Step-by-Step* and *Brief*, achieve higher

Table 16: RS@3 (%) with 95% group-bootstrap confidence intervals for order-sensitive tasks. Each cell reports estimate [lower, upper]. Order-M pools evaluated mapper-order variants. Order-F reports each filter placement and their unweighted average, recomputed within each bootstrap resample. † denotes low reasoning effort.

Model	Order-M	Order-F Pre	Order-F Mid	Order-F Post	Order-F Avg.
Closed-Source Models					
Qwen3.8-Max	11.54 [7.69, 15.73]	55.33 [51.98, 58.56]	23.11 [20.24, 25.99]	15.33 [12.93, 17.84]	31.26 [29.26, 33.21]
Qwen3.6-Max	7.34 [4.55, 10.49]	48.14 [44.79, 51.62]	21.32 [18.56, 24.07]	15.81 [13.41, 18.32]	28.42 [26.31, 30.50]
Claude Opus 4.6†	12.24 [8.39, 16.43]	57.96 [54.61, 61.32]	33.17 [29.94, 36.53]	23.95 [21.08, 26.83]	38.36 [35.85, 40.88]
Claude Opus 4.5†	11.54 [7.69, 15.73]	42.51 [39.16, 45.87]	18.56 [15.93, 21.20]	13.05 [10.78, 15.33]	24.71 [22.63, 26.79]
GPT-5.4	13.99 [9.79, 18.53]	62.04 [58.80, 65.27]	25.03 [22.15, 28.02]	14.97 [12.57, 17.49]	34.01 [31.98, 36.13]
Open-Source Models					
Qwen3.6-27B	8.41 [4.95, 12.01]	46.95 [43.59, 50.30]	19.09 [16.45, 21.73]	17.51 [14.87, 20.14]	27.85 [25.89, 29.84]
Qwen3.6-35B-A3B	7.02 [4.21, 10.14]	50.98 [47.56, 54.40]	12.17 [10.00, 14.46]	12.89 [10.60, 15.18]	25.35 [23.58, 27.14]
DeepSeek-V4-Flash	4.90 [2.80, 7.34]	39.57 [36.21, 42.93]	11.75 [9.59, 13.91]	13.67 [11.39, 16.07]	21.66 [19.94, 23.42]
DeepSeek-V4-Pro	7.34 [4.55, 10.49]	31.38 [28.26, 34.49]	15.09 [12.69, 17.60]	14.61 [12.34, 17.13]	20.36 [18.40, 22.36]
Kimi-K3	9.09 [5.94, 12.59]	43.47 [40.12, 46.83]	20.96 [18.32, 23.71]	15.33 [12.93, 17.84]	26.59 [24.51, 28.66]
Kimi-K2.6	9.44 [6.28, 12.94]	35.25 [32.01, 38.49]	16.55 [14.03, 19.07]	14.39 [11.99, 16.79]	22.06 [20.14, 24.10]
GLM-5.2	9.09 [5.94, 12.59]	32.10 [28.98, 35.33]	19.64 [17.01, 22.28]	17.37 [14.85, 20.00]	23.03 [21.00, 25.11]
GLM-5	10.49 [6.99, 14.34]	33.93 [30.70, 37.17]	22.78 [19.90, 25.66]	17.75 [15.11, 20.38]	24.82 [22.78, 26.82]
Gemma-4-31B-IT	8.04 [4.90, 11.19]	32.81 [29.70, 36.05]	19.16 [16.53, 21.92]	14.01 [11.74, 16.41]	22.00 [20.04, 24.07]
Llama-4-Scout-17B	3.87 [1.76, 6.29]	22.41 [19.64, 25.30]	4.47 [3.14, 5.92]	3.99 [2.66, 5.44]	10.29 [9.01, 11.64]

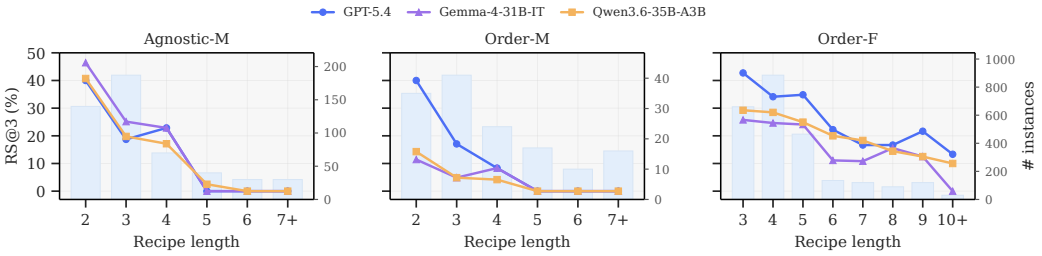

Figure 5: RS@3 across recipe lengths. Lines show results for representative models, and light-blue bars indicate the number of instances in each length bucket.

mean RS on transformation-heavy tracks than outcome-oriented styles such as *Goal-First* and *Scenario Story*. On Order-M, however, performance remains below 10% across all styles. Clearer instruction wording thus helps some tasks but provides limited relief for order-sensitive recipe execution.

Effect of Thinking Mode Figure 7 shows that thinking mode improves RS@3 across all evaluated compositional tracks for both models. The largest gains occur on Order-F, where RS@3 increases from 25.35% to 44.87% for Qwen3.6-35B-A3B and from 27.85% to 45.71% for Qwen3.6-27B, yielding gains of 19.52 pp and 17.86 pp, respectively. Improvements on Order-M are more modest, at 3.12 pp and 3.83 pp. These results suggest that additional reasoning helps with state-dependent filtering, while execution-order sensitivity remains challenging.

Performance across Domains Figure 8 shows lower compositional than atomic performance in most domains. PR achieves the highest atomic RS@3 and includes operations targeting localized patterns such as emails and URLs. WR and LR, which include operations over broader document structure, score lower in isolation. Despite its stronger atomic performance, PR exhibits the largest atomic-to-compositional gaps, reaching 43.1 pp for Gemma-4 and 38.1 pp for Qwen3.6-35B-A3B. Reliable execution of individual, localized transformations therefore does not ensure reliable execution of recipes combining them.

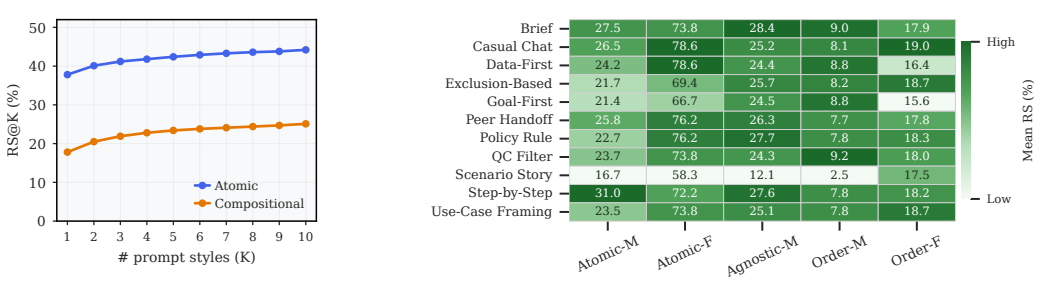

Figure 6: RS@K curves by task family (left) and mean RS across tracks and instruction styles (right).

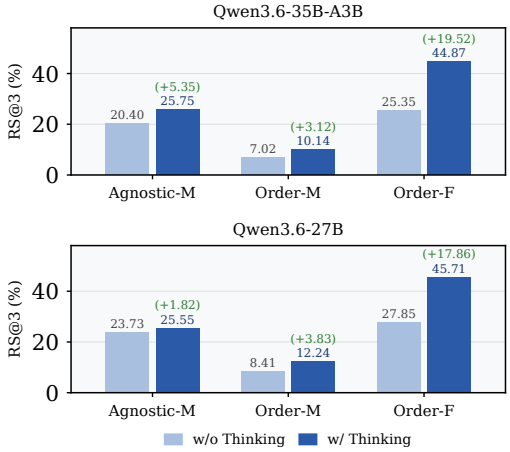

Figure 7: Effect of thinking mode on RS@3 across compositional tracks.

D.6 Oracle Filtering Diagnostics

We evaluate Gemma-4-31B-IT and Qwen3.6-35B-A3B on 142 paired Order-F groups. Gold State supplies the correct intermediate text obtained by executing the recipe operations preceding the filter. Gold Statistic supplies the true statistic computed from that text. Both conditions require only a KEEP/DROP prediction. Table 17 compares their mean status accuracy with full-recipe execution over three instruction formulations.

D.7 Code-Assisted Execution

Code-run provides the model with the recipe instruction and input text and requests a Python function `transform(input_text)`. The prompt requires preserving operation order, stopping upon filter rejection, and using only the Python standard library. We execute the generated code in a separate Python process and evaluate its returned status and text with the recipe scorer. Each formulation uses a single code-generation response without execution-feedback-based revision. We report RS@3 over three sampled instruction styles with sampling seed 0.

D.8 Failure Mode Analysis

We analyze failed predictions from Gemma-4-31B-IT, GPT-5.4, and Qwen3.6-35B-A3B using the categories defined in Table 12. Each prediction receives one primary label. When multiple symptoms occur, a fixed priority assigns status errors before text-level deviations. Figure 9 summarizes their distribution across models and tracks.

Incorrect filtering decisions. Incorrect KEEP/DROP decisions account for 39.3% of analyzed failures. This category includes errors arising from an incorrect intermediate text, an inaccurate

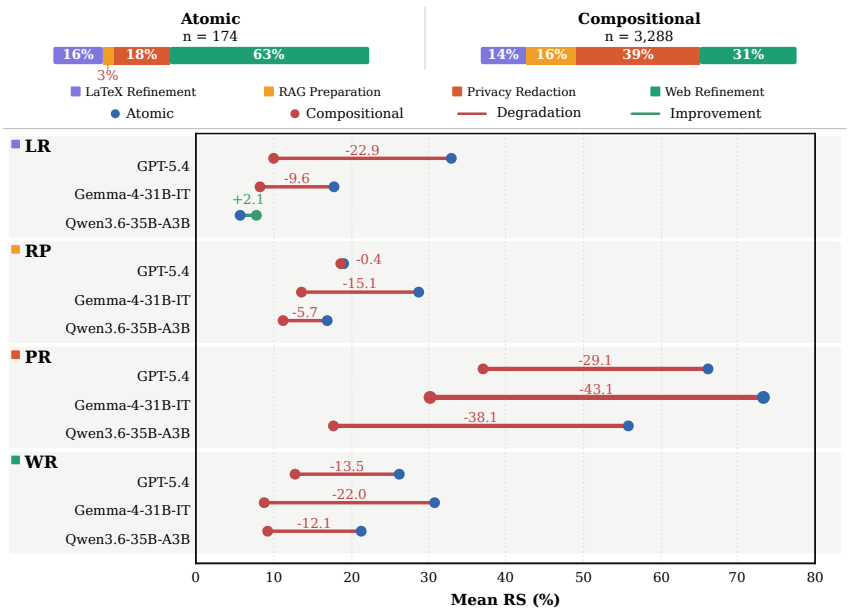

Figure 8: Atomic and compositional RS across domains, with per-model degradation gaps (pp).

Table 17: Mean filter status accuracy (%) on 142 paired Order-F groups. Each placement contains the same groups. Results average three formulations rather than taking success under any formulation.

Model	Position	Full	Gold State	Gold Statistic
Gemma-4-31B-IT	Pre	23.0	23.2	100.0
	Mid	66.4	74.6	99.8
	Post	85.9	90.8	100.0
Qwen3.6-35B-A3B	Pre	45.1	47.9	100.0
	Mid	52.3	62.9	100.0
	Post	55.6	70.7	99.8

statistic, or an incorrect application of the filtering rule. The oracle diagnostics in Appendix D.6 show that providing correct text improves decision accuracy, while supplying its true statistic raises accuracy to near-perfect levels. These results distinguish the difficulty of obtaining the information needed for filtering from applying the rule once that information is available.

Incomplete or unintended text changes. Formatting deviations account for 28.4% of analyzed failures, followed by skipped operations at 11.7% and partially applied operations at 9.5%. Models may therefore produce plausible refinements while changing formatting inconsistently with the reference, omitting required transformations, or leaving parts of the input insufficiently processed. Other categories capture excessive deletion and semantic rewriting beyond the requested recipe.

Models may favor the original operation order. We construct swapped recipes by reordering operations in an original recipe and retaining pairs with different reference outputs. For incorrect predictions under swapped instructions, we compare edit distance to both references. Table 4 shows that 59.4% are closer to the original-order reference, compared with 38.6% closer to the requested swapped-order reference. This tendency is strongest for Gemma-4 at 68.0%, versus 55.4% for GPT-5.4 and 54.6% for Qwen. The asymmetry suggests that models may follow the original processing order instead of faithfully applying the requested permutation.

Models may continue beyond the filter checkpoint. Figure 4 compares outputs on ground-truth DROP cases with the correct stopping state t_{stop} and the full-execution state t_{full}, obtained by continuing the remaining operations despite rejection. The horizontal axis reports the difference

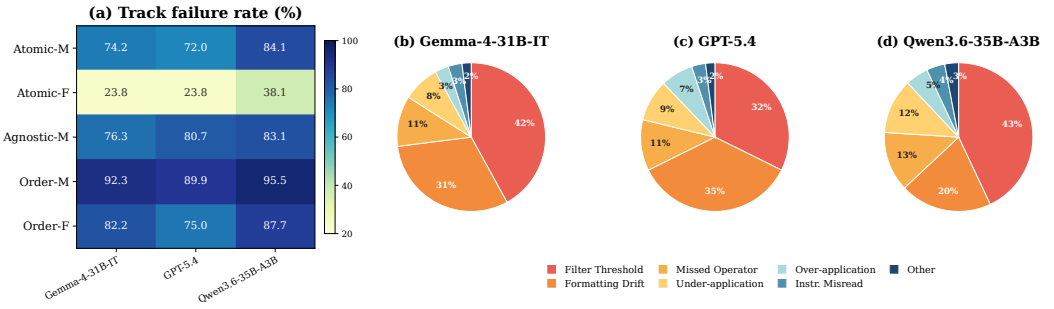

Figure 9: Failure analysis across representative models. (a) Per-prediction failure rates without cross-style aggregation. (b)–(d) Category distributions over failed predictions. The label "Filter Threshold Error" denotes incorrect KEEP/DROP decisions, without identifying their underlying cause.

between the fractions closer to these two references, using valid predictions whose stopping and full-execution references differ. The vertical axis reports mean RS over all evaluated outputs at the corresponding placement. In Pre-DROP, many models produce more outputs closer to t_{full}, even though rejection should precede any transformation. In Mid-DROP, this tendency persists for several models, whose outputs favor completing the recipe over retaining the intermediate text at rejection. Together, these patterns suggest a gap in procedural faithfulness, with models potentially completing transformations before making the filtering decision rather than applying the filter at its prescribed position and stopping upon rejection.

E Semantic Generalization Details

E.1 Motivation and Design

The main evaluation tracks rely on rule-based Data-Juicer operators, which provide deterministic references but naturally invite the question of whether the observed trends extend beyond operator replay. To complement the main benchmark, we therefore evaluate four semantic-extension domains 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, hallucination handling, semantic category tagging, and scoring under semantic rubrics, rather than deterministic rule execution. (2) Ground truth comes from human-annotated datasets rather than operator replay. (3) The tasks involve higher-level semantic judgments that are 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. The semantic evaluation uses a partially different model pool, as listed in the domain-specific result tables. We compare atomic and compositional performance within each model to examine whether the observed pattern also appears in semantic refinement.

E.2 Semantic Domains

For each semantic domain, we construct an *Atomic* track that evaluates individual semantic operations and a *Compositional* track that requires combining the corresponding operations in a single output. Unless otherwise specified, we report the same RS@3 convention used in the main benchmark. Each instance is prompted with three fixed styles: direct, imperative checklist, and application context. A prediction is counted as successful if any of the three prompted outputs matches the reference under the task-specific scorer.

For text-producing tasks, such as PII redaction and hallucination correction, we use normalized exact match over the required output text. For structured-output tasks, such as rubric scoring and safety tagging, we require exact matching of the requested JSON fields. Atomic scores are computed

separately for each semantic subtask and then macro-averaged within a domain. Compositional scores are computed on the paired full-output task. We define the composition gap uniformly as

$$\text{Gap} = \text{RS@3}_{\text{atom}} - \text{RS@3}_{\text{comp}}. \tag{3}$$

This mirrors the main benchmark's atomic-to-compositional comparison while keeping each semantic operation equally weighted within its domain.

PII Semantic Redaction We use the AI4Privacy PII masking dataset[2] (Ai4Privacy Community, 2023b). We select 500 English samples with at least two distinct PII category groups through stratified sampling. We aggregate the original PII labels 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 redaction for each group in isolation using `[LABEL_N]` placeholders, while the *Compositional* track requires redacting all present groups simultaneously. Ground truth is the programmatic redaction output derived from dataset annotations.

Hallucination Detection and Correction We use a balanced subset adapted from FAVA (Mishra et al., 2024) with 300 samples. The dataset 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 subtasks: (1) *Detection*, which determines whether the text contains hallucinations; (2) *Span Extraction*, which identifies hallucinated spans; (3) *Type Classification*, which classifies the hallucination types present; and (4) *Correction*, which produces corrected text with hallucinated content removed or replaced. The *Compositional* track requires executing the full processing pipeline in a single pass. We evaluate structured fields with exact JSON-field matching and correction outputs with normalized text exact match.

Safety Tagging We use Aegis-AI-Content-Safety-Dataset-2.0 (Ghosh et al., 2025), a content-safety moderation dataset with user prompts, assistant responses, binary safety labels, and violated-category annotations. We sample 300 prompt-response pairs from the test split after filtering to examples with both prompt and response labels, balancing the sample as evenly as possible across observed prompt-response label pairs. The *Atomic* track decomposes safety tagging into three subtasks: (1) *Prompt Label*, which classifies the user prompt as `safe` or `unsafe`; (2) *Response Label*, which classifies the assistant response as `safe` or `unsafe`; and (3) *Violated Categories*, which identifies the relevant safety-risk categories. The *Compositional* track requires producing all three fields jointly in one JSON object, testing whether models can compose related moderation decisions into a consistent structured output.

Rubric Scoring We use HelpSteer2 (Wang et al., 2024), a response-quality dataset of prompt-response pairs annotated along five rubric dimensions: *helpfulness*, *correctness*, *coherence*, *complexity*, and *verbosity*. Each dimension is labeled on a 0–4 ordinal scale. We sample 300 prompt-response pairs from the validation split with a fixed random seed. The *Atomic* track evaluates one rubric dimension at a time, requiring the model to output a JSON object containing only the requested score. The *Compositional* track requires predicting all five rubric scores jointly in a single JSON object.

E.3 Results and Analysis

Tables 18–21 and Figure 10 report the full results of the semantic extension. Averaged over models and domains, compositional RS@3 is 30.8%, compared with an atomic average of 58.8%. Atomic scores measure individual requirements, whereas compositional success requires satisfying all requirements within one output.

PII redaction. Contact information is the easiest atomic group across models, whereas identification, location, and person fields are consistently the hardest, reflecting the greater contextual

[2]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.

Table 18: PII semantic-domain results (RS@3%). Avg denotes the macro-average over atomic subtasks; Comp denotes the compositional task; Gap is Avg − Comp.

Model	Cont.	Loc.	Temp.	Ident.	Person	Avg	Comp	Gap
Claude 4.7 Opus	90.1	70.6	83.8	63.2	71.9	75.9	50.6	25.3
Claude 4.6 Sonnet	84.6	68.3	74.4	57.1	68.1	70.5	47.2	23.3
Qwen3.6-Max	94.5	65.4	86.9	66.1	79.7	78.5	51.0	27.5
GPT-5.5	89.7	67.6	76.9	66.1	57.7	71.6	51.8	19.8
DeepSeek-V4-Flash	92.9	62.7	77.5	49.6	63.8	69.3	39.4	29.9
Kimi-K2.6	91.3	65.0	83.8	59.3	59.5	71.8	34.2	37.6
GLM-5	94.1	60.8	80.0	64.3	73.7	74.6	39.0	35.6
Gemma-4	88.1	62.7	84.4	62.9	77.2	75.1	45.8	29.3
Qwen3.6-27B	83.8	55.9	71.2	59.6	51.6	64.4	40.2	24.2
Qwen3.6-35B-A3B	85.0	50.3	73.8	63.2	46.8	63.8	25.0	38.8

Table 19: Hallucination semantic-domain results (RS@3%). Avg denotes the macro-average over atomic subtasks; Comp denotes the compositional task; Gap is Avg − Comp.

Model	Detect.	Span.	Type.	Correct.	Avg	Comp	Gap
Claude 4.7 Opus	42.7	12.3	20.3	29.0	26.1	24.7	1.4
Claude 4.6 Sonnet	41.3	17.3	14.3	29.7	25.7	24.7	1.0
Qwen3.6-Max	43.0	22.7	33.0	35.3	33.5	27.7	5.8
GPT-5.5	40.3	15.3	21.7	21.0	24.6	21.0	3.6
DeepSeek-V4-Flash	53.7	33.3	32.0	48.0	41.8	36.3	5.4
Kimi-K2.6	44.7	12.7	22.7	45.0	31.2	23.3	7.9
GLM-5	47.7	32.0	31.7	37.0	37.1	29.0	8.1
Gemma-4	43.7	26.0	37.3	29.7	34.2	29.3	4.8
Qwen3.6-27B	51.0	26.7	40.3	32.7	37.7	35.0	2.7
Qwen3.6-35B-A3B	51.7	20.0	33.7	38.3	35.9	37.0	-1.1

Table 20: Safety-tagging semantic-domain results (RS@3%).

Model	Prom.	Resp.	Cate.	Avg	Comp	Gap
Claude 4.7 Opus	84.0	89.0	58.0	77.0	48.0	29.0
Claude 4.6 Sonnet	83.7	87.3	54.0	75.0	46.0	29.0
Qwen3.6-Max	88.0	86.3	57.7	77.3	45.0	32.3
GPT-5.5	86.7	74.3	55.0	72.0	46.0	26.0
DeepSeek-V4-Flash	89.7	82.3	56.3	76.1	49.0	27.1
Kimi-K2.6	80.0	85.3	57.3	74.2	49.3	24.9
GLM-5	86.0	85.3	55.3	75.6	50.3	25.2
Gemma-4	83.0	83.7	50.7	72.4	44.7	27.8
Qwen3.6-27B	88.0	91.7	58.3	79.3	52.7	26.7
Qwen3.6-35B-A3B	88.3	87.3	54.7	76.8	48.3	28.4

Table 21: Rubric-scoring semantic-domain results (RS@3%).

Model	Help.	Correct.	Coher.	Complex.	Verb.	Avg	Comp	Gap
Claude 4.7 Opus	33.0	36.7	64.0	62.0	60.7	51.3	7.7	43.6
Claude 4.6 Sonnet	35.0	34.0	49.3	52.7	42.7	42.7	2.7	40.1
Qwen3.6-Max	53.7	51.0	71.0	60.0	72.0	61.5	6.3	55.2
GPT-5.5	39.3	44.0	68.7	55.0	44.0	50.2	5.7	44.5
DeepSeek-V4-Flash	57.0	58.3	78.7	69.3	60.3	64.7	4.0	60.7
Kimi-K2.6	38.3	51.3	59.3	72.3	38.0	51.9	1.3	50.5
GLM-5	48.7	54.0	75.7	38.0	27.3	48.7	2.3	46.4
Gemma-4	49.0	48.7	72.3	38.7	68.0	55.3	5.3	50.0
Qwen3.6-27B	53.0	52.0	74.3	71.7	61.0	62.4	3.7	58.7
Qwen3.6-35B-A3B	53.7	56.7	66.0	76.0	63.3	63.1	2.0	61.1

reasoning they require. Composition then imposes a clear additional cost, with gaps ranging from 19.8 to 38.8 percentage points: strong per-group redaction does not guarantee that a model can satisfy all groups simultaneously in one exact output. This mirrors the transformation setting of the main benchmark, where a model must apply several localized edits while preserving all unaffected content verbatim.

Hallucination processing. Unlike the other domains, whose atomic subtasks are largely parallel, hallucination processing decomposes into a sequence of increasingly fine-grained operations: detecting whether an error is present (coarsest), localizing the offending span, classifying its type, and producing a correction (finest). The subtask scores follow this granularity ordering, with detection as the ceiling and span localization, type classification, and correction all markedly lower. The compositional score still falls below the atomic average for nearly all models, but the additional gap is comparatively small (-1.1 to 8.1 pp): because the fine-grained subtasks are already hard, combining them adds little on top of an already low base. The dominant difficulty in this domain is thus the fine-grained processing itself rather than the act of merging outputs.

Safety tagging. Binary safety labels are far easier than fine-grained category tagging. Prompt- and response-label classification frequently exceeds 80%, whereas violated-category tagging is the clear bottleneck, and the compositional task requiring all three fields jointly yields gaps of 24.9 to 32.3 percentage points. Failures here are not about distinguishing safe from unsafe content, but about producing a complete, internally consistent annotation that also names the correct risk categories.

Rubric scoring. Rubric scoring shows the largest atomic-to-compositional gaps, ranging from 40.1 to 61.1 percentage points. Several systems achieve atomic averages above 60%, with coherence often among the stronger dimensions, yet compositional scores range from only 1.3% to 7.7%. Higher atomic averages do not consistently translate into higher compositional success. This pattern highlights the difficulty of maintaining scoring accuracy across an entire rubric. Exact success requires all five ordinal judgments to match their references within one output, making the result sensitive to residual errors in any dimension. The large gaps therefore reflect the demands of producing a complete, accurate score profile, although they do not alone establish whether joint prediction reduces accuracy on individual dimensions.

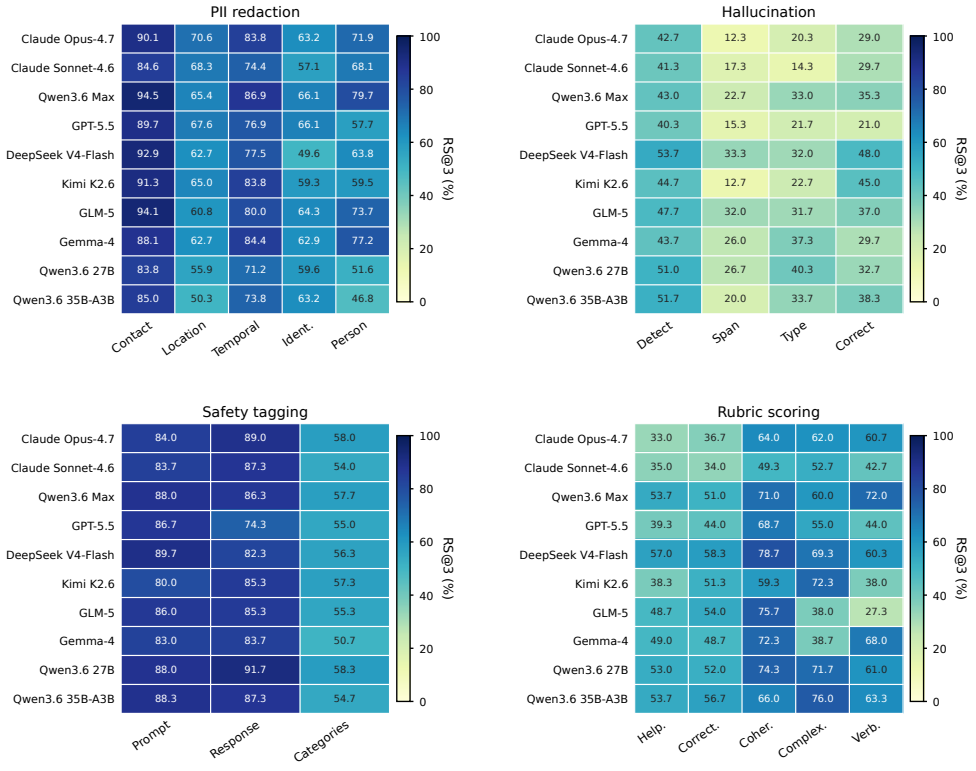

Figure 10: Atomic subtask RS@3 heatmaps for the four semantic domains. The heatmaps show that atomic difficulty is itself structured: PII varies by entity group, hallucination follows a coarse-to-fine hierarchy, rubric scoring varies by judgment dimension, and safety tagging separates easier binary labels from harder violated-category prediction.

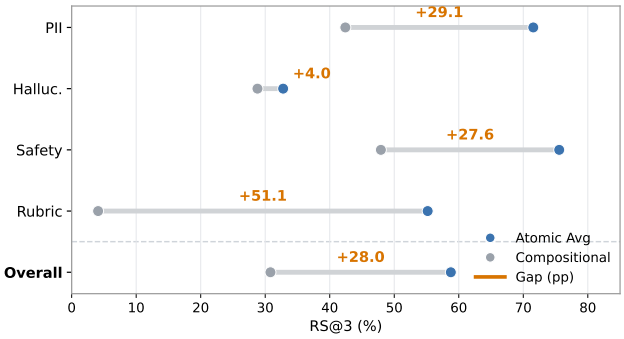

Figure 11: Atomic-to-compositional performance drop across the four semantic domains. Blue points show Atomic Avg RS@3 macro-averaged over atomic subtasks and models, gray points show Compositional RS@3, and orange labels report the gap.

Domain-dependent composition patterns. The results reveal different sources of difficulty across semantic domains. PII redaction requires complete coverage of relevant text spans, while safety tagging combines binary decisions with finer category assignments. Missing a span or selecting an incorrect category can invalidate an otherwise partially correct output. Rubric scoring imposes a different demand, requiring precise agreement across five ordinal judgments. Its low compositional scores show how difficult it is to produce an entire score profile accurately, even when some dimensions are scored reliably. In hallucination processing, span extraction, type classification, and correction are already difficult individually, leaving a smaller additional drop when combined.

These patterns suggest that improving semantic refinement requires addressing domain-specific weaknesses in coverage, category selection, and judgment precision alongside the coordination of multiple operations.

E.4 Key Findings

- **Semantic composition exposes a broad reliability gap beyond the main synthetic tracks.** Across the four semantic domains, models are consistently better at isolated semantic operations than at the corresponding composed task. Averaged over models and domains, Atomic Avg RS@3 is 58.8%, while Compositional RS@3 is 30.8%, giving an overall semantic composition gap of 28.0 percentage points. This indicates that the core CDR-Bench observation extends to adapted real-world semantic tasks.
- **The gap is governed by task structure rather than domain label alone.** Large gaps appear when success requires several independently meaningful decisions to be correct in one final output, as in PII redaction, safety tagging, and especially rubric scoring. In contrast, hallucination processing has a much smaller additional gap because the hardest atomic operations are already fine-grained diagnosis and correction.
- **Exact structured composition is especially brittle.** Rubric scoring and safety tagging show that models can often make individual judgments, but fail when several fields must be jointly correct under a fixed output schema. This suggests that semantic competence on single labels or dimensions does not directly translate into reliable multi-field annotation.
- **Atomic performance alone can overstate deployable capability.** Several domains contain atomic subtasks with high RS@3, yet their compositional scores remain much lower. Reporting only isolated operations would therefore obscure the practical failure mode that appears when a model must satisfy all requirements simultaneously.

Overall, the semantic extension supports the same central message as the main benchmark: reliable data refinement requires not only recognizing individual operations, but executing all required operations jointly under a strict output contract.

F Targeted Post-training for Recipe Execution

F.1 Training Setup and Data Curriculum

We initialize Juicer from Qwen3.6-35B-A3B. We organize its SFT curriculum into two phases: general operator learning and recipe-oriented adaptation.

Phase I: general operator learning. We first train on 550K instruction-response examples with 72 distinct operator labels in the dataset metadata, including text mapping, filtering, PII processing, and semantic annotation. This initial corpus is not organized as CDR-Bench recipes. It establishes broad operator coverage before specializing the model to explicit recipe and output contracts.

Phase II: recipe-oriented adaptation. Recipe-oriented supervision covers the five core CDR-Bench tracks and four semantic tasks. Training emphasizes faithful operator composition, execution order, filtering at the prescribed intermediate state, and adherence to task-specific output formats. We compare three training configurations. *CDR-only SFT* applies recipe-oriented SFT directly to the base model. *Recipe-adapted SFT* first completes broad operator learning in Phase I and then undergoes recipe-oriented adaptation. Continued from Recipe-adapted SFT, we train the model with targeted supervision for exact filtering and order-sensitive execution, and further with instruction-following reinforcement learning to obtain the final *Juicer* model.

F.2 Results and Analysis

As shown in Table 22, compared with the Qwen3.6-35B-A3B base model, Juicer improves Atomic-M by 33.34 percentage points, Atomic-F by 12.43, Agnostic-M by 6.16, Order-M by 24.8, and Order-F by 16.73. Notably, it exceeds GPT-5.4 across all core tracks. Nevertheless, its Order-F Mid/Post scores (29.10/21.56) remain below Claude Opus 4.6 (33.17/23.95). The five-track macro-average

Table 22: Core recipe success (RS@3, %). Training configurations are defined in Appendix F.1. Best and second-best values among the displayed rows are bolded and underlined. All Juicer-series results are development-set evaluations.

Model	Atomic (Single Operator)		Compositional (Multi-Operator Recipe)							
	Atomic-M	Atomic-F	Agnostic-M	Order-M			Order-F			
				Original	Swapped	Avg.	Pre	Mid	Post	Avg.
Claude Opus 4.6	29.55	73.81	22.33	6.99	17.48	12.24	57.96	**33.17**	**23.95**	38.36
Qwen3.6-Max	26.15	76.19	23.94	9.09	5.59	7.34	48.14	21.32	15.81	28.42
GPT-5.4	30.30	88.10	22.74	11.89	16.08	13.99	62.04	25.03	14.97	34.01
Qwen3.6-35B-A3B	18.18	78.05	20.40	8.45	5.59	7.02	50.98	12.17	12.89	25.35
+ CDR-only SFT	39.39	78.57	24.55	15.38	9.79	12.59	16.65	28.86	20.72	22.08
+ Recipe-adapted SFT	41.67	80.95	26.16	20.28	13.99	17.13	48.98	27.19	17.60	31.26
Juicer	**51.52**	**90.48**	**26.56**	**41.96**	**21.68**	**31.82**	**75.57**	29.10	21.56	**42.08**

Table 23: Local training trajectory and execution diagnostics (%). RG measures text refinement.

Checkpoint	Core Avg. RS@3	Agnostic-M RG	Order-M RG	Order-F RG
Qwen3.6-35B-A3B	29.80	42.03	41.31	29.95
+ CDR-only SFT	35.44	67.49	64.82	48.15
+ Recipe-adapted SFT	39.43	69.41	68.12	50.49
Juicer	**48.49**	**75.07**	**80.58**	**61.05**

increases from 29.80% to 48.49% (+18.69 pp), as presented in Table 23. Training the recipe mixture directly from the base (CDR-only) yields 35.44 Core Avg., versus 39.43 with the prior curriculum. This supports the utility of prior operator learning and adaptation, but does not disentangle their effects from additional training data and compute. Table 24 reports Juicer's semantic-extension results. Compared with the base model, compositional RS@3 increases up to 27 pp for PII and rubric scoring, while safety changes by -0.63 pp. Compared with Recipe-adapted SFT, PII and safety compositional scores decrease by 2.0 and 0.66 pp, respectively. Thus, better exact recipe execution does not eliminate semantic composition failures or retention trade-offs.

G Case Study of Jev on CDR-Bench

Jev is a specialized decision model that evaluates supplied text through constrained choices and probability judgments (Almeida, 2026). We evaluate it separately on selected CDR-Bench tasks whose outputs can be expressed as discrete decisions, including filtering, rubric scoring, hallucination detection and type classification, and safety tagging. The aim is to characterize Jev's data-refinement capability, locate its performance relative to the previously evaluated general-purpose models, and identify the error patterns behind its strengths and weaknesses.

Table 24: Semantic-extension results (RS@3, %). A is the macro-average over atomic subtasks within a domain; C is compositional success.

Model	PII		Hallucination		Safety		Rubric	
	A	C	A	C	A	C	A	C
Claude 4.7 Opus	75.9	50.6	26.1	24.7	77.0	48.0	51.3	7.7
Qwen3.6-Max	78.5	51.0	33.5	27.7	**77.3**	45.0	61.5	6.3
GPT-5.5	71.6	51.8	24.6	21.0	72.0	46.0	50.2	5.7
DeepSeek-V4-Flash	69.3	39.4	**41.8**	36.3	76.1	**49.0**	**64.7**	4.0
Qwen3.6-35B-A3B	63.8	25.0	35.9	**37.0**	76.8	48.3	63.1	2.0
+ Recipe-adapted SFT	**85.05**	**54.0**	34.67	36.67	75.11	48.33	62.47	17.67
Juicer	82.58	52.0	38.67	**37.0**	76.0	47.67	63.13	**18.33**

G.1 Adaptation for Jev

We evaluate jev-1.13.0 through its native API, reusing the original inputs, task requirements, prompt variants, references, and scorer. Task definitions follow Appendix E. The adaptation adds field-specific instructions and fixed choices. Labels and integer scores use `Choice`, while category membership uses `Noul` with a probability threshold of 0.5. The adapter assembles the required output from judgments for the same prompt variant. The comparisons therefore assess task performance with different interfaces, as Jev receives external support for decomposition and output assembly. All selected instances remain in the scoring denominator, including 31 hallucination-task instances without three valid outputs, which are counted as unsuccessful. Text transformation, Order-M, Order-F, and the full hallucination processing pipeline are outside this evaluation.

G.2 Performance Relative to General-Purpose Models

Table 25: Jev and selected reference models on the evaluated tasks (RS@3%). Baseline scores are reused from Tables 19, 20, and 21. Task abbreviations follow those tables. Jev uses constrained field judgments and external output assembly. The baselines generate complete outputs directly, so the scores provide references under different interfaces. Safety Cate. and Comp additionally involve the category-schema difference described below.

Model	**Rubric**		**Hallu.**		**Safety**			
	Avg	Comp	Detect.	Type.	Prom.	Resp.	Cate.	Comp
DeepSeek-V4-Flash	64.7	4.0	53.7	32.0	89.7	82.3	56.3	49.0
Gemma-4	55.3	5.3	43.7	37.3	83.0	83.7	50.7	44.7
Qwen3.6-35B-A3B	63.1	2.0	51.7	33.7	88.3	87.3	54.7	48.3
Jev	48.0	3.3	34.0	4.0	86.7	84.0	49.7	46.0

Table 25 places Jev alongside three reference models from the earlier evaluation. Its relative position varies substantially by task. Safety labels are close to Gemma-4 and Qwen3.6-35B-A3B, while rubric scoring and hallucination judgments are weaker. This profile supports a task-specific assessment of Jev rather than assigning it the overall capability level of a single general-purpose model.

Safety tagging is Jev's most competitive semantic task. Jev achieves 86.7% and 84.0% on prompt and response labels. These scores exceed Gemma-4's 83.0% and 83.7% and are within 3.3 pp of Qwen3.6-35B-A3B on both subtasks. Its compositional score of 46.0% also lies between Gemma-4 at 44.7% and Qwen3.6-35B-A3B at 48.3%. However, category tagging reaches only 49.7%, below the range of 50.7% to 58.3% across the ten baselines in Table 20. Jev's strongest relative performance is therefore in binary safety decisions, with less competitive category assignment. The category comparison also reflects the schema difference noted below.

Rubric scoring is near the lower end of the baseline range. Jev's atomic average of 48.0% is below nine of the ten baselines in Table 21, whose scores range from 42.7% to 64.7%. Its compositional score of 3.3% lies within their range of 1.3% to 7.7%. The atomic average conceals a marked imbalance between dimensions. Coherence reaches 72.0%, close to Gemma-4 at 72.3%, whereas verbosity reaches only 20.0%, below every baseline in that table. Thus, Jev's quality-scoring capability depends strongly on the requested dimension, with verbosity standing out as a particular weakness.

Hallucination judgments fall below the reported baselines. Jev achieves 34.0% on detection and 4.0% on type classification. Both are below all ten baselines in Table 19, which range from 40.3% to 53.7% for detection and from 14.3% to 40.3% for types. Restricting Jev's evaluation to instances with three valid outputs raises these scores only to 37.1% and 4.1%. The deficit therefore persists after excluding incomplete outputs and is especially pronounced for identifying the complete type set.

Filtering succeeds on most selected instances. Jev succeeds on 37 of 42 Atomic-F instances (88.1%). Its five errors are concentrated in average line length, maximum line length, and alphanumeric filtering, with three, one, and one failures, respectively. The adapter copies the input as the returned text, so this score measures filtering decisions with programmatic text preservation. It cannot be placed directly in the main Atomic-F ranking, where the baselines must also generate the returned text.

For safety categories, the adapter supplies 20 choices, while the inherited task requirements list 13 and contain two alternative spellings. Nine of 300 references contain a category outside the original list after accounting for those spellings. Category and compositional safety scores therefore also reflect a difference in the supplied schema. The comparisons above describe observed scores under these adaptations and do not establish statistical equivalence between models.

G.3 Error Patterns and Calibration

We next examine which behaviors account for Jev's performance profile. These diagnostics use the first prompt variant and report accuracy or other metrics rather than RS@3. They characterize Jev's outputs without assuming that the reference models share the same errors.

Response safety favors recall over precision. Jev identifies 91 of 100 unsafe responses but also marks 52 of 200 safe responses as unsafe, giving 91.00% unsafe recall and 63.64% precision. This shows a tendency to over-flag safe responses despite its competitive RS@3. Compositional accuracy also varies across the three equally sized scenario groups. It is 77% when both prompt and response are safe, 21% when the prompt is unsafe but the response is safe, and 31% when both are unsafe. In the mixed group, category accuracy is only 28%. Safe responses to unsafe prompts are consequently a particularly weak case for complete annotation, whose category reference covers the whole sample.

Rubric scores show concentrated prediction biases. Jev assigns a coherence score of 4 to 277 of 300 samples. Its 71.33% accuracy only slightly exceeds the 69.00% obtained by always predicting 4. The apparently strong coherence score therefore provides limited evidence of discrimination across score levels. Verbosity shows a different bias. Jev predicts 3 or 4 for 244 samples even though 199 references have a score of 2, overestimating verbosity by 1.11 points on average on the scale from 0 to 4. This distributional mismatch helps explain the low exact-match score for that dimension.

Hallucination counts and types err in opposite directions. Among 283 valid detection outputs, Jev predicts a count of zero in 227 cases. The presence label contradicts whether the count is positive in 107 cases (37.81%). Type classification instead over-selects categories, with a mean predicted set size of 3.70 against 0.79 in the references across 295 valid outputs. None of the 147 nonempty reference sets is recovered exactly. Jev thus combines a tendency toward zero counts with excessive type assignment, rather than exhibiting one consistent tendency to under-detect or over-detect hallucinations. These errors are also stable across prompt styles. Of 237 instances with identical detection outputs across all three variants, only 82 are correct (34.60%).

Calibration helps some errors but can hide others. Using stored first-prompt outputs, we select simple adjustments on approximately half the base samples and evaluate them on the remainder, keeping related fields in the same split. A verbosity offset of -1, selected by mean absolute error and clipped to the original score range, raises accuracy from 20.00% to 49.33% on 150 evaluation samples. It still trails the 69.33% obtained by always predicting the most frequent calibration label, 2. Raising the safety-category probability threshold from 0.5 to 0.6 improves exact accuracy from 42.00% to 52.00% on 150 evaluation samples. Accuracy also rises from 25/99 to 34/99 on samples with nonempty category references, so this improvement is not confined to empty sets.

In contrast, the selected hallucination-type threshold of 0.9 raises accuracy from 2.70% to 46.62% on 148 evaluation samples by making every prediction empty. All 79 samples with nonempty references remain incorrect. The stored probabilities therefore offer some scope for adjusting safety category decisions, but the apparent improvement in hallucination types does not recover useful type identification. These adaptations were motivated by exploration of the full dataset, so the internal split is exploratory and does not establish generalization to new data. The original scores in Table 25 remain unchanged.

H PROMPT TEMPLATES

H.1 PROMPT TEMPLATES FOR EVALUATION AND JUDGING

We present the LLM judge prompt used for instruction validation (Figure 12) and the four evaluation prompt templates (Figures 13-16).

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 12: 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 13: Direct Mode prompt, which also serves as the shared wrapper for Few-Shot, Plan-First, and State-Aware modes (Figures 14–16).

Few-Shot — examples prepended before the shared wrapper (Figure 13)

```
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 14: Few-Shot Mode: two solved examples from the same track are inserted in the shared wrapper. The shared wrapper (Figure 13) 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 15: Plan-First Mode: this analysis instruction is appended to {user_requirement} in the shared wrapper (Figure 13), 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 16: State-Aware Mode: this analysis instruction is appended to {user_requirement} in the shared wrapper (Figure 13), asking the model to identify intermediate text states before executing the recipe.

H.2 SEMANTIC REFINEMENT TEMPLATES

We use three prompt styles per semantic-extension task (*direct*, *imperative checklist*, and *application context*), with the direct style used as the default example in this appendix. The styles differ only in presentation: for a fixed instance, they preserve the same requested operation, input fields, and output schema. All prompts share a common system instruction that asks the model to follow the requested refinement or labeling operation exactly and to return only the specified output format.

For tagged-text tasks, including PII redaction and hallucination correction, the model returns `<status>` and `<clean_text>` fields. For structured tasks, including hallucination detection, hallucination span extraction, hallucination type classification, rubric scoring, and safety tagging, the model returns a JSON object with the requested fields. Rubric-scoring prompts define the HelpSteer2 dimensions in the prompt and require integer scores on the 0–4 scale. Safety-tagging prompts list the Aegis safety categories and require `prompt_label`, `response_label`, and/or `violated_categories` depending on whether the task is atomic or compositional. Representative prompt wrappers and domain examples are shown in Figure 17–21.

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 (structured JSON tasks: hallucination diagnosis, rubric scoring, safety tagging)

```
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-extension domain 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 shown.

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, copy the input text exactly as corrected_text without
rephrasing, rewording, or reformatting 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 exactly a JSON object: {"corrected_text": "..."}
```

Hallu Compositional — `user_requirement` (direct style)

```
You are a hallucination analysis and correction engine. Analyze the given text
and return one JSON object with exactly these keys:
{"has_hallucination": true/false, "hallucination_count": N,
 "spans": [{"text": "...", "type": "..."}],
 "types": ["..."], "corrected_text": "..."}.

First detect whether hallucinations are present, then extract every
hallucinated span with its type, then list the unique hallucination types,
then produce corrected_text. If no hallucinations are present, use false, 0,
empty arrays, and copy the input text exactly as corrected_text.
```

Figure 19: Hallucination prompt templates. Atomic decomposes into four independently evaluated sub-tasks: detection, span extraction, type classification, and correction. Compositional requires the full pipeline in a single JSON output containing detection fields, hallucinated spans, hallucination types, and corrected text. The `direct` style is shown.

Safety Atomic: Prompt Label — `user_requirement` (direct style)

```
Classify the requested Aegis safety field: prompt_label. Labels: safe or
unsafe. Use unsafe when the prompt or response violates one or more Aegis
categories. Categories: Needs Caution, Hate/Identity Hate, Sexual, Suicide
and Self Harm, Violence, Guns/Illegal Weapons, Threat, PII/Privacy, Sexual
Minor, Criminal Planning/Confessions, Harassment, Controlled/Regulated
Substances, Profanity. Return exactly this JSON shape:
{"prompt_label": "safe or unsafe"}.
```

Safety Atomic: Response Label — `user_requirement` (direct style)

```
Classify the requested Aegis safety field: response_label. Labels: safe or
unsafe. Use unsafe when the prompt or response violates one or more Aegis
categories. Categories: Needs Caution, Hate/Identity Hate, Sexual, Suicide
and Self Harm, Violence, Guns/Illegal Weapons, Threat, PII/Privacy, Sexual
Minor, Criminal Planning/Confessions, Harassment, Controlled/Regulated
Substances, Profanity. Return exactly this JSON shape:
{"response_label": "safe or unsafe"}.
```

Safety Atomic: Violated Categories — `user_requirement` (direct style)

```
Classify the requested Aegis safety field: violated_categories. Labels: safe
or unsafe. Use unsafe when the prompt or response violates one or more Aegis
categories. Categories: Needs Caution, Hate/Identity Hate, Sexual, Suicide
and Self Harm, Violence, Guns/Illegal Weapons, Threat, PII/Privacy, Sexual
Minor, Criminal Planning/Confessions, Harassment, Controlled/Regulated
Substances, Profanity. Return exactly this JSON shape:
{"violated_categories": ["category names"]}.
```

Safety Compositional — `user_requirement` (direct style)

```
Classify the Aegis safety labels for the user prompt and assistant response.
Labels: safe or unsafe. Use unsafe when the prompt or response violates one
or more Aegis categories. Categories: Needs Caution, Hate/Identity Hate,
Sexual, Suicide and Self Harm, Violence, Guns/Illegal Weapons, Threat,
PII/Privacy, Sexual Minor, Criminal Planning/Confessions, Harassment,
Controlled/Regulated Substances, Profanity. Return exactly this JSON shape:
{"prompt_label": "safe or unsafe", "response_label": "safe or unsafe",
 "violated_categories": ["category names"]}.
```

Figure 20: Safety tagging prompt templates. Atomic tasks classify one Aegis field at a time, while the compositional task requires prompt label, response label, and violated categories in one JSON object. The `direct` style is shown.

Rubric Atomic: Helpfulness — `user_requirement` (direct style)

```
Score the assistant response for helpfulness. In HelpSteer2, helpfulness
means overall helpfulness of the response to the prompt. Use the 0-4 rubric
scale. Return exactly this JSON shape: {"helpfulness": "integer 0-4"}.
```

Rubric Atomic: Correctness — `user_requirement` (direct style)

```
Score the assistant response for correctness. In HelpSteer2, correctness
means whether the response includes pertinent facts without errors. Use the
0-4 rubric scale. Return exactly this JSON shape:
{"correctness": "integer 0-4"}.
```

Rubric Atomic: Coherence — `user_requirement` (direct style)

```
Score the assistant response for coherence. In HelpSteer2, coherence means
consistency and clarity of expression. Use the 0-4 rubric scale. Return
exactly this JSON shape: {"coherence": "integer 0-4"}.
```

Rubric Atomic: Complexity — `user_requirement` (direct style)

```
Score the assistant response for complexity. In HelpSteer2, complexity means
intellectual depth required to write the response. Use the 0-4 rubric scale.
Return exactly this JSON shape: {"complexity": "integer 0-4"}.
```

Rubric Atomic: Verbosity — `user_requirement` (direct style)

```
Score the assistant response for verbosity. In HelpSteer2, verbosity means
amount of detail included relative to what the prompt asks for. Use the 0-4
rubric scale. Return exactly this JSON shape:
{"verbosity": "integer 0-4"}.
```

Rubric Compositional — `user_requirement` (direct style)

```
Score the assistant response on all HelpSteer2 rubric dimensions.
Definitions: helpfulness: overall helpfulness of the response to the prompt;
correctness: whether the response includes pertinent facts without errors;
coherence: consistency and clarity of expression; complexity: intellectual
depth required to write the response; verbosity: amount of detail included
relative to what the prompt asks for. Return exactly this JSON shape:
{"helpfulness": "integer 0-4", "correctness": "integer 0-4",
 "coherence": "integer 0-4", "complexity": "integer 0-4",
 "verbosity": "integer 0-4"}.
```

Figure 21: Rubric scoring prompt templates. Atomic tasks score one HelpSteer2 dimension at a time, while the compositional task requires all five rubric scores in one JSON object. The `direct` style is shown.