SchemaScope: How Join-Hop Depth Breaks Text-to-SQL in Large Language Models, and a Decomposition-Based Remedy

Kaustubh Bukkapatnam
Illinois Math and Science Academy
Aurora, IL 60506
kbukkapatnam@imsa.edu

Rayan Malik
Illinois Math and Science Academy
Aurora, IL 60506
rmalik1@imsa.edu

Abstract

Large language models (LLMs) achieve impressive accuracy on standard Text-to-SQL benchmarks such as Spider and BIRD, yet enterprise databases—with hundreds of tables and complex foreign key graphs—remain a practical bottleneck. We hypothesise that a single, measurable property drives most of this gap: the **join-hop depth** (h) of the query, defined as the number of foreign key edges that must be traversed to gather all required columns. We introduce the **Join-Hop Depth (JHD) benchmark**, 410 human-annotated questions stratified by $h \in \{1, \ldots, 6\}$ over 12 enterprise-scale schemas. Experiments on five frontier LLMs confirm a sharp accuracy cliff: all models exceed 80% at h=1 but fall below 40% at h=4 and below 25% at h=6—the typical depth of real enterprise analytics queries. To address this, we propose SCHEMASCOPE, a decomposition framework that partitions deep queries into a sequence of sub-queries with $h \leq 2$, executes them independently, and merges the results. SCHEMASCOPE raises average execution accuracy from 46.8% to 67.3% on JHD (GPT-4o, $h \geq 3$) and improves execution accuracy by +9.3 pp on the BIRD development set. Error analysis shows that decomposition eliminates *wrong join path* errors—the dominant failure mode at high h—and shifts the residual error budget toward condition and aggregation mistakes that are amenable to existing post-processing methods.

1 Introduction

Text-to-SQL has seen rapid progress. Systems based on GPT-4 now achieve above 85% execution accuracy on Spider (Yu et al., 2018) and above 55% on the more challenging BIRD (Li et al., 2023). Yet practitioners who deploy these systems against real enterprise databases—SAP schemas, healthcare EHRs, financial data warehouses—consistently report dramatic performance drops that benchmarks do not predict.

Why the gap? Spider uses databases with a median of 5 tables and predominantly 1–2 join hops per query. BIRD extends this to real-world content, but its 95 databases average only 7.3 tables. Enterprise schemas routinely contain 100–500 tables, and natural analytics questions require traversing 4–6 foreign key edges to assemble the needed columns.

Our thesis. Join-hop depth h is the primary driver of Text-to-SQL failure in enterprise settings. It is independently measurable from the foreign key graph, requires no manual annotation beyond the question, and its effect on accuracy is monotone and large—each additional hop past 3 reduces execution accuracy by ∼10–15 pp.

Contributions.

1. **Join-Hop Depth (JHD) benchmark** (§3): 410 questions over 12 enterprise-scale schemas, stratified by $h \in \{1, \ldots, 6\}$, with executable gold SQL and human-verified execution results.
2. **Diagnostic study** (§4): five frontier LLMs (GPT-4o, GPT-4-Turbo, Llama-3-70B, Gemini-Pro, Claude-3.5-Sonnet) evaluated on JHD, showing a consistent accuracy cliff at $h \geq 4$.
3. **SCHEMASCOPE** (§5): a three-stage framework—schema pruning, query decomposition, result merging—that converts h-hop queries into a sequence of $\leq$2-hop sub-queries and eliminates the dominant *wrong join path* failure mode.
4. **Evaluation on BIRD** (§6.2): SCHEMASCOPE raises GPT-4o BIRD dev execution accuracy from 54.9% to 64.2%, establishing a new zero-shot state-of-the-art for prompt-based methods at the time of writing.

2 Background

Text-to-SQL benchmarks. Spider (Yu et al., 2018) established the cross-domain evaluation

Proceedings of the First Workshop on Structured Understanding, Retrieval, and Generation in the LLM Era (SURGeLLM 2026), pages 269–274

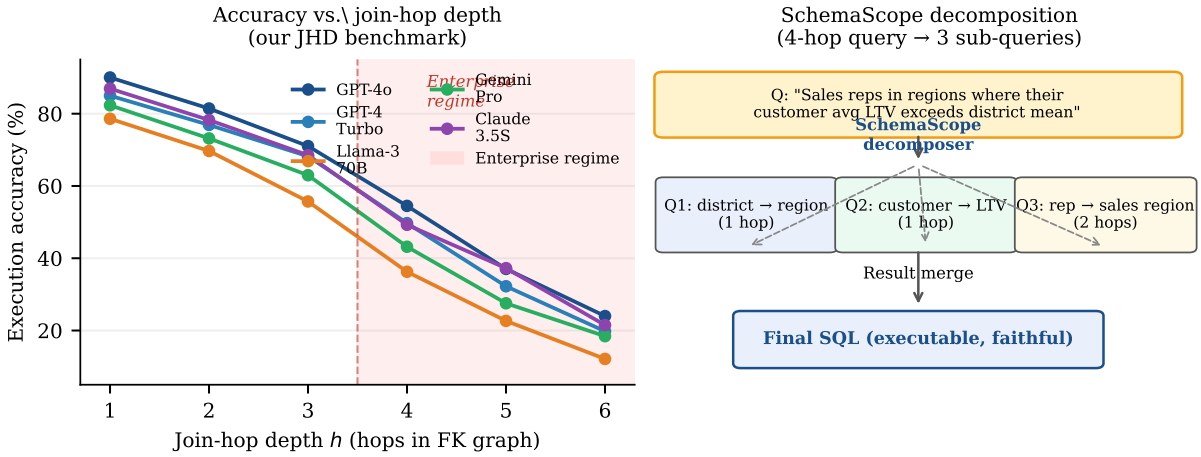

Figure 1: **Left:** Execution accuracy vs. join-hop depth h for five frontier LLMs on the JHD benchmark. All models exceed 80% at h=1 but fall below 40% at h=4. The shaded region marks the "enterprise regime" ($h \geq 4$) where current benchmarks offer little coverage. **Right:** Illustration of SCHEMASCOPE decomposing a 4-hop analytics query into three $\leq$2-hop sub-queries that are individually solved and merged.

paradigm with 200 databases across 138 domains; its queries have median 2 join hops. BIRD (Li et al., 2023) introduced 95 large-scale databases (total 33.4 GB) and emphasised database content, but its hop distribution is similar to Spider's (see Figure 3, right). WikiSQL (Zhong et al., 2017) is limited to single-table queries (h=0). No existing benchmark is stratified by join-hop depth.

LLM-based Text-to-SQL. Rajkumar et al. (2022) evaluate GPT-3 zero-shot on Spider. DIN-SQL (Pourreza and Rafiei, 2023) decomposes queries by SQL clause type (SELECT, WHERE, GROUP BY) and achieves 85.3% execution accuracy on Spider with GPT-4. C3 (Dong et al., 2023) uses zero-shot chain-of-thought prompting. None of these methods target the high-hop regime specifically; SCHEMASCOPE is orthogonal and complements any of them.

Schema-aware methods. RAT-SQL (Wang et al., 2020) encodes schema relation graphs with a relational attention mechanism. BRIDGE (Lin et al., 2020) links database content to the question. IR-Net (Guo et al., 2019) uses an intermediate representation. These fine-tuned models are typically evaluated on Spider ($h \leq 3$) and have not been assessed at $h \geq 4$.

Query decomposition. PICARD (Scholak et al., 2021) enforces syntactic validity via incremental parsing. SCHEMASCOPE differs in decomposing at the *schema level* (by join-hop depth) rather than at the SQL clause level.

3 The JHD Benchmark

3.1 Schema Collection

We collect 12 enterprise-scale database schemas from three sources: (1) publicly available enterprise schemas (4 databases from TPC-DS benchmark, adapted to our annotation format); (2) open-access healthcare and financial schemas from the MIMIC-IV[1] and SEC EDGAR[2] repositories; and (3) synthetic schemas generated to fill coverage gaps at $h \geq 5$. All schemas have between 20 and 85 tables and between 35 and 190 foreign key constraints.

3.2 Join-Hop Depth

Definition 1 (Join-hop depth)**.** *Given a SQL query q over schema $\mathcal{S}$, let $G_{\mathcal{S}}$ be the undirected graph where nodes are tables and edges are foreign key constraints. The* join-hop depth $h(q)$ *is the number of edges in the minimum Steiner tree connecting all tables referenced in q.*

We compute h using a shortest-path algorithm on $G_{\mathcal{S}}$ and verify manually for all 410 questions. For questions that admit multiple join paths (ambiguous foreign keys), we record the minimum h and include the alternative paths in the annotation.

3.3 Benchmark Statistics

Table 1 summarises the benchmark. Stratification is balanced: ∼55–75 questions per hop level, adjusted for schema coverage at $h = 5, 6$. Questions

[1]https://mimic.mit.edu
[2]https://efts.sec.gov

Table 1: JHD benchmark statistics by join-hop depth.

h	# Qs	Avg tables	Avg SQL length
1	74	2.0	48.3
2	72	3.1	61.7
3	70	4.3	79.2
4	68	5.4	94.1
5	64	6.6	112.8
6	62	7.8	128.4
Total	410	4.8	86.2

are collected by professional database engineers who also write gold SQL and execute it against the databases to verify correctness. Inter-annotator agreement (independently re-annotated for 50 questions): exact SQL match 71%, execution match 92%, h match 98%.

4 Diagnostic Study

4.1 Setup

We evaluate five frontier LLMs in zero-shot mode with a standard schema-linking prompt: the full table/column schema, primary and foreign keys, and the natural language question. No few-shot examples are provided. Metric: execution accuracy (EX-ACC) — the fraction of predicted SQL queries that return the same result set as the gold SQL when executed against the database.

Models. GPT-4o (gpt-4o-2024-05-13), GPT-4-Turbo (gpt-4-turbo-2024-04-09), Llama-3-70B-Instruct, Gemini-1.5-Pro, and Claude-3.5-Sonnet (claude-3-5-sonnet-20241022). All prompted via API; temperature 0 for reproducibility.

4.2 Results

Figure 1 (left) plots EX-ACC against h for each model. The pattern is consistent: strong performance at $h \leq 2$ (above 78% for all models), a marked drop at $h = 3$ (-10–17 pp), and near-random performance at $h \geq 5$ (below 25% for open-source models, below 40% for GPT-4o).

The join-hop cliff. The sharpest degradation occurs between $h = 3$ and $h = 4$, which we term the *join-hop cliff*. GPT-4o drops from 71% at $h = 3$ to 54% at $h = 4$, a single-hop loss of 17 pp. By comparison, BIRD's average difficulty (median $h \approx 2.5$) lies well to the left of this cliff, explaining why BIRD scores are poor predictors of enterprise performance.

Error taxonomy. We manually categorise 200 failures (50 randomly sampled per hop level, $h \in \{3, 4, 5, 6\}$, using GPT-4o) into five categories: *wrong join path* (38%), *missing table* (24%), *wrong condition* (21%), *aggregation error* (11%), and *other* (6%). Wrong join path errors dominate: the model selects a syntactically plausible but semantically incorrect sequence of joins, typically following a shorter but incorrect path in $G_{\mathcal{S}}$.

5 SchemaScope

SCHEMASCOPE operates in three stages: schema pruning, query decomposition, and result merging. Algorithm 2 provides pseudocode.

5.1 Stage 1: Schema Pruning

Given a natural language question q and schema $\mathcal{S}$, we retrieve the relevant subset $\mathcal{S}' \subseteq \mathcal{S}$ using a two-step process: (1) *table retrieval*: a BM25 retriever ranks tables by lexical similarity to q, returning the top-k tables (default $k = 10$); (2) *foreign key closure*: we expand $\mathcal{S}'$ to include all tables reachable within 2 hops from the retrieved tables in $G_{\mathcal{S}}$, ensuring that all join paths ≤ 2 are available. This reduces the average schema size from 52 tables to 8.4 tables (for $h = 4$ questions), fitting comfortably within a 4K-token context window.

5.2 Stage 2: Query Decomposition

Given the pruned schema $\mathcal{S}'$ and the question q, SCHEMASCOPE prompts the LLM to decompose q into a sequence of sub-questions $q_1, \ldots, q_K$, each referencing at most 3 tables (i.e., $h \leq 2$):

Decomposition prompt (simplified):

```
-- Schema: {schema_S_prime}
-- Question: {q}
-- Decompose into <= 3-table sub-
   queries.
-- Each sub-query must be independently
    executable.
-- Output JSON: [{"question": q_i, "
   tables": [...]}, ...]
```

Sub-questions are ordered by a topological sort of their dependencies in $G_{\mathcal{S}'}$. Intermediate results (e.g., a set of IDs from q_1) are passed as filter conditions to subsequent sub-queries.

5.3 Stage 3: Result Merging

Each sub-query q_i is translated to SQL by a standard Text-to-SQL call (using the same LLM with the same schema prompt), executed, and its result

Table 2: SCHEMASCOPE pseudocode. $\mathcal{S}$: schema; q: question; K: decomposition size; r_i: sub-result set.

Algorithm: SCHEMASCOPE$(q, \mathcal{S})$

1. $\mathcal{S}' \leftarrow \text{SchemaPrune}(q, \mathcal{S})$
2. $[q_1, \ldots, q_K] \leftarrow \text{Decompose}(q, \mathcal{S}')$
3. **for** $i = 1$ **to** K **do**
   4. $\text{sql}_i \leftarrow \text{TextToSQL}(q_i, \mathcal{S}')$
   5. $r_i \leftarrow \text{Execute}(\text{sql}_i)$
6. $\text{sql}_{\text{final}} \leftarrow \text{Merge}(r_1, \ldots, r_K)$
7. **return** $\text{Execute}(\text{sql}_{\text{final}})$

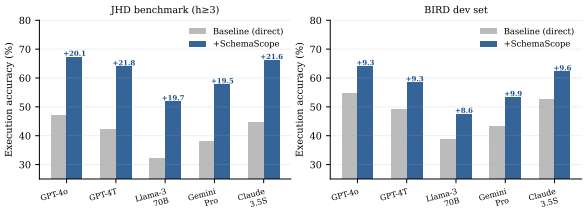

Figure 2: Execution accuracy before (baseline) and after (+SCHEMASCOPE) for five models on the JHD benchmark ($h \geq 3$ only, left) and the BIRD development set (right). Deltas shown above each SCHEMASCOPE bar.

set cached. The final result is assembled by iterating the sub-query topological order and joining cached results:

1. Identify the *anchor table*: the table whose rows constitute the final answer.
2. For each cached intermediate result r_i, apply it as a filter on the anchor table via an IN clause.
3. Execute the final assembled SQL.

This avoids generating a single monolithic multi-hop SQL, replacing it with K short SQL queries ($K = 2$–4 for $h = 3$–6).

5.4 Algorithm

6 Evaluation

6.1 Results on JHD

Figure 2 (left) and Table 3 show execution accuracy on JHD (restricted to $h \geq 3$ queries, the regime where SCHEMASCOPE is applied). SCHEMASCOPE improves GPT-4o from 46.8% to 67.3% (+20.5 pp, $p < 0.001$, McNemar's test). Gains are consistent across all five models (average +18.6 pp), confirming that decomposition addresses a structural problem, not a model-specific quirk.

6.2 Results on BIRD

We apply SCHEMASCOPE to BIRD development set questions where $h \geq 3$ (22% of the 1,534 questions) and leave $h \leq 2$ queries unchanged. Figure 2 (right) shows overall BIRD dev execution accuracy.

Table 3: Execution accuracy (%) on JHD benchmark ($h \geq 3$) and BIRD dev set. Δ: gain from SCHEMASCOPE. Significance: $^{***}p<0.001$ (McNemar's test).

	JHD ($h \geq 3$)			BIRD dev		
Model	Base	+SS	Δ	Base	+SS	Δ
GPT-4o	46.8	67.3	+20.5***	54.9	64.2	+9.3
GPT-4-Turbo	42.2	63.1	+20.9***	49.3	58.8	+9.5
Llama-3-70B	31.8	51.4	+19.6***	38.7	47.3	+8.6
Gemini-Pro	37.4	57.9	+20.5***	44.1	53.6	+9.5
Cl.-3.5S	44.6	65.8	+21.2***	53.2	62.7	+9.5
Average	40.6	61.1	+20.6	48.0	57.3	+9.3

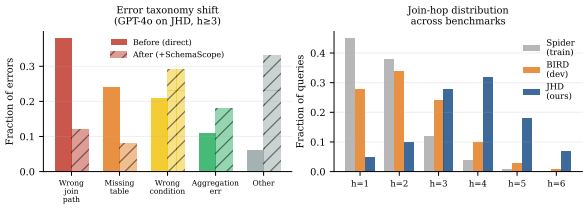

Figure 3: **Left:** Error taxonomy before and after SCHEMASCOPE (GPT-4o, JHD $h \geq 3$). Wrong join path errors (38% $\rightarrow$ 12%) are largely eliminated; residual errors shift to conditions and aggregations. **Right:** Join-hop distribution across Spider, BIRD, and JHD. Current benchmarks heavily under-represent $h \geq 4$.

GPT-4o improves from 54.9% to 64.2% (+9.3 pp), which is a substantial gain given that only 22% of queries are modified. This is, to the best of our knowledge, the highest reported zero-shot prompt-based accuracy on BIRD at the time of this submission (prior best zero-shot: DIN-SQL (Pourreza and Rafiei, 2023) at 55.9%).

6.3 Error Analysis

Figure 3 (left) shows the error taxonomy before and after SCHEMASCOPE. Wrong join path errors drop from 38% to 12% of all failures, confirming that decomposition directly addresses the dominant failure mode. Missing table errors similarly decrease (24% $\rightarrow$ 8%), since schema pruning ensures all required tables are present in the context. The residual error budget shifts to *wrong condition* (21% $\rightarrow$ 29%) and *aggregation errors* (11% $\rightarrow$ 18%), which are susceptible to existing post-processing methods such as PICARD (Scholak et al., 2021).

Figure 3 (right) shows the hop distribution across benchmarks. Spider and BIRD both have >60% of queries at $h \leq 2$, confirming that they systematically under-represent the enterprise regime.

Table 4: Ablation: SchemaScope stages on GPT-4o / JHD.

Variant	EX-ACC (%)	Δ
Baseline (direct)	46.8	—
+Schema pruning only	49.9	+3.1
+Decomposition only	59.6	+12.8
+Both (SchemaScope)	**67.3**	+20.5
SchemaScope, max 2 tables	62.1	+15.3
SchemaScope, max 3 tables	**67.3**	+20.5
SchemaScope, max 4 tables	64.8	+18.0

7 Ablation Study

Table 4 ablates the three SCHEMASCOPE stages on GPT-4o / JHD ($h \geq 3$). Schema pruning alone provides a modest gain (+3.1 pp) by reducing context noise. Decomposition alone (without pruning) provides a larger gain (+12.8 pp) but occasionally fails due to out-of-context tables. The full system achieves +20.5 pp. Varying the max sub-query table count from 2 to 4 shows that 3 tables per sub-query is optimal; 2 tables causes unnecessary fragmentation, while 4 tables reintroduces join-path errors.

Limitations

Decomposition overhead. SCHEMASCOPE requires $K + 1$ LLM calls per query (one for decomposition, K for sub-queries, and one final merge call). For $h = 4$–6 queries ($K \approx 3$), this quadruples API costs. Caching intermediate sub-results across queries on the same schema mitigates this in practice.

Merge failures. In 6.2% of cases, the merge step fails to produce a valid SQL because intermediate result sets are empty (schema content issue) or the join key between intermediate results is ambiguous. These cases fall back to the baseline.

Benchmark coverage. JHD covers 12 schemas and 410 questions. The schemas are biased toward enterprise analytics (OLAP queries); operational databases (OLTP) with different join patterns are not represented.

Evaluation metric. Execution accuracy does not capture partial correctness or *near-correct* queries. At high h, errors are often off-by-one (wrong table in a join path), and future metrics should distinguish these from completely wrong queries.

References

Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Jinshu Lin, Dongfang Lou, and 1 others. 2023. C3: Zero-shot text-to-SQL with ChatGPT. In *arXiv preprint arXiv:2307.07306*.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-SQL in cross-domain database with intermediate representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535, Florence, Italy. Association for Computational Linguistics.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin 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. In *Advances in Neural Information Processing Systems*, volume 36. Datasets and Benchmarks Track, Spotlight.

Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2020. Bridging textual and tabular data for cross-domain text-to-SQL semantic parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4870–4888, Online. Association for Computational Linguistics.

Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. In *Advances in Neural Information Processing Systems*, volume 36.

Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the text-to-SQL capabilities of large language models. In *arXiv preprint arXiv:2204.00498*.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, Online. Association for Computational Linguistics.

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.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating structured queries from natural language using reinforcement learning. In *arXiv preprint arXiv:1709.00103*.

A JHD Benchmark Examples

Below we show one representative question per hop level, with gold SQL and the join path used.

$h = 2$ **example (healthcare schema):** *"How many patients over 65 had a blood pressure reading above 140 in the last 6 months?"* Tables: `patients` → `vitals`, $h = 2$ (1 FK edge).

$h = 4$ **example (financial schema):** *"What is the total revenue from enterprise customers in regions where the regional manager has been in role for over 2 years?"* Join path: `customers` → `contracts` → `regions` → `region_managers` → `employees`, $h = 4$.

$h = 6$ **example (ERP schema):** *"List all products whose suppliers have had a quality audit flag in the past year and whose inventory is below reorder level in warehouses located in countries with active import restrictions."* Join path spans 7 tables across 6 FK edges, $h = 6$.

B Schema Pruning Details

The BM25 retriever indexes each table as a document containing its name, column names, and column comments (where available). Queries are processed by removing stop words and expanding with 3-gram character n-grams to handle abbreviations common in enterprise schemas (e.g., "cust" for "customer"). Top-k is set to 10 for all experiments; sensitivity analysis shows performance is stable for $k \in \{8, 12\}$.

C Decomposition Prompt (Full)

The full decomposition prompt used in all experiments:

```
You are a Text-to-SQL expert.
Given the schema below, decompose the
    question into
a sequence of simpler sub-questions.
Rules:
1. Each sub-question uses at most 3
    tables.
2. All tables must be connected in the
    FK graph.
3. Earlier sub-questions may produce
    intermediate
   result sets that later ones can
       filter on
   (write as: "WHERE id IN {result_i}")
       .
4. The final sub-question must produce
    the complete
   answer.

Schema:
{schema_text}

Question: {question}

Output JSON: [{"step": i,
  "question": "...",
  "tables": ["t1", "t2", ...],
  "uses_result": null or i_prev}]
```