MatXBench: Benchmarking Multi-Table Scientific Reasoning and Executable Querying in Materials Science

Anonymous ACL submission

Abstract

Multi-table question answering over scientific databases requires models to perform reliable table retrieval, relational reasoning, and executable query generation. However, existing benchmarks predominantly focus on single-table settings or simplified schema structures, failing to capture the complexity of real-world scientific data analysis workflows. In this work, we introduce **MatXBench**, a multi-table question answering benchmark in the materials catalysis domain and, to the best of our knowledge, the first benchmark specified targeting multi-table relational reasoning in material science. MatXBench provides two complementary benchmark settings with different table organization strategies, enabling controlled evaluation of retrieval difficulty, schema ambiguity, and executable reasoning behavior under realistic scientific workflows. To facilitate fine-grained capability analysis, we further decompose multi-table reasoning into seven structured stages, including table selection, field selection, condition extraction, join selection, SQL generation, tool invocation, and executable code generation, allowing detailed investigation of model behaviors across different reasoning steps. We evaluate a diverse set of proprietary and open-source large language models under both constrained tool-based execution and open-ended code generation settings. Overall, MatXBench enables systematic evaluation of multi-table relational reasoning and executable question answering in scientific domains, providing a benchmark for analyzing model behavior across structured reasoning stages.

1 Introduction

As scientific research shifts toward a data-intensive paradigm, large-scale structured data has become a key driver of knowledge discovery (Kitchin, 2014; Zhang et al., 2019). In domains such as materials science, chemistry, and biomedicine, experimental measurements, computational results, and curated knowledge are typically stored across multiple relational tables (Jain et al., 2013; Aggour et al., 2024). While this modular organization aids data management, it also causes severe information fragmentation, as no single table contains sufficient evidence for complex scientific questions (Chen et al., 2025). Extracting meaningful insights thus requires reasoning across multiple tables, making cross-table understanding a critical yet underexplored problem(Jiang et al., 2023; Qiu et al., 2024).

Cross-table reasoning over scientific databases poses substantial challenges. Inter-table relationships are often implicit, relying on shared but non-unique attributes such as material identifiers or reaction conditions (Oche and Biswas, 2025; Deng et al., 2022; Gottschalk and Demidova, 2022). Real-world tables further exhibit duplicated entities, many-to-many relations, and inconsistent granularity, complicating joins and multi-step reasoning(Liu et al., 2024; Tong et al., 2025). These issues are amplified in automated settings—such as question generation, SQL synthesis, and tool-augmented reasoning—where naive joins may yield spurious correlations or combinatorial explosions (Li et al., 2023). Effective cross-table reasoning thus demands semantic alignment, entity disambiguation, and logical constraint awareness beyond basic database operations(Jiang et al., 2023; Deng et al., 2022; Gottschalk and Demidova, 2022).

Among scientific domains, catalysis offers a particularly compelling scenario for cross-table reasoning (Aggour et al., 2024; Zivic et al., 2025; Takahashi et al., 2023). Catalyst compositions, structural descriptors, reaction conditions, performance metrics, and computational methodologies are typically distributed across separate tables, requiring synthesis across heterogeneous sources and abstraction levels (Winther et al., 2019; Kirklin et al., 2015; Jain et al., 2013). Robust cross-table reasoning can enable systematic performance comparison, mech-

anistic analysis, and data-driven catalyst screening, making it both methodologically meaningful and impactful for accelerating discovery in this critical domain(Toyao et al., 2020; Butler et al., 2018).

Motivated by the aforementioned challenges and opportunities, we introduce a new benchmark **MatXBench**, for cross-table reasoning in scientific databases, with a particular focus on catalysis-related data. The benchmark is designed to reflect realistic scientific workflows, where answering a single question often requires integrating information across multiple heterogeneous tables and performing multi-step reasoning beyond surface-level retrieval. We highlight the primary contributions of this paper as follows:

- **A cross-table semantic graph for structured scientific data**: We construct a semantic graph over tables, columns, and scientific subdomains, preserving native table–column relations, linking subdomains via attribute-level associations, and adding column–column edges based on semantic similarity to support cross-table understanding.
- **Cross-table QA benchmarks under different data organizations**: We introduce two complementary benchmarks built on the same scientific tasks but differing in data organization: **MatX-Decompose** adopts a decomposed multi-table format, while **MatX-Native** retains the original organization that better reflects real-world scientific data. Both require cross-table association and alignment, enabling controlled analysis of how data organization affects model behavior, with scientific computation tools provided for derived metrics and quantitative reasoning.
- **Comprehensive evaluation across open- and closed-source models**: We conduct extensive experiments on diverse open- and closed-source models, focusing on cross-table association, semantic alignment, and scientific computation, providing empirical insights into their capabilities and limitations.

2 Related Work

Structured data question answering has been widely studied in both table QA and Text-to-SQL research (Jain et al., 2013; Yu et al., 2018; Zhong et al., 2017). Early studies mainly focus on single-table reasoning or SQL generation over relational databases (Yu et al., 2018; Zhou et al., 2025). Recent works extend this setting to multi-table question answering, requiring models to retrieve, connect, and reason across multiple interconnected tables under more complex relational structures(Pal et al., 2023; Wu et al., 2025; Qiu et al., 2024; Zou et al., 2025; Zhu et al., 2024).

Recent benchmarks explore different aspects of cross-table reasoning and evaluation settings. MultiTabQA (Pal et al., 2023) introduces large-scale synthetic multi-table question answering through SQL-template-based generation, focusing on tabular answer generation over multiple relational tables. TQA-Bench (Qiu et al., 2024) studies scalable multi-table reasoning under long-context settings. MMQA (Wu et al., 2025) evaluates multi-hop reasoning and schema understanding through tasks such as retrieval, key selection, and Text-to-SQL generation. GTR (Zou et al., 2025) emphasizes graph-based retrieval and multi-hop reasoning over large table collections. Other datasets, including SPARTA (Park et al., 2026), ReasonTabQA (Pan et al., 2026), and the CISS-based industrial QA benchmark (Wang et al., 2025), further explore reasoning-intensive or domain-oriented multi-table QA scenarios. Table 1 compares representative multi-table QA benchmarks in terms of dataset scale, task diversity, and support for tool or code-based evaluation.

Despite these advances, existing benchmarks mainly focus on generic relational reasoning, retrieval accuracy, or answer generation (Zhu et al., 2021; Chen et al., 2020; Wu et al., 2025). Most datasets evaluate cross-table reasoning independently from external computation or tool execution (Schick et al., 2023; Qu et al., 2025; Akhtar et al., 2025). In addition, current benchmarks provide limited support for scientific-domain reasoning scenarios that require integrating structured data understanding with domain-specific computation workflows (Mirza et al., 2024; Wang et al., 2024). As a result, the evaluation of computation-aware reasoning over interconnected scientific tables remains largely underexplored.

3 The MatXBench benchmark

3.1 Design Principles

MatXBench emphasizes coordinated reasoning across heterogeneous scientific tables rather than isolated single-table retrieval. Questions require cross-table retrieval, relational reasoning, aggre-

Table 1: Comparison of representative multi-table QA benchmarks in terms of dataset scale, task diversity, and execution support. Representative prior benchmarks include MultiTabQA (Pal et al., 2023), TQA-Bench (Qiu et al., 2024), MMQA (Wu et al., 2025), GTR (Zou et al., 2025), CISS-based Multi-table QA (Wang et al., 2025), SPARTA (Park et al., 2026), and ReasonTabQA (Pan et al., 2026).

Dataset	Domain	Data Source/Scenario	#.Tables	#.Questions	#.Tasks	Tool Eval.	Code Eval.
MultiTabQA	General	Spider, GeoQuery, ATIS	–	132,645	1	✗	✓
TQA-Bench	General	Finance, Healthcare, E-commerce	400	56,000	7	✗	✓
MMQA	General	Spider-derived relational QA	3,312	3,313	5	✗	✓
GTR	General	HybridQA, SQA, WikiTables	60,000	25,000	2	✗	✗
CISS-based Bench	Industrial	Traffic accident databases	38	628	4	✗	✗
SPARTA	General	Sports, Movies, Healthcare	19	3,390	3	✗	✓
ReasonTabQA	General	Sales, ERP, Manufacturing	1,932	5,523	1	✗	✓
MatXBench	Scientific	Material open dataset	204	48,732	7	✓ (28)	✓

gation, and scientific computation under realistic materials science scenarios.

MatXBench introduces two complementary benchmark settings with different schema organizations. MatX-Decompose simulates fragmented relational environments through column-wise table decomposition, while MatX-Native preserves richer relational context through cluster-based organization, enabling controlled analysis of structural complexity in multi-table reasoning.

MatXBench evaluates not only SQL generation, but also executable scientific reasoning. In addition to logical-form synthesis, the benchmark incorporates tool invocation and code generation settings to assess realistic executable workflows.

Domain experts participate throughout the benchmark construction process, including schema organization, tool design, and quality evaluation. Expert feedback is incorporated to improve scientific validity, practical relevance, and domain adaptability of generated questions and executable tools.

3.2 Benchmark Overview

MatXBench evaluates large language models on cross-table question answering over structured scientific data. Unlike conventional table QA benchmarks that mainly focus on single-table retrieval or schema-constrained reasoning, **MatXBench** emphasizes multi-table information integration, semantic alignment, and computation-aware reasoning in realistic scientific scenarios. To better reflect practical workflows, the benchmark combines structured reasoning tasks with domain-specific scientific computation. Overall, **MatXBench** contains 68 tables, 48,732 question-answer pairs, and 28 scientific computation tools spanning 12 scientific subfields.

The benchmark includes two complementary settings, namely **MatX-Decompose** and **MatX-Native**, which differ in table organization and relational structure. While MatX-Decompose evaluates reasoning under decomposed multi-table settings, MatX-Native preserves richer structural context from the original scientific data. This dual-setting design enables controlled evaluation of model behavior under different organizational forms of structured scientific data.

To comprehensively evaluate structured reasoning capabilities, **MatXBench** defines three categories of tasks: **Search Query**, **Aggregation**, and **Science Compute**. Search Query tasks evaluate cross-table retrieval and semantic matching abilities. Aggregation tasks further require models to perform structured operations over information collected from multiple tables. Science Compute tasks focus on computation-aware reasoning scenarios that combine structured data understanding with scientific calculations.

To support fine-grained evaluation, **MatXBench** further decomposes reasoning into intermediate steps, including **Table Selection**, **Field Selection**, **Condition Extraction**, **Join Selection**, **SQL Generation**, **Tool Invocation**, and **Code Generation**. These intermediate tasks enable process-level analysis beyond final-answer evaluation and provide more interpretable insights into model behavior.

MatXBench further includes 28 scientific computation tools designed to support evaluation of computation-aware reasoning over structured scientific data.

3.3 Benchmark Construction

MatXBench is constructed from real-world scientific tabular data collected from open scientific databases, published literature, and public reposi-

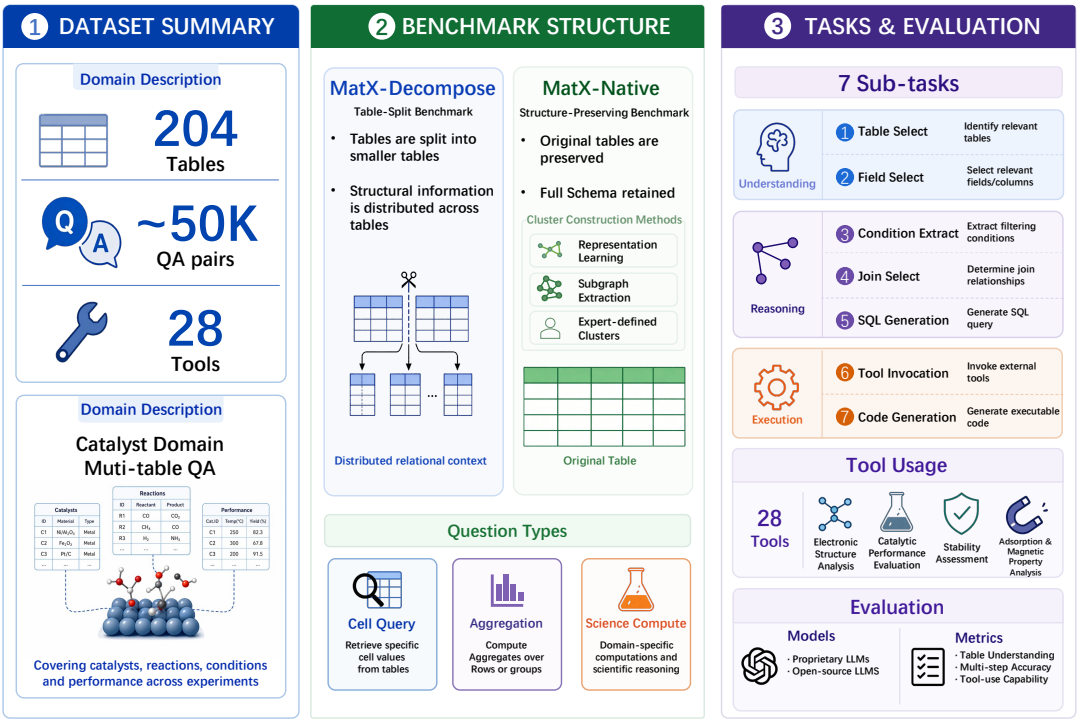

Figure 1: Benchmark overview.

tories. After filtering and cleaning, 68 tables are retained while preserving their original structural properties as much as possible. Each table is further annotated with structured metadata, including table-level descriptions and column-level semantic information. The overall pipeline integrates model-assisted processing, schema analysis, and expert verification to construct both MatX-Decompose and MatX-Native under a unified framework.

For **MatX-Decompose**, each original scientific table is decomposed into two semantically related sub-tables using a large language model. During this process, the original column organization is redistributed while preserving semantic consistency through metadata re-annotation. The resulting sub-tables derived from the same source table are treated as a single cluster, which serves as the basic unit for subsequent question and tool generation.

For **MatX-Native**, we preserve the original table structures and perform schema alignment to normalize heterogeneous field representations across different tables. Based on the aligned schema, we further construct a heterogeneous knowledge graph to capture relations among domains, tables, and fields, providing a unified structural representation for downstream clustering and reasoning.

To capture different types of table relationships, MatX-Native adopts three complementary clustering strategies, including semantic similarity-based clustering, graph-based structural clustering, and expert-defined clustering. The first strategy groups tables according to semantic and structural similarity, while the second extracts connected table groups directly from the constructed knowledge graph. The third introduces manually defined clusters provided by domain experts to preserve scientifically meaningful associations that may not be fully captured by automatic methods.

Finally, clusters from both MatX-Decompose and MatX-Native are fed into a unified ReAct-based generation pipeline for question-answer pair and scientific tool construction. Generated samples are iteratively refined through model-based evaluation and expert verification, while low-quality samples are filtered using predefined quality criteria.

4 Evaluation

4.1 Experimental Setup

We evaluate seven representative large language models, including DeepSeek-R1-Distill-Qwen-14B (DeepSeek-R1-14B) (DeepSeek-AI et al., 2025), Qwen3-14B (Yang et al., 2025), Qwen3.5-27B (Yang et al., 2025), Gemma-3-12B-IT (Gemma-3-12B) (Team et al., 2025), GPT-

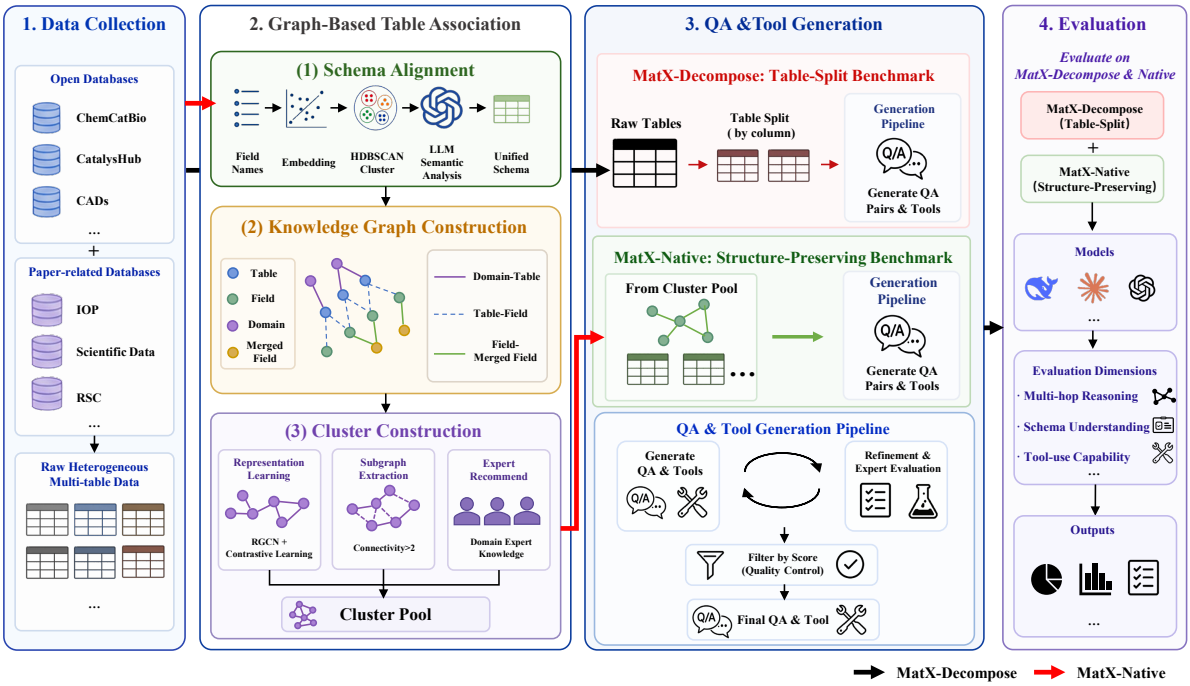

Figure 2: Overview of the MatXBench construction pipeline.

5.3 (Singh et al., 2026), Claude-Haiku-4.5 (Claude-4.5), and Gemini-3.1-Flash-Lite (Gemini-3.1)(202, 2026). Experiments are conducted on both **MatX-Decompose** and **MatX-Native**. For each benchmark setting, we sample 1,000 questions from each of the three task categories. As a result, each model is evaluated on a total of 6,000 questions across the two benchmark settings.

To support fine-grained capability analysis, the overall reasoning workflow is decomposed into multiple intermediate subtasks. After **SQL Generation**, we further evaluate two execution-oriented paradigms, namely **Tool Invocation** and **Code Generation**, to compare external tool-based execution and executable code-based reasoning under the same task setting.

To enable detailed analysis of error sources and capability contributions at different reasoning stages, we adopt a controlled evaluation protocol in which ground-truth outputs from preceding stages are provided as input when evaluating a target subtask. For example, during field selection, the model is provided with the correct table selection result instead of relying on predictions from previous stages. Similarly, downstream stages are evaluated using verified intermediate outputs to minimize cascading errors introduced by earlier failures. This protocol allows us to more precisely measure the contribution and difficulty of each intermediate reasoning stage, rather than only evaluating end-to-end performance.

4.2 Overall Performance

Table 2 presents the overall step-wise performance of different models on **MatXBench**. Overall, all evaluated models exhibit substantial performance gaps across multiple reasoning stages, indicating that cross-table scientific reasoning remains highly challenging even for frontier large language models. Although several models achieve relatively strong performance on intermediate retrieval-oriented stages, none of the evaluated systems demonstrates consistently reliable performance across the complete reasoning workflow.

Figure 3 further provides an overall view of the difficulty distribution across reasoning stages. By aggregating performance over all evaluated models, we observe a clear performance degradation in later stages of the pipeline, highlighting increasing difficulty in executable reasoning tasks.

In the following sections, we focus on several key components that exhibit representative behaviors or reveal notable failure patterns, including table selection, SQL generation, execution paradigms (tool vs. code), and the effect of benchmark structure (MatX-Decompose vs. MatX-Native).

Table 2: Step-wise performance breakdown across different stages of multi-table QA reasoning. Table, Field, Cond., Join, and SQL denote table selection, field selection, condition extraction, join selection, and SQL generation, respectively. Tool and Code represent tool-based and code-based execution paradigms following SQL generation. Overall measures full-pipeline performance and is computed as the joint success of all stages, i.e., Table $\times$ Field $\times$ Cond. $\times$ Join $\times$ SQL $\times$ Tool. Best results in each step are highlighted in bold, while second-best results are underlined.

Bench	Model	Step-wise Reasoning (%)					Execution (%)		Overall (%)
		Table	Field	Cond.	Join	SQL	Tool	Code	
MatX-Decompose	GPT-5.3	**58.8**	6.1	75.4	54.2	16.0	60.8	**10.5**	0.1
	Claude-4.5	55.1	12.8	73.1	57.0	23.1	61.2	7.9	0.4
	Gemini-3.1	53.0	10.7	69.2	48.5	**47.3**	**74.3**	10.2	**0.7**
	Qwen3.5-27B	58.0	6.6	73.5	46.8	23.1	62.2	8.1	0.2
	Qwen3-14B	37.0	18.5	67.5	51.1	17.8	61.4	6.4	0.3
	Gemma-3-12B	4.6	20.5	**76.6**	**81.8**	25.2	46.6	5.5	0.1
	DeepSeek-R1-14B	19.4	**23.0**	63.9	58.5	18.8	62.7	4.3	0.2
MatX-Native	GPT-5.3	**52.2**	**0.9**	**52.3**	63.3	12.5	97.7	0.0	0.0
	Claude-4.5	47.0	0.2	51.5	59.7	20.0	97.7	0.0	0.0
	Gemini-3.1	51.2	0.2	45.8	**63.7**	**60.8**	97.7	0.0	0.0
	Qwen3.5-27B	51.8	0.1	50.6	51.0	20.0	97.7	0.0	0.0
	Qwen3-14B	26.2	0.1	38.0	59.5	8.4	**100.0**	0.0	0.0
	Gemma-3-12B	21.7	0.0	43.1	36.4	16.0	97.1	0.0	0.0
	DeepSeek-R1-14B	25.8	0.2	23.6	14.3	17.0	96.5	0.0	0.0

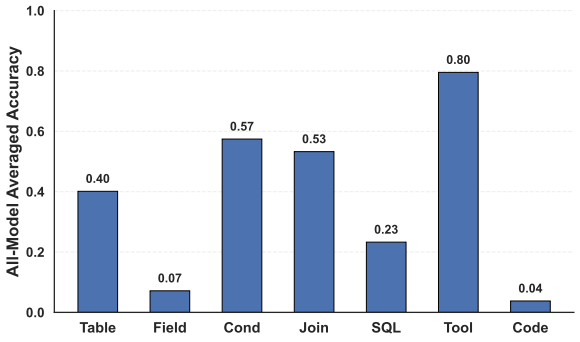

Figure 3: Average performance across all evaluated models at each stage of the multi-table QA pipeline, illustrating the relative difficulty of different reasoning components in MatXBench.

Table 3: Domain-level table selection accuracy under different benchmark settings. MatX-Native-same denotes MatX-Native instances where all tables associated with a question originate from the same domain, while MatX-Native-cross contains cross-domain table combinations.

Setting	#QA Pairs	Accuracy
MatX-Decompose	20,932	0.4098
MatX-Native-same	19,395	0.3860
MatX-Native-cross	1,463	0.0998

4.3 Table Selection Behavior

Table selection performance exhibits substantially larger variation across models compared with most downstream reasoning stages, indicating that table retrieval remains a major bottleneck in multi-table question answering. For example, several proprietary models achieve table selection accuracy above 0.5, while some open-source models remain below 0.2 under the same evaluation setting. Such capability gaps are considerably larger than those observed in later stages such as field selection or join selection, suggesting that accurately identifying relevant tables is still one of the primary challenges in cross-table scientific reasoning pipelines.

We further evaluate domain-level table selection accuracy under different benchmark settings, as shown in Table 3. Overall, models achieve comparable performance on MatX-Decompose and B-same, but degrade substantially under cross-domain retrieval. Specifically, accuracy reaches 0.4098 on MatX-Decompose and 0.3860 on MatX-Native-same, while dropping sharply to 0.0998 on MatX-Native-cross. This result suggests that current models benefit from domain-localized semantic and schema cues, but encounter greater difficulty when retrieving tables across domains with limited structural and semantic overlap.

To further investigate retrieval behavior, we analyze representative domain confusion patterns in

Table 4: Representative domain confusion patterns during table selection. ICF denotes Inorganic Crystal Fundamentals, while 2D Materials refers to Low-Dimensional / Two-Dimensional Materials.

Setting	Confusion Pair	Ratio (%)
MatX-Decompose	2D Materials → ICF	83.3
MatX-Decompose	Polymers → ICF	24.4
MatX-Decompose	Perovskites → ICF	22.5
MatX-Native-same	Perovskites → Photocatalysis	24.5

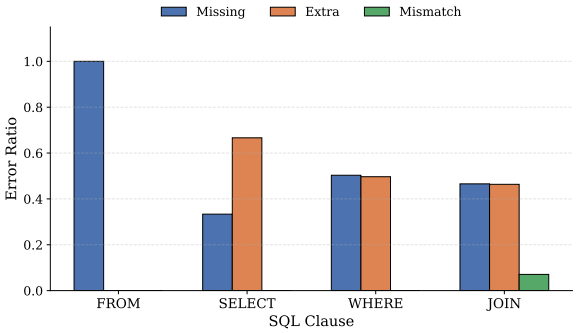

Figure 4: Normalized distributions of SQL clause errors across different structural components.

Table 4. In MatX-Decompose, multiple specialized domains are frequently mapped to broader material domains. For example, 2D Materials and Perovskites are often misclassified as Inorganic Crystal Fundamentals (ICF). Such confusion is highly asymmetric, where specialized domains collapse into more general parent domains, while the reverse direction rarely occurs. This suggests that models tend to over-generalize fine-grained material subfields based on coarse semantic similarity. In contrast, domains such as Magnetic Materials and Microengineering and Synthesis exhibit relatively clear boundaries, indicating stronger schema distinguishability.

MatX-Native-same exhibits a different failure pattern dominated by schema-level ambiguity. The most severe confusion occurs between Perovskites and Inorganic Crystal Fundamentals, where substantial bidirectional errors are consistently observed. Unlike MatX-Decompose, these errors are not purely hierarchical, but instead arise from overlapping structural fields and similar schema organization across related material domains.

Overall, the observed confusion patterns indicate that current models are more effective at leveraging coarse-grained semantic and schema-level similarities than distinguishing subtle cross-domain differences during table retrieval.

4.4 SQL Generation Challenges

To better understand executable reasoning behavior, we further analyze generated SQL queries by decomposing them into major structural components, including `FROM`, `SELECT`, `WHERE`, and `JOIN`. For each component, errors are categorized into three types: *missing*, *extra*, and *mismatch*. Figure 4 shows the normalized distributions of clause-level errors aggregated across all evaluated models.

SQL generation is evaluated under oracle intermediate supervision, where ground-truth outputs from upstream stages, including table selection, field selection, condition extraction, and join selection, are directly provided. Therefore, the observed SQL errors cannot be attributed solely to failures in schema retrieval or intermediate reasoning. Instead, they reveal a more fundamental limitation: even when correct structured constraints explicitly given, models still struggle to faithfully preserve them during SQL synthesis.

A dominant failure pattern is over-generation. For example, 66.6% of `SELECT`-related errors are *extra*, indicating that models frequently introduce redundant fields beyond the provided schema constraints. Similarly, `WHERE` clause errors remain nearly balanced between missing and extra conditions (50.3% vs. 49.7%), suggesting unstable condition grounding where models both omit required filters and hallucinate unsupported constraints. Together, these patterns indicate that SQL generation is often driven more by learned query completion priors than by strict constraint-grounded program construction.

Relational reasoning remains another bottleneck. Although join-related errors are less frequent overall, `JOIN` is the only component exhibiting substantial mismatch errors, suggesting that models still struggle to maintain faithful relational structures during executable synthesis even when correct join information is explicitly provided.

Overall, the observed error patterns suggest that current LLMs perform SQL generation primarily as probabilistic sequence completion rather than strictly grounded symbolic program synthesis.

4.5 Tool vs. Code-based Execution

Another notable observation is the substantial performance gap between **Tool Invocation** and **Code Generation**. Across both benchmark settings, tool-based execution consistently achieves significantly

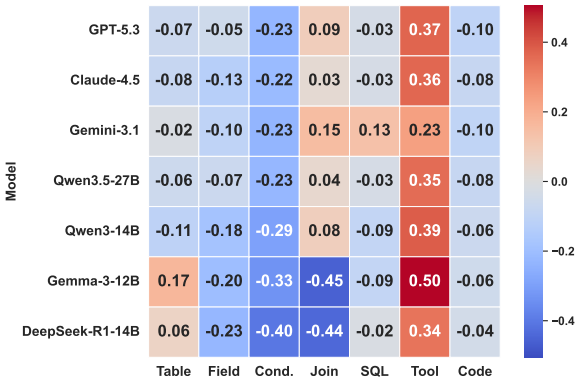

Figure 5: Relative performance differences between MatX-Native and MatX-Decompose across different reasoning stages and evaluated models.

higher accuracy than direct code generation. In MatX-Native, several models even approach near-perfect performance under tool invocation, while code generation accuracy remains extremely limited for most systems.

This discrepancy suggests that invoking predefined external tools is substantially easier than generating executable programs from scratch. Tool invocation primarily requires selecting appropriate APIs or operations under constrained action spaces, whereas code generation demands open-ended compositional program synthesis with precise logical and structural consistency. As task complexity increases in multi-table scientific reasoning scenarios, current models appear substantially more reliable under constrained execution frameworks than under unconstrained code synthesis settings.

4.6 MatX-Decompose vs. MatX-Native

Performance patterns differ substantially between **MatX-Decompose** and **MatX-Native**, indicating that benchmark structure significantly affects reasoning behaviors. Figure 5 shows the relative performance differences between the two benchmark settings across reasoning stages.

A consistent trend across models is degradation in `Field Selection`, `Condition Extraction`, and `Code Generation` under MatX-Native. Condition extraction shows the largest drop, suggesting difficulties in semantic grounding when multiple related tables and overlapping schema fields are involved. Similarly, code generation becomes less reliable in MatX-Native, reflecting challenges in maintaining logical consistency in complex multi-table settings.

In contrast, several models improve in `Join Selection` and especially `Tool Invocation` under MatX-Native. Compared to MatX-Decompose, MatX-Native preserves more complete schema structures and inter-table relationships, making dependencies more explicit. This additional relational context appears beneficial for constrained structural reasoning tasks, particularly when models only need to identify appropriate joins or invoke predefined tools under limited action spaces.

These results suggest that MatX-Native does not simply increase overall difficulty, but instead shifts the primary challenge from isolated retrieval toward coordinated multi-table executable reasoning. While preserved schema structures may benefit relational access and tool-oriented execution, they simultaneously introduce substantially greater ambiguity in fine-grained field grounding, condition alignment, and executable synthesis.

5 Conclusion

In this work, we introduce **MatXBench**, a multi-table question answering benchmark for scientific reasoning in the materials catalysis domain. Unlike existing table QA benchmarks that mainly focus on isolated table understanding or simplified schema settings, **MatXBench** explicitly evaluates cross-table retrieval, relational reasoning, SQL generation, and tool-based scientific execution under realistic multi-table workflows.

Through comprehensive evaluations of proprietary and open-source large language models, we find that current systems still face substantial challenges in fine-grained cross-table reasoning and executable task completion.

In particular, table selection emerges as a critical bottleneck in the overall pipeline, with larger performance variance across models than downstream reasoning stages, suggesting significant differences in table retrieval capability.

We also observe a large performance gap between tool invocation and direct code generation. Across nearly all evaluated settings, tool-based execution consistently achieves substantially higher performance than open-ended code synthesis, indicating that current models remain significantly more reliable under constrained execution settings than under unconstrained program generation scenarios.

6 Limitations

Although **MatXBench** provides a challenging benchmark for scientific multi-table reasoning, several limitations remain. First, the current benchmark focuses primarily on the materials catalysis domain, and future work may extend the framework to broader scientific disciplines involving heterogeneous multi-table reasoning. Second, although the benchmark covers multiple executable reasoning stages, certain analyses still rely on predefined decomposition steps and oracle intermediate supervision. Future work may further investigate fully end-to-end multi-table scientific agents under more realistic autonomous reasoning settings. Finally, the current benchmark mainly focuses on structured tabular reasoning, while additional scientific modalities such as molecular structures, experimental figures, and procedural descriptions remain outside the present scope.

We hope **MatXBench** can serve as a realistic and challenging evaluation platform for future research on scientific multi-table reasoning, executable knowledge grounding, and trustworthy LLM-based scientific agents.

References

2026. Gemini 3.1 Flash-Lite: Built for intelligence at scale. https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-3-1-flash-lite/.

Kareem S. Aggour, Vijay S. Kumar, Vipul K. Gupta, Alfredo Gabaldon, Paul Cuddihy, and Varish Mulwad. 2024. Semantics-Enabled Data Federation: Bringing Materials Scientists Closer to FAIR Data. *Integrating Materials and Manufacturing Innovation*, 13(2):420–434.

Mubashara Akhtar, Chenxi Pang, Andreea Marzoca, Yasemin Altun, and Julian Martin Eisenschlos. 2025. TANQ: An Open Domain Dataset of Table Answered Questions. *Transactions of the Association for Computational Linguistics*, 13:461–480.

Keith T. Butler, Daniel W. Davies, Hugh Cartwright, Olexandr Isayev, and Aron Walsh. 2018. Machine learning for molecular and materials science. *Nature*, 559(7715):547–555.

Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Yang Wang. 2020. HybridQA: A Dataset of Multi-Hop Question Answering over Tabular and Textual Data. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1026–1036, Online. Association for Computational Linguistics.

Ziyi Chen, Yang Yuan, Sihan Liang, Meng Wan, Kai Li, Weiqi Zhou, Yangang Wang, and Zongguo Wang. 2025. An automatic scientific data collection framework for materials science. *Computational Materials Science*, 252:113772.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. https://arxiv.org/abs/2501.12948v2.

Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2022. TURL: Table Understanding through Representation Learning. *SIGMOD Rec.*, 51(1):33–40.

Jun Fujima, Yuzuru Tanaka, Itsuki Miyazato, Lauren Takahashi, and Keisuke Takahashi. 2020. Catalyst Acquisition by Data Science (CADS): A web-based catalyst informatics platform for discovering catalysts. *Reaction Chemistry & Engineering*, 5(5):903–911.

Simon Gottschalk and Elena Demidova. 2022. Tab2KG: Semantic Table Interpretation with Lightweight Semantic Profiles. *Semantic Web*, 13(3):571–597.

Matthew K. Horton, Patrick Huck, Ruo Xi Yang, Jason M. Munro, Shyam Dwaraknath, Alex M. Ganose, Ryan S. Kingsbury, Mingjian Wen, Jimmy X. Shen, Tyler S. Mathis, Aaron D. Kaplan, Karlo Berket, Janosh Riebesell, Janine George, Andrew S. Rosen, Evan W. C. Spotte-Smith, Matthew J. McDermott, Orion A. Cohen, Alex Dunn, and 14 others. 2025. Accelerated data-driven materials science with the Materials Project. *Nature Materials*, 24(10):1522–1532.

Anubhav Jain, Shyue Ping Ong, Geoffroy Hautier, Wei Chen, William Davidson Richards, Stephen Dacek, Shreyas Cholia, Dan Gunter, David Skinner, Gerbrand Ceder, and Kristin A. Persson. 2013. Commentary: The Materials Project: A materials genome approach to accelerating materials innovation. *APL Materials*, 1(1):011002.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. 2023. StructGPT: A General Framework for Large Language Model to Reason over Structured Data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9237–9251, Singapore. Association for Computational Linguistics.

Scott Kirklin, James E Saal, Bryce Meredig, Alex Thompson, Jeff W Doak, Muratahan Aykol, Stephan Rühl, and Chris Wolverton. 2015. The Open Quantum Materials Database (OQMD): Assessing the accuracy of DFT formation energies. *npj Computational Materials*, 1(1):15010.

Rob Kitchin. 2014. Big Data, new epistemologies and paradigm shifts. *Big Data & Society*, 1(1):2053951714528481.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. *Preprint*, arXiv:2305.03111.

Tianyang Liu, Fei Wang, and Muhao Chen. 2024. Rethinking Tabular Data Understanding with Large Language Models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 450–482, Mexico City, Mexico. Association for Computational Linguistics.

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

Agada Joseph Oche and Arpan Biswas. 2025. Role of large language models and retrieval-augmented generation for accelerating crystalline material discovery: A systematic review. *ArXiv*, abs/2508.06691.

Vaishali Pal, Andrew Yates, Evangelos Kanoulas, and Maarten de Rijke. 2023. MultiTabQA: Generating Tabular Answers for Multi-Table Question Answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6322–6334, Toronto, Canada. Association for Computational Linguistics.

Changzai Pan, Jie Zhang, Kaiwen Wei, Chenshuo Pan, Yu Zhao, Jingwang Huang, Jian Yang, Zhenhe Wu, Haoyang Zeng, Xiaoyan Gu, Weichao Sun, Yanbo Zhai, Yujie Mao, Zhuoru Jiang, Jiang Zhong, Shuangyong Song, Yongxiang Li, and Zhongjiang He. 2026. ReasonTabQA: A Comprehensive Benchmark for Table Question Answering from Real World Industrial Scenarios. *arXiv*.

Sungho Park, Jueun Kim, and Wook-Shin Han. 2026. SPARTA: Scalable and Principled Benchmark of Tree-Structured Multi-hop QA over Text and Tables. *arXiv*.

Zipeng Qiu, You Peng, Guangxin He, Binhang Yuan, and Chen Wang. 2024. TQA-Bench: Evaluating LLMs for Multi-Table Question Answering with Scalable Context and Symbolic Extension. *Preprint*, arXiv:2411.19504.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-rong Wen. 2025. Tool learning with large language models: A survey. *Frontiers of Computer Science*, 19(8).

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. *Preprint*, arXiv:2302.04761.

Jonathan Schmidt, Tiago F.T. Cerqueira, Aldo H. Romero, Antoine Loew, Fabian Jäger, Hai-Chen Wang, Silvana Botti, and Miguel A.L. Marques. 2024. Improving machine-learning models in materials science through large datasets. *Materials Today Physics*, 48:101560.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, A. J. Ostrow, Akhila Ananthram, Akshay Nathan, Alan Luo, Alec Helyar, Aleksander Madry, Aleksandr Efremov, Aleksandra Spyra, Alex Baker-Whitcomb, Alex Beutel, Alex Karpenko, and 467 others. 2026. OpenAI GPT-5 System Card. *Preprint*, arXiv:2601.03267.

Keisuke Takahashi, Junya Ohyama, Shun Nishimura, Jun Fujima, Lauren Takahashi, Takeaki Uno, and Toshiaki Taniike. 2023. Catalysts informatics: Paradigm shift towards data-driven catalyst design. *Chemical Communications*, 59(16):2222–2238.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean-bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, and 197 others. 2025. Gemma 3 Technical Report. *Preprint*, arXiv:2503.19786.

Bing Tong, Yan Zhou, Chen Zhang, Jianheng Tang, Jia Li, and Lei Chen. 2025. GalaxyWeaver: Autonomous Table-to-Graph Conversion and Schema Optimization with Large Language Models. *Proceedings of the VLDB Endowment*, 18(12):5100–5112.

Takashi Toyao, Zen Maeno, Satoru Takakusagi, Takashi Kamachi, Ichigaku Takigawa, and Ken-ichi Shimizu. 2020. Machine Learning for Catalysis Informatics: Recent Applications and Prospects. *ACS Catalysis*, 10(3):2260–2297.

Antanas Vaitkus, Andrius Merkys, Thomas Sander, Miguel Quirós, Paul A. Thiessen, Evan E. Bolton, and Saulius Gražulis. 2023. A workflow for deriving chemical entities from crystallographic data and its application to the Crystallography Open Database. *Journal of Cheminformatics*, 15(1).

Xiaoxuan Wang, Ziniu Hu, Pan Lu, Yanqiao Zhu, Jieyu Zhang, Satyen Subramaniam, Arjun R. Loomba, Shichang Zhang, Yizhou Sun, and Wei Wang. 2024. SciBench: Evaluating College-Level Scientific Problem-Solving Abilities of Large Language Models. *Preprint*, arXiv:2307.10635.

Xixi Wang, Miguel Costa, Jordanka Kovaceva, Shuai Wang, and Francisco C. Pereira. 2025. Plugging Schema Graph into Multi-Table QA: A Human-Guided Framework for Reducing LLM Reliance. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 5829–5842.

Kirsten T. Winther, Max J. Hoffmann, Jacob R. Boes, Osman Mamun, Michal Bajdich, and Thomas Bligaard. 2019. Catalysis-Hub.org, an open electronic structure database for surface reactions. *Scientific Data*, 6(1):75.

Jian Wu, Linyi Yang, Dongyuan Li, Yuliang Ji, Manabu Okumura, and Yue Zhang. 2025. MMQA: Evaluating LLMs with Multi-Table Multi-Hop Complex Questions. *International Conference on Learning Representations*, 2025:48626–48643.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 Technical Report. *Preprint*, arXiv:2505.09388.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Siwei Zhang, Xuan Wang, and Hongtao Huang. 2019. Research on the Factors Affecting the "Internet +" Reading Effects Among Primary and Middle School Students. In *2019 International Symposium on Educational Technology (ISET)*, pages 293–296.

Yong Zhao, Mohammed Al-Fahdi, Ming Hu, Edirisuriya M. D. Siriwardane, Yuqi Song, Alireza Nasiri, and Jianjun Hu. 2021. High-Throughput Discovery of Novel Cubic Crystal Materials Using Deep Generative Neural Networks. *Advanced Science*, 8(20):2100566.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *Preprint*, arXiv:1709.00103.

Jun Zhou, Lei Shen, Miguel Dias Costa, Kristin A. Persson, Shyue Ping Ong, Patrick Huck, Yunhao Lu, Xiaoyang Ma, Yiming Chen, Hanmei Tang, and Yuan Ping Feng. 2019. 2DMatPedia, an open computational database of two-dimensional materials from top-down and bottom-up approaches. *Scientific Data*, 6(1):86.

Wei Zhou, Mohsen Mesgar, Annemarie Friedrich, and Heike Adel. 2025. Efficient Multi-Agent Collaboration with Tool Use for Online Planning in Complex Table Question Answering. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 945–968, Albuquerque, New Mexico. Association for Computational Linguistics.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021. TAT-QA: A Question Answering Benchmark on a Hybrid of Tabular and Textual Content in Finance. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3277–3287, Online. Association for Computational Linguistics.

Jun-Peng Zhu, Peng Cai, Kai Xu, Li Li, Yishen Sun, Shuai Zhou, Haihuang Su, Liu Tang, and Qi Liu. 2024. AutoTQA: Towards Autonomous Tabular Question Answering through Multi-Agent Large Language Models. *Proceedings of the VLDB Endowment*, 17(12):3920–3933.

Fatima Zivic, Ana Kaplarevic Malisic, Nenad Grujovic, Boban Stojanovic, and Milos Ivanovic. 2025. Materials informatics: A review of AI and machine learning tools, platforms, data repositories, and applications to architectured porous materials. *Materials Today Communications*, 48:113525.

Jiaru Zou, Dongqi Fu, Sirui Chen, Xinrui He, Zihao Li, Yada Zhu, Jiawei Han, and Jingrui He. 2025. GTR: Graph-Table-RAG for Cross-Table Question Answering. *Preprint*, arXiv:2504.01346.

A Data Construction

A.1 Data Source

We have retrieved data from multiple open-source databases. The following lists the major categories of data sources with large volumes. For the complete data sources, please refer to the tabular metadata in our GitHub repository.

- **Catalyst Acquisition by Data Science (CADS):** An integrated catalyst informatics platform designed to support catalyst discovery and design (Fujima et al., 2020).
- **Catalytic Material Database (CMD):** A database containing material compositions, properties, reactions, products, and related catalytic information (Zhao et al., 2021).
- **Catalyst Hub:** A database for surface reactions containing large-scale chemisorption and reaction energy data obtained from electronic structure calculations (Winther et al., 2019).
- **Crystallography Open Database (COD):** An open-access collection of crystal structures for organic, inorganic, metal-organic compounds, and minerals (Vaitkus et al., 2023).
- **Materials Project:** A platform providing computed information and analysis tools for known and predicted materials (Horton et al., 2025).
- **2DMatPedia:** A dataset containing a large collection of two-dimensional materials (Zhou et al., 2019).
- **Alexandria_DB PBE 3D:** A dataset of stable and metastable materials computed using the PBE functional (Schmidt et al., 2024).
- **OQMD-3D:** A database containing density functional theory (DFT) calculated thermodynamic and structural properties of materials (Kirklin et al., 2015).
- **Associated Data from Papers:** Additional material-related datasets collected from published scientific papers.

A.2 Table Annotation Instance

We annotated each dataset using the following schema:

Dataset Annotation Example

Tabular Name: `table1`

Tabular Description:
This table evaluates Mo-doped $BaTiO_3$ photocatalysts for hydrogen production by analyzing the relationship between Mo doping levels, band gap energy (E_g), and hydrogen evolution activity (RH_2). The dataset supports the identification of optimal dopant concentrations for improving photocatalytic efficiency in solar-driven water splitting.

Field Description:

1. **[Molecular formula]** – elemental composition of the compound
2. **[RH_2 (μmol h^{-1} g^{-1})]** – hydrogen production rate
3. **[E_g (eV)]** – band gap energy
4. **[Preparation method]** – synthesis method of the material
5. **[Calcination temperature (K)]** – calcination temperature used during synthesis
6. **[Calcination time (h)]** – calcination duration used during synthesis
7. **[Light intensity (W)]** – power of the illumination source
8. **[Reaction solution]** – chemical solution used in the reaction
9. **[Co-catalyst]** – auxiliary catalyst used with the photocatalyst
10. **[Photocatalyst dose (g L^{-1})]** – concentration of photocatalyst in the reaction system
11. **[Ref]** – reference source of the experimental data

A.3 Subdomain Distribution

To illustrate the scientific domain coverage of MatXBench, Figure 6 presents the distribution of

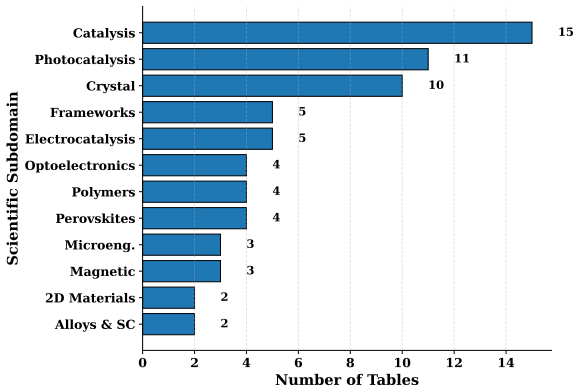

Figure 6: Distribution of tables across different scientific subdomains in MatXBench.

tables across different subdomains in the collected dataset. The benchmark covers a broad range of materials science topics, including catalysis, crystal materials, low-dimensional materials, optoelectronic materials, polymers, and porous frameworks.

For readability, several domain names are abbreviated in the figure. Specifically, the abbreviations correspond to the following full domain names: "Catalysis" (Catalysis and Photoelectrocatalysis), "Alloys & SC" (Functional Alloys and Superconductors), "Crystal" (Inorganic Crystal Fundamentals), "2D Materials" (Low-Dimensional / Two-Dimensional Materials), "Microeng." (Microengineering and Synthesis), "Optoelectronics" (Optoelectronic and Energy Materials), "Polymers" (Polymers and Molecular Systems), and "Frameworks" (Porous Framework Materials).

A.4 Question Type Distribution

Table 5 summarizes the distribution of question types across different benchmark settings. We categorize questions into three major types: `search query`, `aggregation`, and `science compute`.

Table 5: Distribution of question types across different benchmark settings.

Benchmark	Search	Agg.	Sci. Compute
MatX-Decompose	21,153	15,532	2,836
MatX-Native	6,138	1,352	1,723

B Question & Tool Generation

B.1 Template Generation

As illustrated in Figure 2, after the cluster construction stage (Step 4), each cluster serves as the basic unit for question generation. For **MatX-Decompose**, clusters are derived from a direct column-wise splitting strategy, where each cluster typically corresponds to approximately two tables. In contrast, **MatX-Native** constructs clusters using three different generation strategies (as shown in Step 4 of Figure 2), resulting in more heterogeneous clusters, each containing approximately 4–10 tables. This design introduces varying degrees of schema complexity and cross-table interaction requirements across the two benchmark settings.

For each cluster, we first collect all associated table metadata, including *table descriptions*, *field names*, and *field-level descriptions*. These structured inputs are then provided to GPT-4o to generate candidate cross-table question templates. Specifically, the model is required to select two tables within a cluster and construct a cross-table question together with its corresponding SQL template.

Each cluster is passed through the generation pipeline multiple times to increase coverage of possible question templates. Specifically, clusters in MatX-Native are typically processed around 20 times, whereas clusters in MatX-Decompose are processed approximately 5 times. This repeated sampling strategy is designed to improve the diversity of generated cross-table questions within each cluster. To avoid redundancy, each newly generated question template is compared against the existing template pool, and duplicates are filtered out based on semantic and structural similarity checks to ensure uniqueness.

B.2 Diversity Control

However, we observe a strong tendency that when the same cluster is repeatedly fed into the generation pipeline, the model consistently selects a small subset of table pairs, even under relatively high temperature settings. This behavior significantly reduces the diversity of generated question templates within the same cluster.

To mitigate this issue, we introduce a simple but effective intervention strategy. If a specific table pair appears more than three times within repeated generations from the same cluster, this pair is temporarily removed from the candidate set for subsequent generations. This forces the model to explore alternative table combinations within the cluster, thereby improving the diversity and coverage of generated cross-table reasoning scenarios. This mechanism encourages broader exploration of

table combinations and reduces structural concentration during repeated template generation.

B.3 Template Refinement

To ensure quality and scientific validity, each generated question template is further evaluated by GPT-5, which assigns scores and provides textual feedback from multiple perspectives. The evaluation emphasizes both scientific correctness and cross-table reasoning necessity. Based on these feedback signals, the question templates are revised by GPT-4o and re-evaluated by GPT-5. Only templates that achieve a score of at least 3 across all evaluation dimensions are retained in the final dataset.

Search Query Question Templates. For search query-based questions, the evaluation focuses on: *scientific validity*, *cross-table necessity*, *slot design semantics*, *slot-to-table mapping and join rationality*, and *SQL answer specification correctness*, in addition to an overall assessment.

Aggregation Question Templates. For aggregation-based questions, the evaluation considers: *scientific validity*, *cross-table necessity*, *slot design semantics*, *slot-to-table mapping and join rationality*, and *aggregation answer specification correctness*, together with an overall assessment.

Science Compute Question Templates. For science computation questions, the evaluation includes: *scientific validity*, *cross-table necessity*, *tool necessity*, *slot design semantics*, *slot-to-table mapping and join rationality*, *tool input mapping correctness*, and *answer specification correctness*, along with an overall assessment.

After the refinement stage, the retained templates are instantiated using real tabular data. Specifically, slot values are populated from the corresponding database instances, and the generated SQL templates are executed to produce concrete question-answer pairs.

Only instantiated samples with valid executable outputs are preserved in the final benchmark. Templates that produce empty execution results, invalid SQL behavior, or inconsistent slot mappings during instantiation are discarded.

B.4 Representative Failure Cases

During large-scale template generation, we observe several recurring failure patterns produced by LLM-based generation pipelines. Although these templates may appear structurally reasonable at the natural language level, they often contain invalid schema assumptions, unsupported joins, or unrealistic scientific relations. Representative examples are shown below.

Invalid Join Construction. *What is the bandgap energy value in eV for the photocatalytic material characterized by the calcination temperature of {temperature}°C and reference code {reference}?*

This template attempts to associate two tables through reference-related fields. However, although both tables contain citation-related columns, their reference formats and citation contents are inconsistent and do not correspond to the same underlying entities. As a result, the generated join condition is semantically invalid despite appearing syntactically plausible.

Hallucinated Schema Fields. *What is the Shannon ionic radii for host A-site atoms in the material with the chemical formula {Material Composition} that has a PBE calculated band gap of {PBE_gap}?*

In this case, the generated template introduces a non-existent field (`Material Composition`) that is not present in the associated tables. This failure reflects schema hallucination behavior, where the model generates scientifically plausible but unsupported field names based on prior knowledge rather than the actual schema definitions.

Lack of Real Entity Correspondence. *What is the bulk modulus of the compound with the chemical formula {compound_formula} that falls under the chemical sub-family {sub_family}?*

This template attempts to connect magnetic materials tables with inorganic crystal tables through material composition information. However, the corresponding tables do not share overlapping material entities in practice. Consequently, although the question appears scientifically meaningful, no executable or semantically grounded cross-table relation exists between the selected tables.

These failure cases highlight that generating scientifically valid cross-table questions requires more than syntactic SQL construction. Effective template generation additionally depends on faithful schema grounding, valid entity overlap, and scientifically meaningful relational constraints across heterogeneous scientific tables.

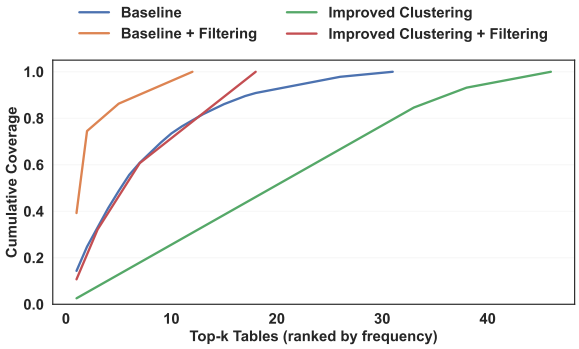

Figure 7: Pareto-style cumulative coverage of table usage distribution. The curve shows how many top-ranked tables are required to cover the majority of usage frequency under different clustering and filtering strategies.

Setting	Entropy	Gini
Baseline	2.97	0.53
Baseline + Filtering	1.66	0.62
Improved Clustering	3.77	0.14
Improved Clustering + Filtering	2.78	0.24

Table 6: Entropy and Gini coefficient of table usage distributions under different generation and filtering strategies.

B.5 Distributional Analysis

To further analyze the distributional properties of table usage within generated question templates, we study the coverage characteristics of table selection patterns. Figure 7 presents a Pareto-style cumulative coverage curve, where tables are ranked by frequency and the cumulative proportion of total usage is plotted against the number of top-ranked tables.

The baseline setting shows a relatively concentrated distribution, where a small subset of tables accounts for a large proportion of total usage. In contrast, the *improved clustering* strategy significantly increases distributional entropy, indicating that table usage becomes more evenly distributed across a larger number of tables. However, when the filtering mechanism is applied, the distribution becomes more concentrated again, reflecting the removal of low-frequency but structurally diverse table combinations.

To quantify these effects more precisely, we report entropy and Gini coefficients under different configurations in Table 6.

As shown in Table 6, improved clustering substantially increases entropy and reduces inequality in table usage distribution (lower Gini coefficient), indicating more balanced coverage across tables.

Overall, these results demonstrate that our clustering and filtering strategies provide a controllable trade-off between coverage diversity and distributional concentration in question generation, enabling more flexible control over dataset construction properties.

B.6 Tool Construction

In addition to question template generation, we further construct a set of executable scientific tools to support `science_compute` questions in MatXBench. These tools are designed to simulate realistic scientific computation workflows and enable models to perform executable reasoning beyond standard SQL-based retrieval.

Each generated tool is associated with a representative usage case, which serves as a standardized example for both automated and manual evaluation. These cases are designed to reflect realistic scientific scenarios and allow annotators to inspect tool behavior, input-output consistency, and scientific plausibility under controlled settings.

All tools are first evaluated through a multi-dimensional GPT-5 based assessment framework, covering: *scientific validity*, *cross-table necessity*, *tool necessity*, *slot design semantics*, *slot-to-table mapping and join rationality*, *tool input mapping correctness*, and *answer specification correctness*. This stage provides fine-grained diagnostic feedback on both functional correctness and reasoning consistency.

Subsequently, each tool is independently reviewed by human experts, who assess it from three complementary perspectives: *complexity*, *scientific relevance*, and *practical applicability*. Based on the combined feedback from GPT-5 and human annotators, we iteratively refine tool definitions to improve their alignment with domain-specific scientific workflows. In particular, expert suggestions are incorporated to enhance domain adaptability, improve input-output clarity, and ensure consistency with realistic scientific procedures.

Only tools that pass both automated evaluation thresholds and human verification are retained in the final benchmark, ensuring high-quality, scientifically meaningful, and executable tool definitions.

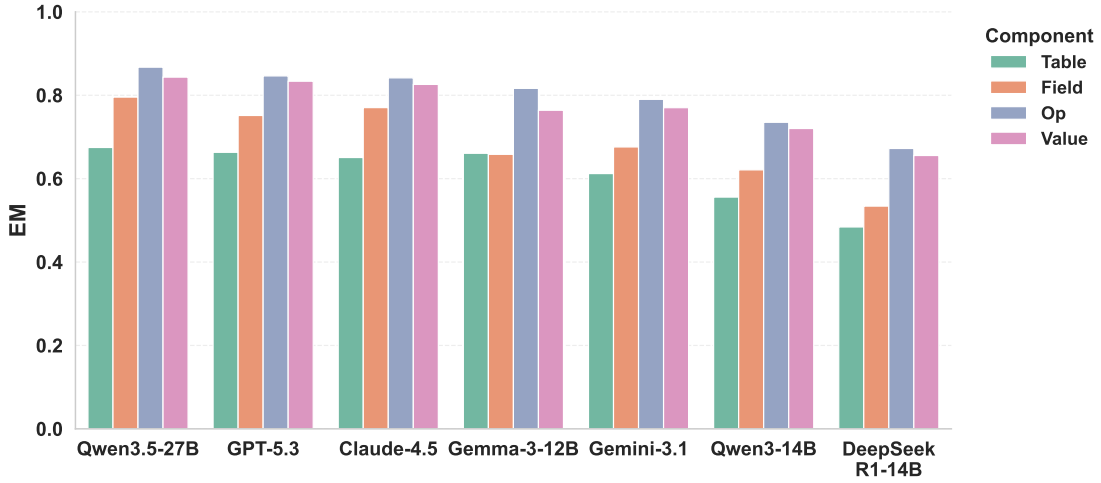

Figure 8: Fine-grained condition extraction accuracy across different structural components and evaluated models. Each extracted condition is decomposed into table, field, operator, and value components for detailed error analysis.

C Analysis

C.1 Condition Extraction Error Analysis

To further analyze model behavior in the condition extraction stage, we decompose each extracted condition into four structural components: `Table`, `Field`, `Operator (Op)`, and `Value`. This analysis focuses specifically on the condition extraction task itself rather than downstream SQL generation, aiming to identify which semantic components are most challenging for current models during structured condition grounding.

Figure 8 presents the component-level extraction accuracy across different evaluated models.

Overall, operator and value extraction are consistently more reliable than table and field grounding. Across most evaluated models, `Op` achieves the highest accuracy, often exceeding 0.8, while `Table` remains the most challenging component. This trend suggests that models can usually identify common comparison patterns and numerical constraints correctly, but still struggle to precisely associate conditions with the correct relational context and schema locations.

In addition, stronger proprietary models such as GPT-5.3 and Claude-4.5 achieve relatively balanced performance across all components, whereas smaller open-source models exhibit substantially larger degradation in table and field grounding accuracy. These findings suggest that accurately grounding conditions to the correct tables and fields remains substantially more challenging than identifying operators or values.

C.2 Performance differences between MatX-Decompose and MatX-Native

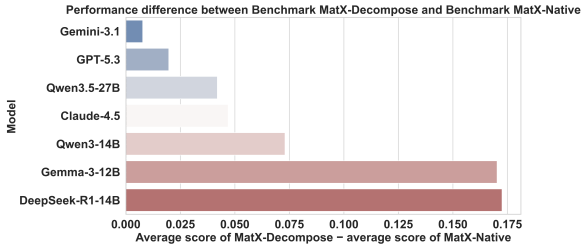

Figure 9: Performance differences between MatX-Decompose and MatX-Native across evaluated models. Positive values indicate that a model achieves a higher average score on MatX-Decompose than on MatX-Native, suggesting that MatX-Native is generally more challenging.

As shown in Fig. 9, all evaluated models exhibit positive average performance differences between MatX-Decompose and MatX-Native, measured as $MatX-Decompose-MatX-Native$. This indicates that the models consistently achieve higher scores on MatX-Decompose than on MatX-Native, suggesting that MatX-Native is more challenging or imposes stricter requirements on cross-table querying capabilities. The magnitude of the $MatX - Decompose - MatX - Native$ gap further reveals model-specific robustness across benchmarks: `Deepseek-R1-14B` and `Gemma-3-12b` show the largest performance drops, indicating greater sensitivity to benchmark difficulty, whereas `GPT-5.3` and `Gemini-3.1` exhibit much smaller gaps, suggesting more stable cross-benchmark generalization. Fine-grained metric

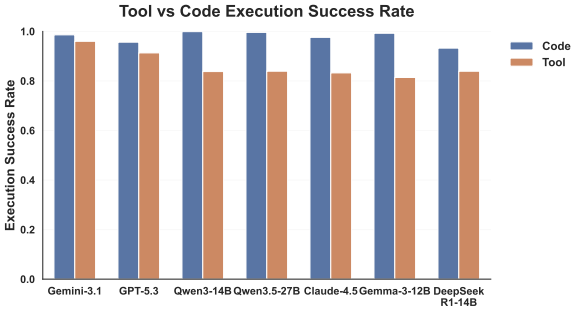

Figure 10: Execution success rates of tool invocation and code generation across different models. Although code generation achieves slightly higher raw execution success rates, tool-based execution yields more reliable correct-answer outcomes.

inspection further shows that several models can still maintain high scores in `sql_generation`, `tool_invocation`, and `code_generation` on MatX-Native, while their performance decreases substantially in upstream Schema Linking components such as `table_select` and `field_select`. Therefore, the main challenge of MatX-Native appears to lie not in SQL or code generation itself, but in structured understanding, including table selection, field grounding, and cross-table semantic alignment.

C.3 Execution Success Rate: Tool vs. Code

To further analyze execution reliability, we report the raw execution success rates of **Tool Invocation** and **Code Generation** across different models in Figure 10.

Interestingly, we observe a consistent pattern that **code generation achieves higher execution success rates than tool invocation across most models**. For instance, models such as Qwen3-14B and Qwen3.5-27B nearly reach perfect execution success under code generation settings. This indicates that modern LLMs are highly capable of producing syntactically valid code when operating in constrained programming environments.

However, execution success alone does not translate into correct reasoning outcomes. Despite lower execution success rates, **tool-based execution consistently achieves higher end-task accuracy (EM)** compared to code generation in the main evaluation results. This suggests that tool invocation, although more restrictive and sometimes less syntactically robust, provides stronger inductive bias and constrained action spaces that better align with the underlying task semantics.

D Prompt

Table Selection Prompt

```
[System Prompt]

You are an expert in multi-table question
    answering and database schema
    understanding.

Your task is to identify exactly TWO
    tables that are most relevant to
    answer a given user query, based on
    the provided table metadata.

You will be given:
- A natural language query
- A list of table information, where each
    table includes:
  - table_name
  - table_description
  - field_description

Instructions:
1. Carefully analyze the query and
    understand what information is
    required to answer it.
2. Examine the table descriptions and
    field descriptions to determine
    which tables contain the necessary
    data.
3. Select exactly TWO distinct tables
    that together can best answer the
    query.
4. Prioritize tables that:
   - Contain complementary information
    required by the query
   - Have relevant fields directly
    related to the query intent
5. Do NOT select more or fewer than two
    tables.

Output Format:
Return the result strictly as a Python
    list of strings:

["table_name_1", "table_name_2"]

Constraints:
- Do NOT include any explanations,
    reasoning, or additional text.
- Do NOT return anything other than the
    list.
- Table names must exactly match those
    provided in the input.

Example Output:
["table1", "table3"]

[User Prompt Template]

<query>
{{query}}

<available_tables_information>
{{available_tables_information}}
```

Field Selection Prompt

```
[System Prompt]

You are an expert in database schema
    understanding and multi-table
    question answering.

Your task is to identify all relevant
    columns (fields) needed to answer a
    given query, based on the provided
    table metadata.

You will be given:
- A natural language query
- A list of table information, where each
    table includes:
  - table_name
  - table_description
  - field_description

Instructions:
1. Carefully analyze the query to
    understand what information is
    required.
2. Examine each tables description and
    field descriptions.
3. Identify all columns that are
    necessary to answer the query.
4. For each relevant table:
   - Select only the fields that are
    directly or indirectly required.
   - Always include "key_id" for every
    selected table.
5. Include all relevant tables, not
    limited to a fixed number.
6. Do NOT include irrelevant fields or
    tables.

Output Format:
Return the result strictly as a Python
    dictionary:

{"table_name_1": ["field1", "field2"],
    "table_name_2": ["fieldA", "fieldB"}

Constraints:
- Do NOT include explanations, reasoning,
    or extra text.
- Do NOT return anything other than the
    dictionary.
- Field names and table names must
    exactly match those provided.
- Each table must contain "key_id" in its
    field list.
- Do NOT duplicate fields within the same
    table.

Example Output:
{"table107": ["Acid_OVP", "key_id"],
    "table63": ["DOI", "Facility",
    "key_id"]}

[User Prompt Template]

<query>
{{query}}

<relative tables information>
{{relative_tables_information}}
```

Join Selection Prompt

```
[System Prompt]

You are an expert in database schema
    reasoning and multi-table query
    planning.

Your task is to identify the join paths
    between tables required to answer a
    given query, based on the provided
    table metadata and selected tables.

You will be given:
- A natural language query
- A list of table information, where each
    table includes:
  - table_name
  - table_description
  - field_description
- A list of selected tables that are
    relevant to the query

Instructions:
1. Analyze the selected tables and
    determine how they can be joined to
    answer the query.
2. Identify join relationships based on
    shared keys, common identifiers, or
    semantically matching fields (e.g.,
    "key_id").
3. Each join must connect two tables
    using a pair of fields.
4. Prefer joins using primary keys or
    clearly matching fields (e.g.,
    identical names or described as
    identifiers).
5. If multiple join paths are possible,
    choose the most direct and reliable
    one.
6. Ensure that all selected tables are
    connected in a valid join path
    (i.e., the result forms a connected
    graph).
7. Do NOT invent fields or relationships
    that are not supported by the schema.

Output Format:
Return the result strictly as a Python
    list of dictionaries:

[
  {
    "left_table": "table_name_1",
    "right_table": "table_name_2",
    "left_field": "field_name",
    "right_field": "field_name"
  }
]

Constraints:
- Do NOT include explanations or
    reasoning.
- Do NOT return anything other than the
    list.
```

```
- Table names and field names must
    exactly match the provided schema.
- Do NOT duplicate joins.
- The joins must connect all selected
    tables.

Example Output:
[
  {
    "left_table": "table1",
    "right_table": "table2",
    "left_field": "key_id",
    "right_field": "key_id"
  }
]

[User Prompt Template]

<query>
{{query}}

<relative tables information>
{{relative_tables_information}}
```

Condition Extract Prompt

```
[System Prompt]

You are an expert in database querying
    and information extraction.

Your task is to extract all filtering
    conditions from a natural language
    query, based on the provided table
    metadata.

You will be given:
- A natural language query
- A list of table information, where each
    table includes:
  - table_name
  - table_description
  - field_description

Instructions:
1. Carefully analyze the query and
    identify all filtering conditions
    (constraints) required to answer the
    query.
2. Each condition must correspond to a
    specific field in a specific table.
3. For each condition, extract:
   - table: the table name
   - field: the field name
   - operator: one of (=, >, <, >=, <=,
    !=)
   - value: the value to compare against
    (number or string)
4. Only include conditions that are
    explicitly or implicitly required by
    the query.
5. Do NOT include projection fields
    (i.e., fields that are only used for
    output).
6. If a condition involves multiple
    tables, choose the most relevant
    table based on field description.
7. If no filtering condition exists,
    return an empty list.

Output Format:
Return the result strictly as a Python
    list of dictionaries:

[
  {
    "table": "table_name",
    "field": "field_name",
    "operator": "operator",
    "value": value
  }
]

Constraints:
- Do NOT include explanations or
    reasoning.
- Do NOT return anything other than the
    list.
- Table names and field names must
    exactly match the provided schema.
- Do NOT hallucinate fields or tables
    that do not exist.

Example Output:
[
  {
    "table": "table1",
    "field": "Temp. ()",
    "operator": ">",
    "value": 700
  }
]

[User Prompt Template]

<query>
{{query}}

<relative tables information>
{{relative_tables_information}}
```

SQL GENERATION Prompt

```
[System Prompt]

You are an expert in SQL generation and
    multi-table query construction.

Your task is to generate a correct SQL
    query based on the provided
    information.

You will be given:
- A natural language query
- A list of selected fields in the format:
  {"table_name": ["field1", "field2",
    ...]}
- A list of join paths
- A list of filtering conditions
- Table metadata, where each table
    includes:
  - table_name
  - table_description
  - field_description
```

```
Instructions:
1. Construct a valid SQL query that
     answers the given query.
2. Use ONLY the provided tables and
     fields. Do NOT introduce new tables
     or fields.
3. The SELECT clause must include all
     fields specified in the selected
     fields input.
4. Use table_name.field_name format for
     all columns to avoid ambiguity.
5. Construct JOIN clauses strictly based
     on the provided join paths:
   - Each join connects two tables.
   - If multiple conditions exist for a
     join, combine them using AND.
6. Construct the WHERE clause using the
     provided filtering conditions:
   - Each condition includes table,
     field, operator, and value.
   - Combine multiple conditions using
     AND unless otherwise specified.
7. Ensure the query is executable and
     logically correct.
8. Do NOT include unnecessary fields,
     joins, or conditions.
9. Do NOT hallucinate any schema elements.

Output Format:
Return ONLY the SQL query as a string.

Constraints:
- Do NOT include explanations, comments,
     or formatting outside the SQL.
- Do NOT use fields or tables not present
     in the inputs.
- All table and field names must exactly
     match the provided inputs.
- Ensure proper SQL syntax.

Example Output:
SELECT table1."Temp. ()",
     table2.production_rate
FROM table1
JOIN table2 ON table1.key_id =
     table2.key_id
WHERE table1."Temp. ()" > 700 AND
     table2.production_rate >= 100

[User Prompt Template]

<query>
{{query}}

<relative tables information>
{{relative_tables_information}}

<relative fields>
{{relative_fields}}

<join paths>
{{join_paths}}

<filtering conditions>
{{filtering_conditions}}
```

Tool Invocation Prompt

```
[System Prompt]

You are an expert in tool invocation
     planning for science_compute tasks.

Your task is to select the most
     appropriate tool and define how to
     apply it based on the given
     question, SQL query, and SQL result
     summary.

You will be given:
- A natural language question
- A SQL query
- A summary of the SQL execution results
- A list of available tools, where each
     tool includes:
  - tool_name
  - input parameters
  - output fields

Instructions:
1. Carefully analyze the question and
     determine what computation is
     required.
2. Review the SQL query and result
     summary to understand what data is
     available.
3. Select exactly ONE tool that best fits
     the required computation.
4. Choose an appropriate aggregation
     method based on the question intent.
     Common options include:
   - "mean" (average)
   - "max" (maximum)
   - "min" (minimum)
   - "sum" (total)
   - "count" (number of records)
5. If the selected tool has multiple
     output fields, choose the most
     appropriate one to aggregate.
6. Map tool input parameters to SQL
     result columns whenever possible.
7. If a required tool input cannot be
     mapped from SQL results, provide it
     as a constant value.
8. If uncertain about a parameter, leave
     it unmapped.

Output Format:
Return the result strictly as a JSON
     object with the following schema:

{
  "tool_name": "...",
  "aggregation": "...",
  "output_field": "...",
  "parameter_mapping": {
    "tool_input_name": "sql_column_name"
  },
  "constant_arguments": {
    "tool_input_name": "literal_value"
  }
}

Constraints:
- You MUST select exactly one tool from
```

```
    the provided tools.
- "aggregation" MUST be a valid
    aggregation operation (e.g., mean,
    max, min, sum, count).
- "output_field" MUST be one of the
    selected tool's output fields; if
    uncertain, choose the first one.
- Keys in "parameter_mapping" MUST be
    valid tool input names.
- Values in "parameter_mapping" MUST be
    column names from the SQL result.
- Use "constant_arguments" only for
    inputs that cannot be derived from
    SQL results.
- Do NOT include any explanations,
    reasoning, or additional text.
- Return ONLY the JSON object.

[User Prompt Template]

<question>
{{question}}

<sql>
{{sql}}

<sql_result_summary>
{{result_summary}}

<available_tools>
{{available_tools}}
```

Code Generation Prompt

```
[System Prompt]

You are an expert in Python programming
    and data analysis using pandas.

Your task is to generate a Python
    function that computes the final
    answer to a query based on a SQL
    result dataframe summary.

You will be given:
- A natural language query
- The SQL query that was executed
- A summarized view of the SQL result
    dataframe (shape, dtypes,
    statistics, and sample rows)

Instructions:
1. Generate a Python function with EXACT
    signature:

   def solve(df: pandas.DataFrame):

2. The function should compute and return
    the final answer required by the
    query.
3. Use only columns shown in the
    dataframe summary.
4. Use robust pandas operations and guard
    against empty/invalid values.
5. If no valid answer can be computed,
    return None.

Output format:
- Return ONLY Python function code.
- Do not include explanation text.

[User Prompt Template]

<question>
{query}

<sql>
{sql}

<sql_result_summary>
{result_summary}
```

E Responsible Research and Compliance Statement

E.1 Licensing and Usage

MatXBench is constructed from publicly available scientific resources and is intended solely for academic research purposes. The benchmark annotations and generated question-answer pairs are released under the CC BY 4.0 license. Users are responsible for complying with the original licenses and terms of use associated with the underlying source data.

E.2 Data and Code Availability

The source code and benchmark implementation are publicly available at:https://anonymous.4open.science/r/MatXBench-2026

The full benchmark dataset, including question-answer pairs, table metadata, executable tools, and evaluation annotations, is available at:https://zenodo.org/records/20390391?token=eyJhbGciOiJIUzUxMiJ9.eyJpZCI6IjY3NjI5YTVkLTk3YjctNDRhYy04ZjFlLTlkMTUzMTI3MjMyZCIsImRhdGEiOnt9LCJyYW5kb20iOiIyMDJhZTg1OGQ3OTU0NWY5M2MxYWY1NzgwYjVlNDIwNSJ9.BqF_jAR-50OclBTXuzIPbUTP1oaIj-AqGTEqyXQCPTk9hja4kLw-OvXrBpaQHl77okfKcOgei0QqctlHmoO4ig

The released resources include evaluation and execution pipelines, annotated table metadata, generated question templates, executable scientific tools, and the complete benchmark splits for both MatX-Decompose and MatX-Native.

E.3 Use of AI Assistants

AI assistants were used for language polishing and minor coding assistance. All scientific content, experimental design, and conclusions were verified by the authors.