MOSAICREC: DIAGNOSING SPECIFICATION GROUNDING AND SET CONSTRUCTION IN RECOMMENDATION AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern recommendation agents can draw on user histories, item catalogs, and external tools to satisfy complex user requests. Their task is not merely to rank plausible items, but to construct recommendation sets that jointly satisfy constraints grounded in distributed evidence. End-to-end success alone cannot reveal whether an agent correctly grounded these constraints, successfully constructed a compliant set, or recovered from an earlier error. We introduce MosaicRec, a benchmark that separates the post-retrieval recommendation pipeline into specification grounding and set construction, enabling separate evaluation of each stage and diagnosis of their interaction. MosaicRec contains held-out tasks from Goodreads and an independently constructed Amazon split, each paired with a solver-certified feasible specification. Its constraints include personalized predicates that require joining user histories with catalog metadata. The evaluation crosses predicted and gold specifications with language-model and exact constructors, checks specification safety through counterexample search, and compares model-generated recommendations with exact execution of the same cached specifications. This design reveals stage-specific weaknesses and cross-stage recovery. Under identical common gold specifications, Gemma 4 31B substantially outperforms a similar-sized Qwen3-32B model, revealing a gap in constrained set construction. Separately evaluating intermediate specifications and final recommendations shows that GPT-5.6-sol can produce valid recommendations despite errors in its own specification by retaining access to the original request and tools; replacing this stage with exact execution of its predicted specifications can therefore reduce end-to-end success. These findings demonstrate how MosaicRec makes individual stages diagnosable while exposing how their interaction shapes final outcomes. Code and data are available at an anonymous repository: `https://anonymous.4open.science/r/MosaicRec_Anonymous`.

1 INTRODUCTION

Modern recommendation agents can access user histories, item catalogs, and external tools when responding to natural-language requests (Geng et al., 2022; Bao et al., 2023; Zhang et al., 2026; Shang et al., 2026). This becomes challenging when satisfying a request requires combining evidence distributed across these sources. Suppose a reader asks for four recent books carrying the genre tag they read most, by authors they have never read, avoiding main genres they have already explored, with at least three under-the-radar titles and no two sharing a main genre. The request specifies only part of what is needed to evaluate a recommendation: the reader's history determines the most-read tag, the read authors, and the explored genres, while the catalog supplies genre tags, main genres, publication years, and rating counts. A set of books may appear valid when each source is considered separately and still fail once these constraints are combined. Producing a valid recommendation therefore requires both grounding the request in distributed evidence and checking the recommended set as a whole.

Figure 1 highlights two distinct capabilities required by this setting. **Specification grounding** converts the request and retrieved evidence into executable conditions. Both the applicable rules and some of their values may need to be inferred across sources: a constraint such as "by an author

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.

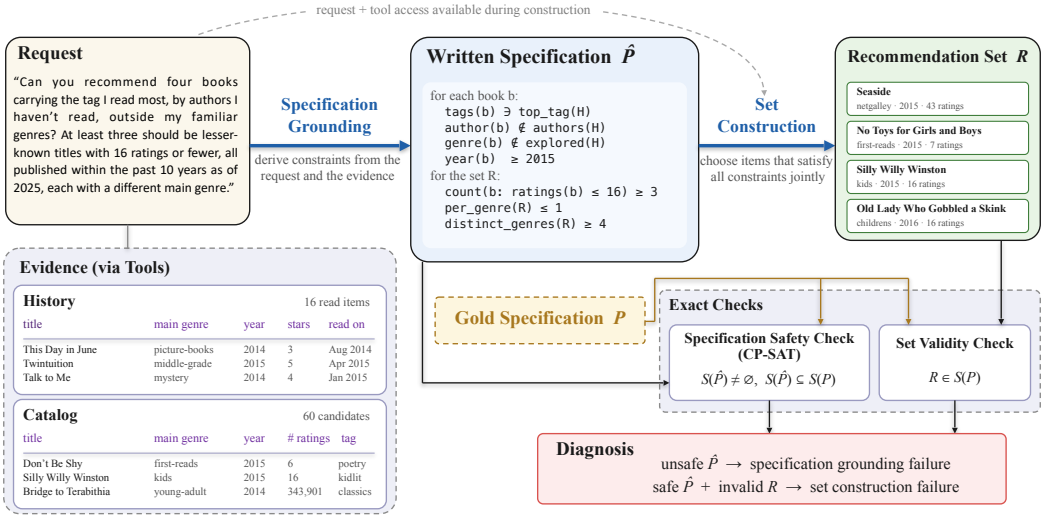

Figure 1: Overview of MosaicRec. The agent reads the request and reaches the user's history and the candidate catalog only through tools. **Specification grounding** turns them into a written specification $\hat{P}$; the most-read tag and the read authors must be derived from the history, and the explored genres are the reader's most-read main genres apart from that tag. **Set construction**, during which the request and the tools remain available, returns a set R satisfying every constraint jointly. Two exact checks compare $\hat{P}$ with the hidden gold specification P (CP-SAT counterexample search) and test R against P: an unsafe $\hat{P}$ is a specification grounding failure; an invalid R under a safe $\hat{P}$ is a set construction failure.

the user has never read," for example, requires joining the user's interaction history with catalog metadata. **Set construction** then selects items that satisfy the grounded conditions simultaneously. This cannot be reduced to filtering items independently, since every selected item may satisfy its local constraints while the set as a whole violates an author quota, genre cap, diversity requirement, or coverage condition. Industrial recommender systems explicitly optimize for such set-level objectives: the final re-ranking stage of production pipelines treats the displayed list as a whole, via author and category caps, freshness and long-tail quotas, and learned re-rankers, with reported online gains over item-wise ranking (Liu et al., 2022; Pei et al., 2019; Ie et al., 2019; Wilhelm et al., 2018). As language-model agents take on recommendation decisions, they must meet the same set-level requirements; the count, diversity, and long-tail constraints in our benchmark mirror these production rules.

Existing recommendation benchmarks, however, do not directly target this problem. Instruction-following and conversational benchmarks evaluate natural-language constraints (Zhang et al., 2026; Huang et al., 2026; Narasimhan & Narasimhan, 2026), while tool-based benchmarks expose user histories, item metadata, and reviews (Shang et al., 2026), but none jointly evaluates whether an agent can derive request-specific set constraints from distributed evidence, represent them correctly, and then construct a set that satisfies them. Evaluating only the final recommendation therefore conflates distinct failure modes: an agent may ground the specification incorrectly or fail to construct a valid set from a safe one.

We introduce **MosaicRec**, a benchmark that makes this evidence-to-specification-to-set process executable and diagnosable. Each task provides a natural-language request, a candidate pool, and tools for accessing user history and catalog metadata. The hidden target is an executable conjunction of item-level and set-level predicates, with some predicate values requiring a history–catalog join. We instantiate MosaicRec independently on Goodreads and Amazon, with 100 development tasks and 900 test tasks in each catalog. Every task is certified feasible by an exact solver, so evaluation failures cannot be attributed to impossible requests.

Beyond measuring end-to-end success, MosaicRec evaluates the written specification and the resulting recommendation set to distinguish failures with unsafe specifications from failures under

Table 1: Comparison with closely related recommendation-agent benchmarks along the path from external evidence to a valid set: external data access, grounding constraint values from multiple sources, global constraints on the returned set, guaranteed feasibility, and grounding-versus-construction diagnosis. A checkmark means explicitly required and evaluated; $\sim$ means partial or weaker support.

Benchmark	Tool Use	Cross-Source Grounding	Global Set Constraints	Certified Feasible	Failure Diagnosis
RecBench+ Huang et al. (2026)	×	×	×	×	×
AgentRecBench Shang et al. (2026)	✓	×	×	×	×
τ-Rec Narasimhan & Narasimhan (2026)	✓	×	×	∼	×
RecoAtlas Aouali et al. (2026)	✓	∼	×	×	×
MosaicRec	✓	✓	✓	✓	✓

safe ones. We use an exact constraint solver (CP-SAT) to search for counterexamples: recommendation sets that satisfy the predicted specification but violate the hidden gold specification. This is stronger than checking a single output produced from the predicted specification, since a constructor may happen to return a valid set even when the specification itself permits invalid ones. We therefore attribute failures to grounding when the predicted specification is infeasible or fails the safety check, and attribute a remaining failure to construction only when the specification has passed that check. We additionally test stricter specification equivalence. Two complementary diagnostics further separate the stages. Gold-specification construction gives the model constructor the hidden gold specification in place of its own, measuring construction ability free of grounding errors. Paired solver replay executes each model's cached predicted specification with CP-SAT and compares the result with the model's own recommendation on the same task, testing whether exact execution of a written specification helps or hurts.

Across five models from three providers and both MosaicRec catalogs, similarly sized Qwen3-32B and Gemma 4 31B show markedly different construction reliability under gold specifications, while GPT-5.6-sol exhibits no observed failures on the benchmark test tasks or five harder construction stress splits. Paired solver replay further shows that exact execution is not uniformly beneficial: replacing the model constructor with CP-SAT helps models with weaker construction but hurts those with stronger construction, such as GPT-5.6-sol. When the written specification is unsafe, stronger model constructors can sometimes recover by consulting the original request and tools, whereas exact solver execution of that specification removes this opportunity. Together, these results illustrate the value of an explicit specification as an intermediate representation: it allows failures to be attributed to a specific stage, and it reveals behavior that end-to-end success conceals, such as a constructor recovering from errors in its own specification.

Our contributions are:

- **Benchmark.** We introduce MosaicRec, a solver-certified benchmark for set recommendation under distributed evidence, instantiated independently on Goodreads and Amazon.
- **Diagnostic framework.** We jointly assess specification safety and set validity using exact counterexample search, gold-specification construction, and paired solver replay.
- **Empirical diagnosis.** Construction reliability under gold specifications can differ markedly between similarly sized models, and exact execution of a model's own specification helps weaker constructors but hurts stronger ones, which can recover from errors in their specifications.

2 Related Work

Language models and recommendation agents. Early work tunes language models on interaction data Geng et al. (2022); Bao et al. (2023) or prompts them to rank candidates Hou et al. (2024); InstructRec Zhang et al. (2026) additionally lets users express preferences in language. RecBench+ Huang et al. (2026) broadens the query space to condition- and profile-based requests, but evaluates conditions per item rather than over the joint output. AgentRecBench Shang et al. (2026) provides tools for accessing user histories, item metadata, and reviews to rank candidate

items. Conversational benchmarks evaluate how agents clarify user needs through dialogue. τ-Rec Narasimhan & Narasimhan (2026) assesses whether agents can recommend a movie that satisfies user constraints elicited through conversation. MosaicRec brings these two settings together, requiring agents to interpret conversational constraints in the context of user history and recommend a set of items that jointly satisfies those constraints.

Set-aware recommendation. Recommendation considers not only the relevance of individual items but also the quality of the recommended set as a whole. Prior work optimizes such sets for diversity, calibration, and joint utility Carbonell & Goldstein (1998); Chen et al. (2018); Steck (2018); Ie et al. (2019); Himan et al. (2019), typically under a predefined objective. RecoAtlas Aouali et al. (2026) extends this line of work to recommendation agents that search a large catalog and produce comparative or bundle reports. These outputs are evaluated using held-out recovery, learned behavior-grounded utilities, and language-model judgments. MosaicRec requires agents to derive request-specific constraints from user instructions and supporting evidence, then construct a set that satisfies them jointly. It evaluates the resulting set against executable item-level and set-level predicates and checks the agent's specification separately, allowing failures to be attributed to specification grounding or construction under a fixed priority rule.

Executable agent evaluation. AgentBench Liu et al. (2024), WebArena Zhou et al. (2024), ToolLLM Qin et al. (2024), and τ-bench Yao et al. (2024) evaluate agents through interaction with environments and tools. MosaicRec focuses on executable verification of recommendation sets and their underlying specifications. It additionally analyzes ReAct-style tool traces Yao et al. (2022) to measure whether agents access the required sources, resolve history items to catalog metadata, and encode derived facts in their specifications. IFEval Zhou et al. (2023) similarly checks instruction following through executable constraints, but evaluates properties of generated text rather than evidence-derived constraints on a recommendation set. MosaicRec uses the public Goodreads corpus Wan & McAuley (2018); Wan et al. (2019), distributed through the AgentSociety Challenge Yan et al. (2025). We build on these data to introduce new tasks, constraint annotations, solvers, and diagnostic evaluations.

Table 1 summarizes how the compared benchmarks cover individual aspects of this evaluation, while MosaicRec combines them in a single framework. Agents derive request-specific constraints from accessible evidence and construct a recommendation set that satisfies them jointly. Each task is certified feasible, and exact checks of both the returned set and the agent's specification support failure attribution to grounding or construction under a fixed priority rule.

3 Task and Benchmark

Executable set recommendation. A task presents an agent with a user u, a candidate pool C_u, a target size k, and a request q. History H_u and catalog metadata $\mathcal{M}$ are accessible only through tools. Each request is paired with an executable specification P_q, hidden from the agent, that formalizes what it requires. The agent returns a set $R \subseteq C_u$ with $|R| = k$ and succeeds only if all predicates hold:

$$\mathrm{CSR} = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \mathbf{1} \left[\bigwedge_{\phi \in P_{q_t}} \phi(R_t, H_{u_t}, \mathcal{M}) \right]. \tag{1}$$

CSR is all-or-nothing per task and does not need model judge. Some predicates check items independently; others constrain the returned set jointly, via author quotas, genre coverage, or long-tail counts. Cross-source predicates further involve values that are never stated in the request and must be derived from accessible evidence, such as the set of authors appearing in the users' history. Table 2 lists the eight constraint families.

Construction. MosaicRec builds 1,000 tasks from Goodreads histories and 1,000 from Amazon reviews, split into 100 development and 900 test tasks each. A chronological 70/30 cut yields the observed history and the later highly rated positives. Starting from a feasible positive set, the builder derives constraints and adds candidates sampled from targeted violation categories, plausible distractors, and popular items. Categories include publication-year, excluded-genre, prior-reading, and history-derived violations. Candidates may violate zero or more item-level predicates; the builder

Table 2: The eight constraint families. Level: checked on each item or on the returned set. Counts are test tasks containing the family; a task may contain several.

Family	Requirement	Level	Tasks
preference	Include or exclude requested genres	Item	900
temporal	Respect publication dates and prior reading	Item	900
interest	Include or exclude requested shelves	Item	290
keyword	Match a requested keyword	Item	524
cross_source	Apply author, genre, or shelf facts derived from history	Both	584
count	Meet set size and author or genre quotas	Set	900
diversity	Limit genre concentration or require genre coverage	Set	340
long_tail	Include enough long-tail items	Set	625

records these classes after checking the complete item-level specification. Candidate order is shuffled, no tool exposes the positive set or the hidden specification, and history tools are restricted to the pre-cut window. Each task contains 60 candidates and asks for three to eight items. Genre labels are an item's most popular user shelves after removing status and ownership shelves, with the most popular remaining shelf as the main genre; every relative time constraint states its reference year (Appendix B).

Recoverability and feasibility. A request can state a value literally, paraphrase it deterministically, or give a derivation rule whose value must be computed from history. A checker recomputes every inferred value from the frozen evidence and rejects ties or labels that the stated rule does not uniquely determine; all 900 test tasks in the books pass. Separately, an exact constrained-set solver certifies that every task has a valid answer, so no failure can be attributed to an infeasible task. The Amazon split reuses the predicate definitions, checker, evaluator, and certification pipeline, adapting only the domain vocabulary and taxonomy, and dropping fields absent from Amazon data, such as publication year.

Agent interface and scope. Agents can page through history and metadata, search or summarize the pool, and check a proposed set against their own working specification. The verifier reports which stated predicates a set violates, but it cannot reveal constraints the agent never wrote down. MosaicRec starts after candidate retrieval: it isolates the problem of turning a fixed pool into a request-compliant set; candidate generation, relevance learning, or open-ended clarification are out of scope. Appendix B details construction, predicates, tools, and quality controls.

4 Diagnostic Protocol

4.1 Crossing specification and constructor

An end-to-end score cannot say whether an agent misunderstood the request or failed to act on a correct understanding. To examine both, we cross two specifications (predicted and gold) with two constructors (the language model and an exact solver). Table 3 names the four cells; the first three columns of Table 4 report them.

E2E (predicted specification, model constructor) is the deployed agent: the model writes the specification from the request and tools and then constructs the set itself, so its CSR conflates grounding and construction.

Pred+Solver (predicted specification, CP-SAT) keeps the model's grounding but hands the written specification to an exact solver, so it measures what the written specification alone supports when construction is error-free; comparing it with E2E asks whether the model's own construction adds or removes value (Section 5.3).

Gold+LLM (gold specification, model constructor) injects the hidden gold specification directly into agent state and asks the model only to construct, so it isolates construction from grounding; the request is never parsed a second time. The remaining cell, gold specification with CP-SAT, is the feasibility oracle and reaches 1.000 by certification.

Table 3: The four specification–constructor cells. Rows fix the specification, columns the constructor; the three named cells are the CSR columns of Table 4.

	Language-model constructor	Exact solver (CP-SAT)
Predicted specification	**E2E**: the deployed agent; grounding and construction both by the model	**Pred+Solver**: model grounding, exact construction; what the written specification alone supports
Gold specification	**Gold+LLM**: gold specification injected, model constructs; isolates construction	Gold+Solver: feasibility oracle, 1.000 by certification

Executing a specification once is not enough to judge it: a missing predicate can be masked by favorable solver tie-breaking. Let $\mathcal{S}(\hat{P})$ and $\mathcal{S}(P)$ denote the candidate sets admitted by the predicted and gold specifications. We call $\hat{P}$ safe when

$$\mathcal{S}(\hat{P}) \neq \varnothing \quad \text{and} \quad \mathcal{S}(\hat{P}) \subseteq \mathcal{S}(P). \tag{2}$$

CP-SAT first checks that $\hat{P}$ is satisfiable at all, then searches for a set that satisfies $\hat{P}$ while violating some active gold predicate. A witness proves unsafety; if $\hat{P}$ is satisfiable and no such witness exists, it is safe. The resulting taxonomy is simple: an unsatisfiable prediction cannot answer the request, a satisfiable but unsafe prediction admits at least one gold-invalid set, and a safe prediction can be passed to exact construction without risk of gold violations. Within the safe class, equivalence $\mathcal{S}(\hat{P}) = \mathcal{S}(P)$ additionally demands that the prediction preserve every gold-valid answer. Safety permits over-restriction and does not imply complete request understanding. We report both rates, using safety for attribution and equivalence as a stricter check.

A failed end-to-end run is attributed to grounding if $\hat{P}$ is unsafe and to construction otherwise, following an earliest-blocking-stage convention. The two events are disjoint and sum exactly to $1 - \mathrm{CSR}$. We additionally condition construction failure on safety, separating the failure rate within that subset from how often a usable specification is produced. Safe subsets differ across models, however, so this conditioning does not control task difficulty; the common gold-specification tasks provide the controlled construction comparison. This is not a causal decomposition: an unsafe specification can coexist with construction errors, which this priority rule absorbs into grounding.

4.2 Grounding trace

The tool traces support three complementary measurements. Source reached averages coverage of the required acquirable source categories; entities resolved averages the fraction of history item identifiers joined to metadata; and operationalized measures, for each derivable gold fact, whether it appears in a history-derived list predicate of the submitted specification, averaged over predicates. Their denominators differ, so differences between columns are not stage-wise losses. The trace summaries cover the full rollout, including construction retrieval after the specification was submitted. They measure the association between retrieval and the submitted specification, not the evidence available at submission time. Appendix A gives the CP-SAT encoding and its validation; Appendix G gives the trace and predicate-level diagnostics.

4.3 Paired solver replay

The two predicted-specification cells use separate grounding rollouts; the gold-specification cells bypass grounding. Differences between the predicted cells therefore cannot isolate the effect of executing a fixed specification exactly. To measure that effect directly, we perform a paired constructor-substitution experiment: for each end-to-end rollout, we hold the task and the cached specification $\hat{P}_t$ fixed and replace the model constructor with CP-SAT, compiling the identical object to unbounded CP-SAT and comparing the solver's answer with the set the rollout returned. We refer to this intervention as paired solver replay. The solver sees metadata and $\hat{P}_t$ but not the request or gold constraints. The LLM constructor starts a fresh conversation with $\hat{P}_t$ and the original request and can retrieve evidence again; the parser conversation is not carried forward. The replay therefore evaluates exact execution of the written specification as a policy. Because each pair shares both task

Table 4: Core results on all 900 Goodreads test tasks. The first three columns cross specification and constructor (Table 3). Safe/Equiv evaluate the end-to-end rollout's specification: safe admits only gold-valid sets; equivalent also preserves every gold-valid answer. F_{ground}: share of all tasks failing with an unsafe specification; $F_{\text{construct|safe}}$: failure rate given a safe specification.

	Observed CSR			End-to-end diagnosis			
Model	Pred+Solver	Gold+LLM	E2E	Safe	Equiv	F_{ground}	$F_{\text{construct\|safe}}$
Qwen3-32B	0.704	0.677	0.503	0.593	0.531	0.322	0.294
Gemma 4 31B	0.791	0.976	0.792	0.709	0.639	0.200	0.011
GPT-4o-mini	0.569	0.383	0.258	0.338	0.302	0.539	0.602
GPT-4o	0.753	0.572	0.429	0.656	0.511	0.279	0.446
GPT-5.6-sol	0.812	1.000	0.874	0.777	0.706	0.126	0.000

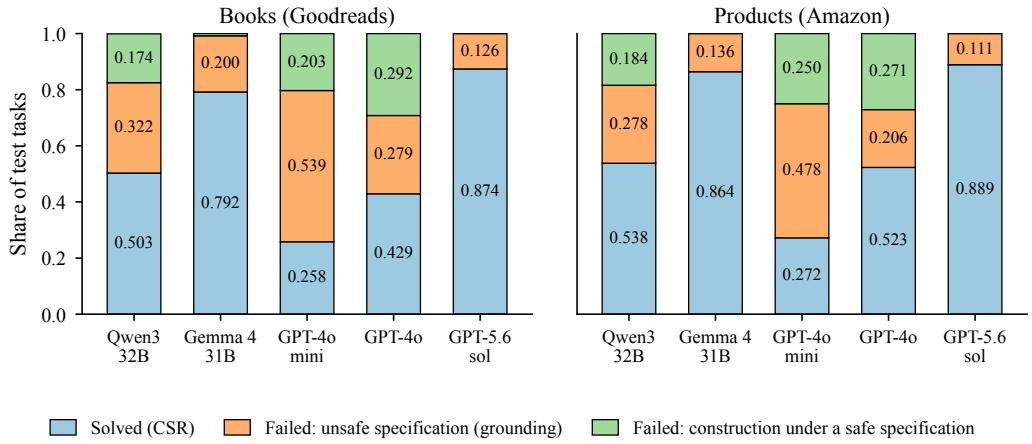

Figure 2: Failure attribution across catalogs. Each bar partitions 900 tasks into solved (CSR), failed with an unsafe specification (grounding), and failed with a safe specification (construction).

and specification, exact McNemar tests apply to the discordant outcomes. We then stratify by safety to compare replay outcomes under safe and unsafe written specifications.

5 Experiments

Setup. We evaluate Qwen3-32B, Gemma 4 31B, GPT-4o-mini, GPT-4o, and GPT-5.6-sol on all 900 held-out tasks in each catalog. Open-weight models run locally with vLLM. Main runs use the standard tool tier, a 16-step parsing budget with one additional forced-submit retry, and a separate 12-step construction budget. Gold-specification runs bypass parsing.

5.1 Similar-sized models exhibit different failure modes

Despite similar parameter counts, Qwen3-32B and Gemma 4 31B differ substantially in end-to-end CSR: 0.503 versus 0.792 (Table 4). The strongest controlled evidence of a construction gap comes from the common gold-specification tasks (Gold+LLM): Qwen constructs a valid set on 67.7% of tasks, compared with 97.6% for Gemma, a 29.9 percentage-point difference.

Family-level results locate construction weaknesses in joint composition requirements (Table 6): with gold specifications, Qwen satisfies long-tail and diversity constraints at rates of 0.771 and 0.776, compared with 0.984 and 0.985 for Gemma. Similar parameter counts therefore do not imply similar construction reliability.

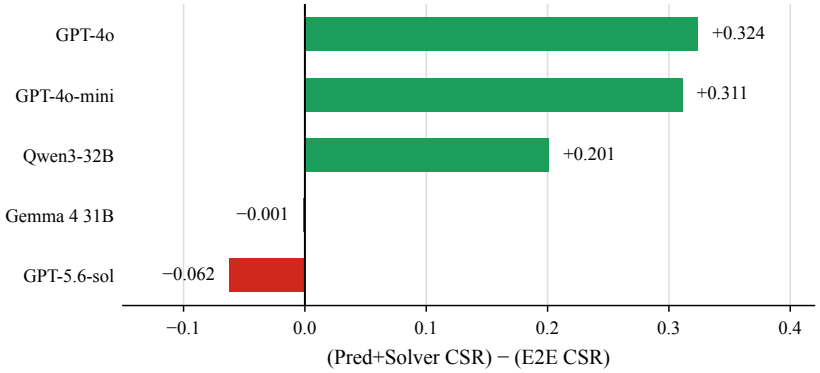

Figure 3: Exact solver execution versus end-to-end construction on Goodreads. Each bar is Pred+Solver CSR minus E2E CSR from Table 4: the model's written specification executed by CP-SAT versus the model constructing the set itself.

5.2 Specification safety and gold-specification construction

GPT-5.6-sol exhibits no construction failures when its predicted specification is safe (Figure 2, left) and achieves 1.000 observed CSR with gold specifications (Table 4). Its Goodreads specification safety/equivalence rates are 0.777/0.706, respectively. We investigate whether this performance depends on repeated verifier-guided search and whether more demanding construction tasks expose failures.

The implemented verifier-guided local search achieves only 0.268 CSR with 12 proposals, improving to 0.599 and 0.840 with 100 and 500 proposals, respectively (Table 6). In contrast, GPT-5.6-sol averages just 1.01 verifier calls per task on the main test split (Table 10). This indicates that its success does not depend on repeatedly querying the verifier to find a valid set.

Across five gold-specification stress splits varying pool size, set size, scarcity, competing constraints, and numeric requirements, the estimated solution density drops from 0.4082 on the main test split to 0.0001 on the numeric probe. On that probe, Search-500 achieves only 0.033 CSR, while GPT-5.6-sol maintains 1.000 across all five stress splits (Figure 4 in Appendix E).

5.3 Stronger models can recover from specification errors

Figure 3 compares the two predicted-specification cells of Table 4: executing the model's written specification with CP-SAT (Pred+Solver) versus letting the model construct the set itself (E2E). Exact execution raises CSR for GPT-4o, GPT-4o-mini, and Qwen3-32B, leaves Gemma 4 31B essentially unchanged, and lowers it for GPT-5.6-sol.

The breakdown by specification safety explains this divergence. CP-SAT succeeds whenever the specification it receives is safe. When it is unsafe, however, the model constructor retains access to the original request and retrieval tools, allowing it to recover from errors in its own specification. On Goodreads, GPT-5.6-sol returns valid answers for 27 of the 112 tasks with infeasible predicted specifications.

Thus, exact execution benefits models whose construction errors outweigh their recoveries, but hurts models such as GPT-5.6-sol for which the recoveries it forecloses outweigh the construction errors it repairs.

5.4 The main diagnostic patterns persist across catalogs

All five models have end-to-end results on 900 Amazon tasks as well as Goodreads; three models have paired solver replay on both catalogs. On Amazon, Qwen3-32B has construction-attributed failures on 18.4% of all tasks, compared with 17.4% on Goodreads. Gemma 4 31B and GPT-5.6-sol have none on Amazon (Figure 2). Solver replay changes Amazon CSR by $+16.3$, -5.4, and

-6.3 percentage points for Qwen, Gemma, and GPT-5.6-sol, respectively, preserving the direction observed on Goodreads (Table 7). These diagnostic patterns thus persist across catalogs under the shared generation and evaluation pipeline.

The attribution pattern is also robust to a stricter grounding criterion. Conditioning on strict equivalence rather than safety, which additionally requires the written specification to preserve every gold-valid answer, changes each model's conditional construction-failure rate by at most 0.022 under either catalog and leaves it at zero wherever it was already zero (Table 11). The two criteria differ exactly on the over-restricted class, safe specifications that exclude some gold-valid answer, which holds between 1.3% and 14.4% of tasks across models and catalogs.

6 Discussion and Limitations

Architectural implication. Explicit specifications are useful because they provide an inspectable and verifiable interface between grounding and construction. Our paired replay results further show that the utility of exact execution depends on the quality of this interface. Exact execution reliably realizes safe specifications, but it cannot in general recover information omitted or misrepresented during grounding. This suggests a design in which specifications are validated against the original request and supporting evidence before exact execution, with revision when necessary. More broadly, the paired analysis characterizes how specification quality and constructor choice interact under the evaluated protocol.

Construction headroom. Under gold specifications, the strongest evaluated models achieve very high construction success: Gemma approaches full success and GPT-5.6-sol has no observed failures on the tested tasks. Harder construction settings that vary set size, candidate scarcity, interacting quotas, aggregate numeric constraints, and verifier availability sharply reduce the density of valid solutions and defeat model-free search, yet the strongest model still solves every tested instance. The current construction axes therefore offer limited resolution among the strongest constructors, which does not mean that constrained set construction is solved. Future benchmark extensions could introduce qualitatively different forms of coupling, such as longer evidence chains, interacting budgets, or constraints whose feasible regions require additional evidence acquisition.

Scope and uncertainty. Our results characterize five models, two independently instantiated catalogs, and a fixed agent protocol. Each configuration is evaluated once per task; accordingly, the reported intervals quantify variation across tasks rather than repeated-rollout variability, and very small differences should be interpreted cautiously. The semantic guarantees hold with respect to the finite candidate pools and executable specifications. Both source catalogs are public, so pretraining exposure cannot be ruled out, although success still requires satisfying task-specific constraints over held-out instances. The study covers the post-retrieval setting with well-specified requests; structured grounding methods, ambiguous or conflicting requests, candidate retrieval, further domains, and repeated rollouts remain open. Reported perfect or near-perfect construction scores should therefore be read as empirical results for the evaluated configurations

7 Conclusion

MosaicRec makes the post-retrieval recommendation pipeline executable: an agent must derive constraints from evidence scattered across a request, a history, and a catalog, then construct a globally valid set on a task certified to be feasible. Exact semantic checks and controlled experiments on MosaicRec show that the strongest tested constructors approach or reach perfect observed success under gold specifications. Paired solver replay shows that exact execution of the written specification helps three models and hurts two, with outcomes depending on both construction reliability and the sufficiency of the written interface. Agent designers should validate and revise specifications before executing them, rather than delegating unconditionally; evaluators should measure specification safety and set validity alongside end-to-end success.

AI USE STATEMENT

Generative AI assistance was used during manuscript revision to review the methodology and interpretation of existing results, edit the text, update reporting and check numerical consistency. The language models evaluated as experimental agents are identified in Section 5. The authors retain responsibility for the final manuscript and its claims.

REFERENCES

Imad Aouali, Flavian Vasile, Otmane Sakhi, Alexandre Gilotte, and Benjamin Heymann. Recoatlas: From semantic plausibility to set-level utility in llm recommendation agents. *arXiv preprint arXiv:2605.18805*, 2026.

Keqin Bao, Jizhi Zhang, Yang Zhang, Wenjie Wang, Fuli Feng, and Xiangnan He. Tallrec: An effective and efficient tuning framework to align large language model with recommendation. In *Proceedings of the 17th ACM conference on recommender systems*, pp. 1007–1014, 2023.

Jaime G Carbonell and Jade Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR*, volume 98, pp. 290941–291025, 1998.

Laming Chen, Guoxin Zhang, and Eric Zhou. Fast greedy map inference for determinantal point process to improve recommendation diversity. volume 31, 2018.

Shijie Geng, Shuchang Liu, Zuohui Fu, Yingqiang Ge, and Yongfeng Zhang. Recommendation as language processing (rlp): A unified pretrain, personalized prompt & predict paradigm (p5). In *Proceedings of the 16th ACM conference on recommender systems*, pp. 299–315, 2022.

Abdollahpouri Himan, Mansoury Masoud, Burke Robin, and Mobasher Bamshad. The unfairness of popularity bias in recommendation. *arXiv preprint arXiv:1907.13286*, 2019.

Yupeng Hou, Junjie Zhang, Zihan Lin, Hongyu Lu, Ruobing Xie, Julian McAuley, and Wayne Xin Zhao. Large language models are zero-shot rankers for recommender systems. In *European conference on information retrieval*, pp. 364–381. Springer, 2024.

Jiani Huang, Shijie Wang, Liangbo Ning, Wenqi Fan, Shuaiqiang Wang, Dawei Yin, and Qing Li. Towards next-generation recommender systems: A benchmark for personalized recommendation assistant with llms. In *Proceedings of the Nineteenth ACM International Conference on Web Search and Data Mining*, pp. 217–226, 2026.

Eugene Ie, Vihan Jain, Jing Wang, Sanmit Narvekar, Ritesh Agarwal, Rui Wu, Heng-Tze Cheng, Tushar Chandra, and Craig Boutilier. Slateq: A tractable decomposition for reinforcement learning with recommendation sets. In *IJCAI*, volume 19, pp. 2592–2599, 2019.

Weiwen Liu, Yunjia Xi, Jiarui Qin, Fei Sun, Bo Chen, Weinan Zhang, Rui Zhang, and Ruiming Tang. Neural re-ranking in multi-stage recommender systems: A review. 2022.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, volume 2024, pp. 52989–53046, 2024.

Bharath Sivaram Narasimhan and Karthik R Narasimhan. tau-rec: A verifiable benchmark for agentic recommender systems. *arXiv preprint arXiv:2606.10156*, 2026.

Changhua Pei, Yi Zhang, Yongfeng Zhang, Fei Sun, Xiao Lin, Hanxiao Sun, Jian Wu, Peng Jiang, Junfeng Ge, Wenwu Ou, et al. Personalized re-ranking for recommendation. In *Proceedings of the 13th ACM conference on recommender systems*, pp. 3–11, 2019.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. 2024:9695–9717, 2024.

Yu Shang, Peijie Liu, Yuwei Yan, Zijing Wu, Leheng Sheng, Yuanqing Yu, Chumeng Jiang, An Zhang, Fengli Xu, Yu Wang, et al. Agentrecbench: Benchmarking llm agent-based personalized recommender systems. volume 38, 2026.

Harald Steck. Calibrated recommendations. In *Proceedings of the 12th ACM conference on recommender systems*, pp. 154–162, 2018.

Mengting Wan and Julian McAuley. Item recommendation on monotonic behavior chains. In *Proceedings of the 12th ACM conference on recommender systems*, pp. 86–94, 2018.

Mengting Wan, Rishabh Misra, Ndapandula Nakashole, and Julian McAuley. Fine-grained spoiler detection from large-scale review corpora. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2605–2610, 2019.

Mark Wilhelm, Ajith Ramanathan, Alexander Bonomo, Sagar Jain, Ed H Chi, and Jennifer Gillenwater. Practical diversified recommendations on youtube with determinantal point processes. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pp. 2165–2173, 2018.

Yuwei Yan, Yu Shang, Qingbin Zeng, Yu Li, Keyu Zhao, Zhiheng Zheng, Xuefei Ning, Tianji Wu, Shengen Yan, Yu Wang, et al. Agentsociety challenge: Designing llm agents for user modeling and recommendation on web platforms. pp. 2963–2967, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. 2022.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ bench: A benchmark for tool-agent-user interaction in real-world domains. corr abs/2406.12045 (2024). *arXiv preprint arXiv:2406.12045*, 2024.

Junjie Zhang, Ruobing Xie, Yupeng Hou, Wayne Xin Zhao, Leyu Lin, and Ji-Rong Wen. Recommendation as instruction following: A large language model empowered recommendation approach. *ACM Transactions on Information Systems*, 43(5):1–37, 2026.

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

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

A Deciding Specification Safety

Encoding. For each task we build one Boolean variable per candidate item, indicating whether the item is selected. Item-level predicates fix the corresponding variables; cardinality, per-genre and per-author caps, distinct-genre coverage, new-author counts, and long-tail quotas become linear constraints over those variables. Caps are weighted sums rather than membership counts, so an item listing the same author twice contributes twice, matching the evaluator. Safety of a predicted specification $\hat{P}$ against gold P is decided with CP-SAT: one initial call establishes that $\hat{P}$ is satisfiable, then one counterexample query per active gold predicate searches for a set that satisfies $\hat{P}$ and violates that predicate; since $\neg P$ is a disjunction over active predicates, one feasibility problem per predicate covers it. A witness proves unsafety, and infeasibility for every predicate proves $\mathcal{S}(\hat{P}) \subseteq \mathcal{S}(P)$.

Validation. We run the solver with no time limit and no node budget. A safety certificate requires conclusive feasibility and infeasibility statuses; a timeout or unknown status would not constitute a proof. Each returned counterexample is re-checked with the benchmark evaluator, and a witness the evaluator does not confirm raises an error rather than being counted. On small candidate pools we compare CP-SAT against exhaustive enumeration of all subsets and require exact agreement on both the safety verdict and the set of violated predicates.

Equivalence. Deciding equivalence adds a third family of CP-SAT calls. Completeness, $\mathcal{S}(P) \subseteq \mathcal{S}(\hat{P})$, is the mirror image of safety: for each active predicate of $\hat{P}$ we search for a set that satisfies gold and violates that predicate, and a witness proves that $\hat{P}$ excludes a gold-valid set. Equivalence is safety together with completeness, and both directions are validated against exhaustive enumeration on small pools as above. Appendix F reports the end-to-end rates and what the gap between the two tests costs. The solver-backed cell behaves the same way: Qwen3-32B reaches 0.530 against 0.607 safe, Gemma 4 31B 0.650 against 0.718, GPT-4o-mini 0.299 against 0.327, GPT-4o 0.509 against 0.636, and GPT-5.6-sol 0.713 against 0.781.

The safety–equivalence gap is widest for GPT-4o in the books solver-backed cell. Class proportions are descriptive and do not establish a capability ordering. We report safety alongside equivalence because safety is the criterion the attribution needs. A specification that is safe in this benchmark is also satisfiable, so it admits at least one set and admits only gold-valid sets; a failure under such a specification is one that a correct constructor would have avoided, which is exactly what F_{build} is meant to count. Charging those failures to grounding instead would attribute to the parser a task that its own output was sufficient to solve.

Implementation self-check. The solver-backed agent is by construction the composition of the parser with the benchmark solver, so recomputing its score offline from cached specifications has to reproduce the online score task by task. Across all 900 test tasks both are 0.753 and they agree on all 900 tasks individually. This checks the offline pipeline; it says nothing about whether executing a specification once is a good measure of grounding, which is the question safety answers.

Same-specification constructor replay. The paired solver replay of Section 4.3 uses the satisfiability half of the same CP-SAT encoding. For each cached end-to-end specification it asks for one feasible set with no objective, one search worker, and no time limit, then validates the returned set with the benchmark evaluator. The replay is deterministic and changes neither the specification nor the candidate pool. When the specification is safe, tie-breaking is irrelevant because every feasible set is gold-valid; this is why the exact solver reaches 1.000 on every safe slice in Table 7. When it is unsafe, the score of one execution can depend on which admitted set the solver selects and is not a safety certificate. We therefore use the unsafe-slice execution rate to explain the observed replay outcomes, while the stronger architectural conclusion is set-theoretic: satisfying an unsafe specification alone does not guarantee request satisfaction; recovering such a guarantee requires information outside that interface.

B Benchmark Construction and Interface

Histories and positives. Goodreads interactions are ordered by date and cut at the 70th percentile. The earlier window is the history exposed to the agent; later items rated at least four stars and absent from the earlier window form the positive set. Readers need at least twelve dated interactions. The same abstraction is used for Amazon reviews through a domain adapter.

Item fields. For Goodreads, an item's genre tags are its twenty most popular user shelves after removing a fixed list of status and ownership shelves (for example `to-read`, `owned`, `kindle`); the most popular remaining shelf is the main genre used by the diversity and history-genre predicates. Shelves are user-generated, so a small share of main genres are activity labels rather than topics: in the test candidate pools 1.7% of items have a main genre such as `netgalley` or `first-reads`, and such a label enters the requested genre of 5 of 900 tasks and the explored-genre list of 4 of the 92 tasks that carry one. Predicates are evaluated against the recorded labels, so scoring is exact with respect to the released metadata, but the semantic validity of those few constraints is correspondingly weaker. Amazon uses the catalog's category taxonomy, which has no such labels. Relative time constraints are rendered with an explicit reference year (2025) stated in the request, so "within the last ten years" is well defined regardless of when the history ends.

Tools and solver. Agents can page through history and item metadata, search and summarize the candidate pool, and check a set against their working specification. Metadata is returned in pages of ten in the standard tool tier. The exact constructor reads pool metadata directly and performs depth-first search after item-level filtering, with admissible bounds for set-level predicates. It is complete

Table 5: Constraint satisfaction on the complete 900-task test split. CSR: fraction of tasks whose returned set satisfies every predicate; violation rate: mean per-task fraction of returned items breaking an item-level predicate; R@10 and Jaccard: overlap with the reader's observed future items, for context only. CSR intervals are two-sided 95% Wilson intervals over tasks.

Agent	CSR	95% CI	Viol. rate	R@10	Jaccard	Tokens/task
Random from pool	0.000	[0.000, 0.004]	0.740	0.076	0.044	–
Most popular	0.001	[0.000, 0.006]	0.765	0.101	0.062	–
Item oracle + greedy cap	0.200	[0.175, 0.227]	0.000	0.328	0.239	–
Qwen3-32B parser + solver	0.704	[0.674, 0.733]	0.185	0.455	0.350	35,901
Qwen3-32B end-to-end	0.503	[0.471, 0.536]	0.168	0.375	0.277	82,880
Qwen3-32B, gold specification given	0.677	[0.645, 0.706]	0.092	0.424	0.315	34,569
Gemma 4 31B parser + solver	0.791	[0.763, 0.816]	0.164	0.476	0.372	39,389
Gemma 4 31B end-to-end	0.792	[0.765, 0.817]	0.064	0.461	0.349	90,183
Gemma 4 31B, gold specification given	0.976	[0.963, 0.984]	0.008	0.498	0.381	38,551
GPT-4o-mini parser + solver	0.569	[0.536, 0.601]	0.181	0.456	0.343	42,206
GPT-4o-mini end-to-end	0.258	[0.230, 0.287]	0.196	0.287	0.195	99,038
GPT-4o-mini, gold specification given	0.383	[0.352, 0.416]	0.069	0.345	0.244	62,414
GPT-4o parser + solver	0.753	[0.724, 0.780]	0.170	0.455	0.349	17,139
GPT-4o end-to-end	0.429	[0.397, 0.461]	0.077	0.352	0.246	83,328
GPT-4o, gold specification given	0.572	[0.540, 0.604]	0.004	0.413	0.295	59,243
GPT-5.6-sol parser + solver	0.812	[0.785, 0.836]	0.149	0.484	0.379	14,996
GPT-5.6-sol end-to-end	0.874	[0.851, 0.895]	0.025	0.534	0.417	50,137
GPT-5.6-sol, gold specification given	1.000	[0.996, 1.000]	0.000	0.556	0.438	32,140
Gold specification + solver (oracle)	1.000	[0.996, 1.000]	0.000	0.574	0.451	–

for the supplied specification, but one selected answer need not maximize success against hidden gold when that specification is unsafe. The semantic checks and paired replay instead use CP-SAT.

C Complete System Results

Table 5 reports confidence intervals, item-level violations, future-item overlap, and token cost in addition to CSR. Future overlap is context rather than the target: only 0.718 of later highly rated items satisfy the corresponding request, while 0.426 of items in a certified valid answer are absent from observed future behavior.

Calibration of the gold-specification cell. The verifier becomes an exact test against gold in this cell, so we compare the models with greedy repair and verifier-guided model-free search at nominal 12-step and larger budgets. The implemented short verify-and-repair loops do not reproduce the strongest model's score; this does not exclude better model-free search procedures.

Efficiency. Solver-backed pipelines use fewer tokens for every model. For Qwen this saving accompanies higher accuracy; for Gemma the scores are nearly equal. The saving also accompanies higher accuracy for GPT-4o-mini and GPT-4o. GPT-5.6-sol is the exception: its solver-backed run is cheaper but less accurate, which motivates the paired intervention rather than an unconditional efficiency recommendation.

D Paired Solver Replay Details

The safe slice isolates removable construction loss. CP-SAT is perfect there for every model by definition, while direct success ranges from 0.398 to 1.000. On the unsafe slice, one solver execution is not a semantic guarantee: another feasible-set objective could select a different admitted answer. The set-theoretic conclusion is objective-independent: an unsafe interface is infeasible or admits a gold-invalid set, so it cannot guarantee a request-valid answer. Exact McNemar tests use direct-only and solver-only discordant task pairs. Table 8 also separates infeasible predictions from other unsafe specifications. For GPT-5.6-sol on Goodreads, direct execution succeeds on 27 of 112 infeasible predictions. Those successes contribute 27 of the net 52 tasks lost under solver replay; the

Table 6: Construction baselines with gold constraints. The item oracle ranks by popularity with a greedy author cap and no global planning. Budget counts repair rounds (greedy repair), swap proposals (local search), or model-response steps (LLMs); the units are not comparable. At 12 proposals local search reaches 0.268, versus 1.000 for the strongest model.

Agent	Budget	CSR	long_tail	diversity
Item oracle + greedy cap	–	0.200	0.003	0.574
Greedy + repair tool	12	0.260	0.003	0.747
Verifier-guided search	12	0.268	0.174	0.856
Verifier-guided search	100	0.599	0.722	0.982
Verifier-guided search	500	0.840	0.901	1.000
Qwen3-32B	12	0.677	0.771	0.776
Gemma 4 31B	12	0.976	0.984	0.985
GPT-4o-mini	12	0.383	0.379	0.553
GPT-4o	12	0.572	0.557	0.662
GPT-5.6-sol	12	1.000	1.000	1.000
Exact solver (oracle)	–	1.000	1.000	1.000

Table 7: Same-specification solver replay. Each cached end-to-end specification is executed by CP-SAT, which sees only the specification and metadata, and compared with the model's original answer. n_S/n_U: safe and unsafe specifications, including infeasible ones; Δ_{all}: direct minus solver CSR over 900 tasks. Table 8 gives discordant counts and paired tests.

			safe specification		Unsafe specification		
Model	Catalog	n_S/n_U	Solver	Direct	Solver	Direct	Δ_{all}
Qwen3-32B	Books	534/366	1.000	0.706	0.260	0.208	-0.196
Qwen3-32B	Products	584/316	1.000	0.716	0.149	0.209	-0.163
Gemma 4 31B	Books	638/262	1.000	0.989	0.210	0.313	+0.022
Gemma 4 31B	Products	701/199	1.000	1.000	0.141	0.387	+0.054
GPT-4o-mini	Books	304/596	1.000	0.398	0.337	0.186	-0.303
GPT-4o	Books	590/310	1.000	0.554	0.287	0.190	-0.326
GPT-5.6-sol	Books	699/201	1.000	1.000	0.179	0.438	+0.058
GPT-5.6-sol	Products	727/173	1.000	1.000	0.092	0.422	+0.063

Table 8: Paired solver replay tests and infeasible predictions. Direct-only and solver-only count discordant outcomes on the same 900 cached specifications; p is a two-sided exact McNemar test without multiplicity adjustment. The last two columns count infeasible specifications and gold-valid direct answers among them; solver success on them is zero.

Model	Catalog	Direct-only	Solver-only	p	Infeasible	Direct pass
Qwen3-32B	Books	34	210	3.79e-32	120	6
Qwen3-32B	Products	42	189	1.86e-23	158	16
Gemma 4 31B	Books	43	23	1.87e-2	115	17
Gemma 4 31B	Products	53	4	5.91e-12	123	23
GPT-4o-mini	Books	43	316	1.75e-52	36	0
GPT-4o	Books	26	319	2.76e-65	109	4
GPT-5.6-sol	Books	58	6	9.03e-12	112	27
GPT-5.6-sol	Products	58	1	2.08e-16	120	31

remaining net loss comes from satisfiable unsafe predictions. Paired tests are conditional on these cached specifications and this solver selection policy.

E Replication and Construction Stress Tests

Products change taxonomy, vocabulary, item space, and available fields; in particular, they have no publication-year predicate. Table 9 reports the complete replication numbers behind the right panel of Figure 2: the ordering of end-to-end CSR and the absence of construction-attributed failure for GPT-5.6-sol reproduce. Gemma has seven such failures on Goodreads and none on Amazon, and

Table 9: Failure attribution on books and on an independently built products catalog. CSR is end-to-end; F_{ground} and $F_{\text{construct}}$ are shares of all tasks assigned to grounding and construction by the priority rule.

	Books			Products		
Model	CSR	F_{ground}	$F_{\text{construct}}$	CSR	F_{ground}	$F_{\text{construct}}$
Qwen3-32B	0.503	0.322	0.174	0.538	0.278	0.184
Gemma 4 31B	0.792	0.200	0.008	0.864	0.136	0.000
GPT-4o-mini	0.258	0.539	0.203	0.272	0.478	0.250
GPT-4o	0.429	0.279	0.292	0.523	0.206	0.271
GPT-5.6-sol	0.874	0.126	0.000	0.889	0.111	0.000
Rule parser + solver	0.766	–	–	0.850	–	–
Feasibility oracle	1.000	–	–	1.000	–	–

Table 10: Construction probes. Reference splits contain 900 tasks; each stress split contains 60. Density: median over 25 task files, each estimated from 20,000 random k-subsets of item-valid candidates. Search-500: local search with 500 swap proposals. Checks: mean verifier calls per task. The scarce-pool, enlarged-set, and numeric probes raise the completion cap from 4,000 to 8,000 tokens; verifier removal is tested only on the enlarged-pool split.

Split	Construction pressure	Density	Search-500	GPT-5.6-sol	Checks
Main test split	$\lvert C_u\rvert = 60, k \in [3, 8]$	0.4082	0.840	1.000	1.01
Products split	$\lvert C_u\rvert = 60$	0.3452	0.917	1.000	1.00
Scarce pool	$k = 20, \lvert C_u\rvert = 120$	0.0273	0.083	1.000	1.03
Enlarged pool	$\lvert C_u\rvert = 200, k \leq 12$	0.0111	0.283	1.000	1.00
Competing families	diversity + long tail	0.0059	0.467	1.000	1.02
Enlarged set size	$k = 25, \lvert C_u\rvert = 250$	0.0015	0.050	1.000	1.07
Numeric aggregate	count sum in $\pm 2\%$ window	0.0001	0.033	1.000	1.08
Verifier removed	enlarged pool; no verifier	0.0111	–	1.000	0.00

the same-specification table above shows that all three models with paired replay on both catalogs retain their replay sign.

Solution density estimates the fraction of sampled k-subsets of item-valid candidates that satisfy all set-level conditions. The probes vary pool and set size, scarcity, competing quotas, and a narrow numeric aggregate. Density is estimated on the first 25 task files in lexicographic order, using 20,000 random subsets per task; these are descriptive subset estimates, not whole-split measurements. At density 0.0001, only about two samples per task succeed, so the lowest estimates are imprecise. Low density also does not imply difficulty for a structured search algorithm. Search-500 is the implemented local swap search, scored by the number of satisfied families; its replacement pool includes item-invalid candidates. Its failure does not exclude stronger model-free heuristics. The main test splits contain 900 tasks, whereas each stress split contains 60. The scarce-pool, enlarged-set, and numeric probes raise the completion cap from 4,000 to 8,000 tokens; the other probes and enlarged-pool verifier ablation retain 4,000. The numeric constraint sums ratings counts, not rating scores. GPT-5.6-sol solves all tested tasks under these budgets; verifier-call counts alone do not measure total reasoning effort.

F Strict Grounding and Shared Misreads

Safety admits satisfiable over-restriction; equivalence rejects it. The strict criterion can only move failures from construction to grounding because equivalence implies safety. Across the ten model–catalog pairs it changes conditional construction failure by at most 0.022 and leaves every zero unchanged. Over-restriction that makes the specification infeasible is already classified as unsafe; the residual safe-but-incomplete class retains at least one gold-valid answer but may exclude others.

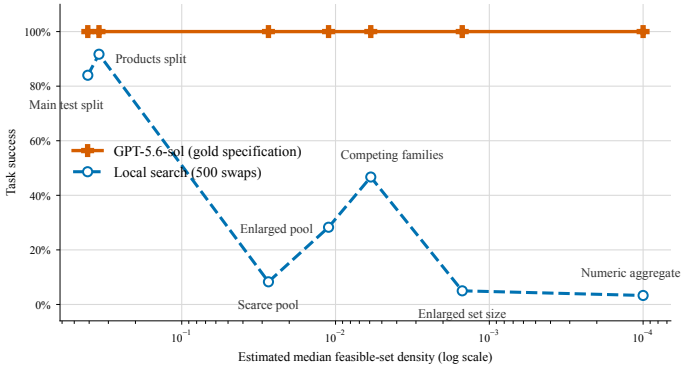

Figure 4: Construction probes do not break observed saturation. Each stress split contains 60 tasks; reference splits contain 900. Estimated median solution density spans about 3.6 orders of magnitude. Local search reaches 0.033 on the numeric probe; GPT-5.6-sol solves every tested task. Budgets and density-estimation limits are described in this appendix.

Table 11: Grounding under a strict test. Each predicted specification falls in exactly one class: equiv, $\mathcal{S}(\hat{P}) = \mathcal{S}(P)$; over, safe but excluding some gold-valid sets; unsat, over-restricted to infeasibility; perm, satisfiable but unsafe. The middle block reports observed CSR on the two over-restrictive classes. The right block recomputes the conditional construction rate of Table 4 with equivalence instead of safety; since equivalence implies safety, this moves failures only from construction to grounding, by at most 0.022.

		Specification class (share of tasks)				CSR on slice		$F_{\text{construct}}$ given	
Model	Catalog	equiv	over	unsat	perm	over	unsat	safe	equiv
Qwen3-32B	Books	0.531	0.062	0.133	0.273	0.661	0.050	0.294	0.289
Gemma 4 31B	Books	0.639	0.070	0.128	0.163	0.968	0.148	0.011	0.009
GPT-4o-mini	Books	0.302	0.036	0.040	0.622	0.312	0.000	0.602	0.592
GPT-4o	Books	0.511	0.144	0.121	0.223	0.631	0.037	0.446	0.467
GPT-5.6-sol	Books	0.706	0.071	0.124	0.099	1.000	0.241	0.000	0.000
Qwen3-32B	Products	0.543	0.106	0.176	0.176	0.705	0.101	0.284	0.282
Gemma 4 31B	Products	0.678	0.101	0.137	0.084	1.000	0.187	0.000	0.000
GPT-4o-mini	Products	0.401	0.013	0.027	0.559	0.500	0.000	0.603	0.607
GPT-4o	Products	0.596	0.116	0.114	0.174	0.654	0.175	0.381	0.388
GPT-5.6-sol	Products	0.708	0.100	0.133	0.059	1.000	0.258	0.000	0.000

Consensus-misread screen. No model writes a safe specification on 86 books tasks. Requiring at least 4 models to be refuted by the same predicate leaves 27 tasks, versus a conditioned null mean of 6.0 with interval $[2, 10]$. Products leave 15 candidates. The null preserves which models name a counterexample and restricts draws to each task's active predicates; models with unsatisfiable specifications cast no predicate vote. The resulting counts summarize shared predicate-level failures across models.

G Additional Failure Diagnostics

Evidence trace. Source coverage is generally high in end-to-end runs, but the history metadata and operationalization measurements are incomplete. Columns average different quantities and cannot be subtracted to obtain losses along a common funnel. Coverage is measured over the entire trajectory, so operationalization may condition on facts retrieved only after the parser submitted its object. Pre-submission trace snapshots would be required to identify facts available to the parser at the time it formed the specification.

Table 12: Complementary evidence measurements. Source reached: coverage of required source categories; entities resolved: history-item metadata coverage; operationalized: fraction of derivable gold facts written into history-derived list predicates; Specification safe: task-level safety rate. Denominators differ and retrieval covers the full rollout, so column differences are not stage losses. Rule-parser specifications are outside the semantic-check runs.

Agent	Source reached	Entities resolved	Operationalized	Specification safe
Qwen3-32B parser + solver	0.750	0.564	0.786	0.607
Qwen3-32B end-to-end	0.949	0.587	0.701	0.593
Gemma 4 31B parser + solver	0.811	0.697	0.889	0.718
Gemma 4 31B end-to-end	1.000	0.706	0.874	0.709
GPT-4o-mini parser + solver	0.948	0.735	0.456	0.327
GPT-4o-mini end-to-end	0.992	0.736	0.467	0.338
GPT-4o parser + solver	1.000	0.917	0.760	0.636
GPT-4o end-to-end	1.000	0.918	0.757	0.656
GPT-5.6-sol parser + solver	0.858	0.779	0.950	0.781
GPT-5.6-sol end-to-end	1.000	0.941	0.951	0.777

Table 13: Not every wrong predicate costs a task. Sufficient: share of tasks lost by injecting the agent's value into an otherwise safe specification; necessary: share of lost tasks won back by restoring this predicate alone. Predicates wrong on fewer than 5 tasks are omitted; brackets are Wilson intervals.

Predicate	Family	Wrong	Sufficient	Necessary
include_genres	preference	283	0.28 [0.23, 0.34]	0.52 [0.43, 0.62]
exclude_genres	preference	321	0.20 [0.16, 0.25]	0.23 [0.16, 0.31]
exclude_read_authors	cross_source	362	0.12 [0.09, 0.16]	0.35 [0.26, 0.44]
include_shelves	interest	131	0.24 [0.18, 0.32]	0.46 [0.32, 0.61]
exclude_history_genres	cross_source	69	0.36 [0.26, 0.48]	0.18 [0.09, 0.33]
require_history_shelves	cross_source	94	0.17 [0.11, 0.26]	0.30 [0.17, 0.47]
exclude_read_items	temporal	73	0.08 [0.04, 0.17]	0.38 [0.25, 0.54]
genre_composition	count	8	0.75 [0.41, 0.93]	0.33 [0.10, 0.70]
min_publication_year	temporal	5	1.00 [0.57, 1.00]	0.60 [0.23, 0.88]
keyword	keyword	20	0.20 [0.08, 0.42]	0.17 [0.05, 0.45]
exclude_shelves	interest	150	0.02 [0.01, 0.06]	0.11 [0.04, 0.24]
max_per_genre	diversity	20	0.05 [0.01, 0.24]	0.20 [0.04, 0.62]
min_new_authors	cross_source	52	0.00 [0.00, 0.07]	0.00 [0.00, 0.24]

Table 14: Per-family satisfaction. A family is satisfied when all of its predicates hold, so one task can fail in several columns. The gold-specification columns isolate families the agent cannot satisfy even when told what to satisfy. No family is active in fewer than 30 test tasks.

			Qwen3-32B		Gemma 4 31B		GPT-4o			GPT-5.6-sol	
Family	Tasks	Rule	Pred+Solver	Gold+LLM	Pred+Solver	Gold+LLM	Pred+Solver	End-to-end	Gold+LLM	End-to-end	Gold+LLM
preference	900	0.960	0.952	0.884	0.954	0.990	0.958	0.918	0.994	0.974	1.000
temporal	900	0.838	0.936	0.893	0.974	0.991	0.973	0.944	0.994	0.990	1.000
count	900	0.982	0.873	0.990	0.878	0.999	0.861	0.970	0.977	0.996	1.000
long_tail	625	0.978	0.850	0.771	0.853	0.984	0.837	0.510	0.557	0.936	1.000
cross_source	584	0.959	0.866	0.918	0.959	0.985	0.920	0.848	0.995	0.962	1.000
keyword	524	1.000	0.979	0.866	0.966	0.996	0.998	0.964	0.996	0.996	1.000
diversity	340	0.997	0.850	0.776	0.862	0.985	0.865	0.624	0.662	0.935	1.000
interest	290	1.000	0.893	0.903	0.876	0.993	0.931	0.886	0.993	0.917	1.000

Predicate attribution. A frequent mismatch need not be a costly one. Replacing one gold predicate with the agent's value measures whether the error is sufficient to lose a task; restoring one predicate in the agent's object measures whether the repair is enough to recover an observed loss. These solver-conditional interventions rank engineering priorities rather than certify specifications.

Constraint families. Cross-source deficits primarily shrink when gold grounding is supplied, while long-tail and diversity deficits persist for weaker constructors. Qwen versus Gemma demonstrates that construction reliability can differ sharply at the same scale, and GPT-5.6-sol eliminates the latter deficits from gold.

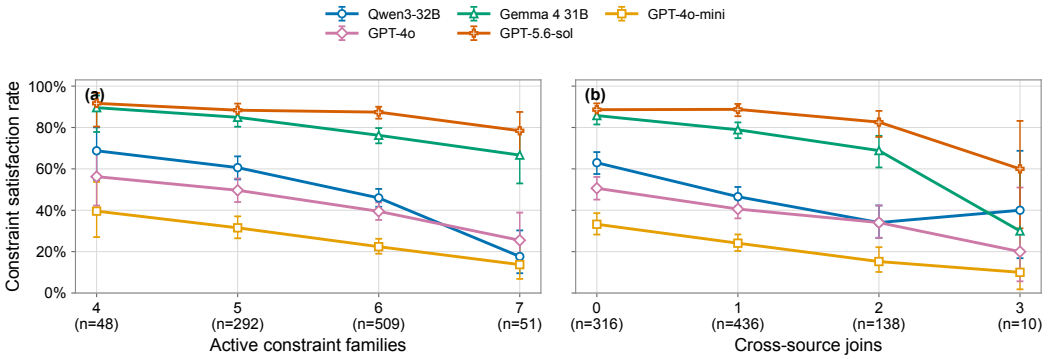

Figure 5: Satisfaction against the number of constraint families combined (left) and the number of cross-source joins required (right). Curves report end-to-end CSR for all five evaluated models; error bars are 95% Wilson intervals and slices with fewer than ten tasks are omitted. The intervals quantify task sampling, not rollout variance. Every task is certified feasible, so the plot describes an association with task composition rather than a causal effect of complexity.

Specification complexity. Higher family counts are generally associated with lower CSR across models. More history–catalog joins show the same broad pattern, although the three-join slice contains only ten tasks and should not support a monotonicity claim. Because bucket composition changes with the horizontal axis, these curves diagnose where performance is weakest; they do not estimate a causal effect of adding one constraint or join.