

Contexts are Never Long Enough: Structured Reasoning for Scalable Question Answering over Long Document Sets

Harshit Joshi Priyank Shethia Jadelynn Dao Monica S. Lam
 Computer Science Department, Stanford University
 {hj, lam}@cs.stanford.edu

Abstract

Real-world document question answering is challenging. Analysts must synthesize evidence across multiple documents and different parts of each document. However, any fixed LLM context window can be exceeded as document collections grow. A common workaround is to decompose documents into chunks and assemble answers from chunk-level outputs, but this introduces an *aggregation bottleneck*: as the number of chunks grows, systems must still combine and reason over an increasingly large body of extracted evidence. We present SLIDERS, a framework for question answering over long document collections through *structured reasoning*. SLIDERS extracts salient information into a relational database, enabling scalable reasoning over persistent structured state via SQL rather than concatenated text. To make this locally extracted representation globally coherent, SLIDERS introduces a data reconciliation stage that leverages provenance, extraction rationales, and metadata to detect and repair duplicated, inconsistent, and incomplete records. SLIDERS outperforms all baselines on three existing long-context benchmarks, despite all of them fitting within the context window of strong base LLMs, exceeding GPT-4.1 by 6.6 points on average. It also improves over the next best baseline by ~ 19 and ~ 32 points on two new benchmarks at 3.9M and 36M tokens, respectively.

<https://sliders.genie.stanford.edu/> <https://github.com/stanford-oval/sliders>

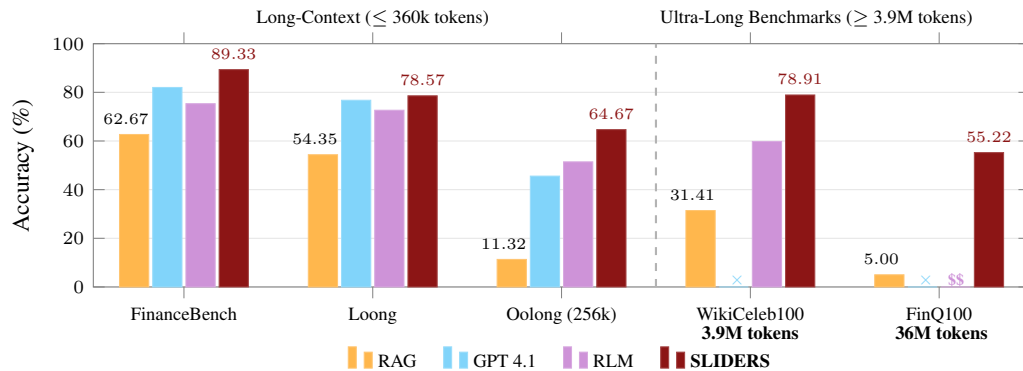


Figure 1: Accuracy across long-context and ultra-long benchmarks. SLIDERS consistently outperforms all baselines and is the only system to scale beyond 3.9M tokens. × denotes baselines that exceeded the model’s context window; \$\$ denotes prohibitively expensive runs.

1 Introduction

Answering questions over long or multi-document corpora accurately is essential for analysis in knowledge-intensive domains such as finance, healthcare, and the social sciences. Analysts routinely need to synthesize evidence scattered across multiple pages in a single report or across thousands of reports in a collection [Zhang et al., 2021, Van De Schoot et al., 2021, Del Fiore et al., 2014]. However, even with million-token context windows, modern LLMs fall short of real-world document analysis, where models must integrate and reason over distributed evidence. Realistic corpora remain far beyond what can be processed in a single pass, forcing systems to rely on selective retrieval or decomposition, where missing relevant evidence can lead to incorrect conclusions. At the same time, LLMs struggle to reliably combine information across distant sections or documents, often producing incomplete, duplicated, or contradictory outputs [Liu et al., 2023a, Hsieh et al., 2024, Goldman et al., 2024]. Finally, free-form outputs hinder auditing and compliance in high-stakes settings, while ultra-long-context inference remains computationally expensive [Gu and Dao, 2024].

A widely adopted workaround is to partition documents into smaller fragments and assemble answers from chunk outputs [Zhang et al., 2024, Zhao et al., 2024, Shankar et al., 2025]. While chunking avoids exceeding context limits and can improve attention to local details, it introduces a new bottleneck: as the number of chunks increases, systems must aggregate and reason over evidence extracted from all chunk outputs, ultimately recreating the very long-context problem it was designed to avoid, a fundamental limitation we refer to as the ‘‘Aggregation Bottleneck’’.

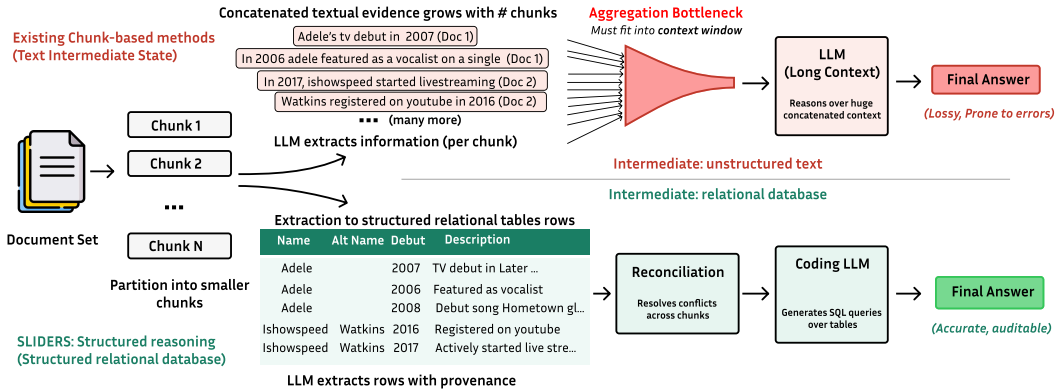


Figure 2: Chunking based methods regenerate the long-context problem. SLIDERS mitigates it by using structured reasoning.

To overcome the aggregation bottleneck in question answering over long document sets, we propose structured reasoning. Instead of relying on LLMs to reason over concatenated chunk outputs, we extract salient information into a relational database, transforming unstructured text into a persistent, structured state. This representation is inherently scalable, allowing us to store and organize arbitrarily large volumes of evidence in a compact, queryable form, rather than being constrained by context limits. Once information is explicitly stored, reasoning reduces to querying: we execute SQL over the database to aggregate, compare, and compute over the extracted evidence, enabling operations that LLMs struggle to perform reliably in free-form generation [Hsieh et al., 2024, Bai et al., 2023, Li et al., 2025, Mirzadeh et al., 2024]. In this way, long-context reasoning becomes database reasoning, where answers are derived by systematically inspecting and combining stored information without requiring all evidence to be reloaded into the model’s context (Figure 2). Leveraging the code-generation capabilities of LLMs, we bridge unstructured text and structured execution, using models to translate questions into queries while offloading symbolic computation to the database where it can be performed deterministically [Jimenez et al., 2023, Yang et al., 2025, Hong et al., 2025].

To transform independently extracted evidence into a coherent and scalable database for reasoning, we introduce a data reconciliation stage. Because documents distribute, repeat, and refine information across sections at varying levels of granularity [Sollaci and Pereira, 2004, Mann and Thompson, 1988], independently extracted records often provide overlapping, partial, or conflicting views of the same underlying fact. Reconciliation groups related records, integrates partial evidence,

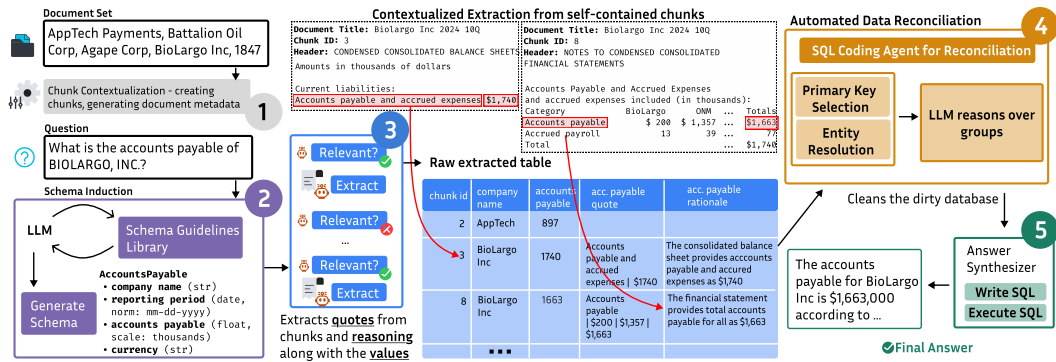


Figure 3: SLIDERS overview showing the process of (1) Contextualization of chunks, (2) Inducing the schema, (3) Structured Extraction, (4) Data Reconciliation, and (5) Question Answering over the final database. An example from the Loong benchmark.

and resolves inconsistencies into a unified representation. Our key insight is that this process can exploit how each record was produced: every entry is stored with its provenance, extraction rationale, and metadata, enabling a reconciliation agent to inspect supporting evidence and generate updates grounded in the original extraction context. This design also improves efficiency. Rather than performing pairwise comparisons over all extracted records, which scales quadratically with the number of entries, reconciliation restricts reasoning to key-based partitions, substantially reducing the effective problem size. By combining LLM reasoning with database operations, reconciliation converts local extractions into a globally consistent, queryable state for downstream question answering. To our knowledge, this is the first approach to treat provenance and extraction rationale as first-class signals for reconciliation in LLM-based information extraction.

We present **SLIDERS** (**S**calable **L**ong-document **I**ntegration through **D**ecomposed **E**xtraction and **R**econciliation **S**ystem), a framework for structured extraction and reconciliation over long document collections. On three long-context benchmarks that fit within the context window of strong base LLMs, SLIDERS outperforms all baselines, exceeding GPT-4.1 by 6.6 points on average. To evaluate reasoning beyond current context limits, we also introduce two new benchmarks of 100 documents each, totaling 3.9M tokens for Wikipedia articles and 36M tokens for financial filings. On these benchmarks, SLIDERS achieves 78.91% accuracy on Wikipedia and 55.22% on financial documents, compared to 59.80% and 5% for the next best baseline, respectively.

2 From Documents to Structured Reasoning

Given a question q and a collection of long documents \mathcal{D} , SLIDERS answers q by transforming unstructured text into a globally coherent database. In the following, we overview the research questions (RQ) and our approach, which consists of five tasks, as shown in Figure 3.

RQ1: How to decompose documents for extraction to preserve the required global context for correct interpretation?

1. The *Contextualized Chunking Task*. Decompose documents \mathcal{D} into context-aware chunks \mathcal{C}_d using extracted document-level and structural metadata \mathcal{M} , producing locally self-contained units suitable for extraction.

RQ2: How to represent and extract information from chunks of a document with a relational database that preserves all information necessary for answering the question on the original documents?

2. The *Schema Induction Task*. Derive a schema \mathcal{S} from question q and document metadata \mathcal{M} , specifying the entities, attributes, and relationships to extract.

3. The *Structure Extraction Task*. Extract relevant information from document \mathcal{D} with metadata \mathcal{M} according to schema \mathcal{S} for question q to create a set of tables \mathcal{T} while preserving provenance and extraction rationales.

RQ3: How to construct a coherent global database from partial, redundant, and potentially conflicting extractions, and answer questions using the database?

4. The *Data Reconciliation Task*. Cleans tables \mathcal{T} generated by the Structure Extraction Task, with respect to question q to enhance its correctness with an SQL-coding agent.

5. The *Question Answering Task*. Synthesize the answer to question q by generating and executing SQL queries over the reconciled database \mathcal{T} [Liu et al., 2024]. The QA agent iteratively generates queries over the reconciled database: (1) it is given the schema and database, (2) the query is executed and the result returned, (3) the agent refines the query if needed, repeating until a satisfactory answer is produced.

3 Contextualized Extraction

3.1 The Contextualized Chunking Task

To ensure that extracted records are interpretable on their own, we retain the global and layout context as we extract information from documents to avoid issues such as headers detached from tables or orphaned paragraphs [Lewis et al., 2020, Jain et al., 2025]. SLIDERS augments each document $d \in \mathcal{D}$ with metadata $m_d = (m_d^G, m_d^L) \in \mathcal{M}$ prior to chunking, where the global part m_d^G consists of a generated title and brief document-level description to provide high-level context shared across all chunks; the local part m_d^L captures structural signals such as section headers, tables, and figure captions, enabling chunk boundaries be formed that aligns with the document’s natural layout.

Using this enriched representation, we partition each document into chunks $c \in \mathcal{C}_d$ while preserving semantic and structural coherence, avoiding splits within paragraphs, tables, code blocks, or captions. Each chunk is thus locally self-contained and associated with metadata including document identifiers, page indices, and structural tags.

3.2 Schema Induction

Unlike free-form extraction, where an LLM generates unstructured natural language responses for each document chunk, relational databases fundamentally differ in their requirement for a rigid schema. In addition, free-form outputs are inherently polymorphic: one chunk might yield a verbose explanation, while another provides a concise numeric value or uses a different unit of measurement (e.g., 75°F vs. 24°C). We need to enforce a standard type and format for all values in each column to support SQL operations, like aggregation or filtering. To bridge this gap, we must constrain the extraction process with a task-specific schema.

Definition 1. *The Schema Induction Task takes a question q and extracted document metadata \mathcal{M} and produces a relation schema with multiple tables $\mathcal{S} = \{S_1, S_2, \dots, S_k\}$; a table schema $S = \langle sn, f_1, \dots, f_n \rangle$ has a schema name sn , and a set of fields f_i . A field is defined as a tuple*

$$f = \langle fn, d, \tau, u, \sigma, \rho \rangle$$

*where fn is the field name; d is a semantic text description; τ is the data type (e.g., *int*, *str*); u is the unit of measurement (e.g., *USD*, *kg*); σ is the scale (e.g., *millions*, *thousands*); and ρ represents normalization rules (e.g., *currency conversion*, *date formatting*).*

We create a schema library that provides guidance for constructing task-appropriate relational schemas. Using the library, an LLM first classifies the query type and document type, retrieves the guidelines, and follows it to produce the desired schema. We provide the following question types: “Ordering”, “Multiple Choice”, “Others” and the following document types: “Narration”, “Policy”, “Dataset”, “Others”. More details on the guidelines are present in Appendix A.3.

3.3 Contextualized Extraction with Relevance Gating

In our experiment to use in-context learning to extract knowledge for our schema, we observed empirically that hallucination is common when a chunk contains no mention of information for a field [Liu et al., 2023b, BBC, 2025]. The fact that our schema has strict type requirements appears to exacerbate this problem, which we hypothesize to be caused by training biases that prioritize structural compliance over factual abstinence.

To mitigate this, we introduce a *relevance gate* prior to extraction. For each chunk, the model first determines if the text contains evidence relevant to the schema entities; we invoke the extraction on this chunk only when this gate passes. This two-stage process ensures that the extraction model is active only when presented with high-signal context, preventing the injection false positives into the database. We evaluate the relevance gate on 20 sampled incorrect predictions across benchmarks. Across 516 chunks, 282 were rejected by the relevance gate, with only 1 false negative, yielding a false-negative rate of 0.4%. This indicates that the relevance gate is not a major source of error; most failures arise downstream (e.g., schema mismatch or reconciliation).

Definition 2. The Structured Extraction Task, *SE*, accepts as input question q , document \mathcal{D} , metadata \mathcal{M} , and schema $\mathcal{S} = \{S_1, S_2, \dots, S_k\}$ to produce a set of rows, capturing relationships mentioned in the chunk, in tables $\mathcal{T} = \{T_1, T_2, \dots, T_k\}$, where T_i conforms to schema S_i .

Each entry of field f , $e_f = \langle v, p, r \rangle$, where v is the normalized field value respecting the unit and scale specified for f , p is the provenance, the minimal textual span supporting the value extracted, and r describes the decision used to map the quote to the extracted value.

Structure Extraction Task. The Structured Extraction task for a document is formulated as

$$SE(q, \mathcal{D}, \mathcal{M}, \mathcal{S}) = \cup_{c \in \mathcal{D}, d \in \mathcal{D}} SE(q^e, c, m_c, \mathcal{S}),$$

where q^e is an adaptation of q for the extraction process, and \cup denotes table union without removing duplicates.

$$SE(q^e, c, m, \mathcal{S}) = \begin{cases} SE_{LLM}(q^e, c, m, \mathcal{S}), & \text{if } R(q^e, c, m, \mathcal{S}) \\ \emptyset, & \text{otherwise.} \end{cases}$$

where $R(q^e, c, m_c, \mathcal{S})$ is an explicit relevance gate determining if c is relevant to q^e , to minimize hallucination. The extraction of information from a chunk, SE_{LLM} , is implemented with in-context learning, generating structured output as JSON objects.

Why extraction scales? (i) **Unbounded corpus size:** The analysis is applied to a chunk at a time, whose size can be chosen to fit in any single model’s context window. (ii) **Parallelism:** The analysis of each chunk (gating and extraction) is independent of each other and can thus be parallelized.

Table 1: Executor agents for data reconciliation. Each agent applies a distinct reconciliation operation over related records, using provenance and rationale to guide decisions and produce a coherent representation.

Operation	Reconciliation Need	Decision Making	Action
Deduplication	Semantically identical or near-identical rows expressed with different phrasing	Prefer the most precise or explicitly stated value; assess specificity via provenance	Select a canonical representation and collapse redundant rows
Conflict Resolution	Competing values for the same attribute across related rows	Examine provenance and rationale to determine which value is best supported in context	Retain the best-supported value and remove incompatible alternatives
Consolidation	Partial rows capturing complementary attributes of the same entity or fact	Determine whether attributes can be meaningfully combined across rows without contradiction	Merge complementary records into a more complete representation and propagate shared values where appropriate

4 Data Reconciliation

Although information extraction from individual document chunks is often accurate in local context, aggregating these extractions across an entire document or document set can still yield an incoherent global representation. Documents distribute, repeat, and refine related information across sections at different levels of granularity. For example, a Wikipedia article may mention an artist’s date of birth and primary profession in the introduction, while later sections discuss additional professions. Each chunk may therefore yield a correct extraction in isolation, yet their aggregation can still produce missing attributes, redundant entries, or conflicting values. Reconciliation addresses this challenge by aligning overlapping, partial, or conflicting records into a coherent relational state.

Definition 3. *The Data Reconciliation Task, DR, accepts as input question q and extracted tables $\mathcal{T} = SE(q, \mathcal{D}, \mathcal{M}, S)$ and produces reconciled tables $\mathcal{T}' = \{T'_1, T'_2, \dots, T'_k\}$.*

A naive approach would reason jointly over all extracted rows, requiring pairwise comparisons across the full table and becoming quadratic in the number of entries. Our key observation is that relational structure provides a natural decomposition. Our key observation is that reconciliation can exploit the structure of text itself: although information is distributed across sections, statements about the same entity or claim are typically anchored by a common identifying attribute or set of attributes. In the extracted relational representation, these attributes serve as a primary key. Grouping rows by this key allows us to reconcile overlapping evidence within small, semantically coherent partitions.

Rather than relying on a fixed reconciliation program, we collect during extraction the evidence needed to reconcile each group: provenance, extraction rationale, and document-level metadata for every table entry. We then use an agent to reason over each key-based partition and dynamically generate SQL programs that align duplicate references, integrate complementary attributes, and resolve competing values. In this way, reconciliation converts independently extracted local views into a globally coherent and queryable database for downstream question answering.

Phase 1: Partitioning for Reconciliation

Primary Key Selection. To determine the primary key, which may consist of one or more columns, we present an LLM agent with (1) an adaptation of the input question q for reconciliation, (2) the table schema, and (3) a sample of extracted rows with their rationales. The LLM identifies the primary key, and to improve robustness we query it three times and use majority voting to select the final key.

Entity Resolution of Primary Key Values. A central step in reconciliation is aligning semantically equivalent key values [Bhattacharya and Getoor, 2007, Fellegi and Sunter, 1969]. For example, the same person may appear as “J. Smith,” “John Smith,” or “Smith, John” across documents. These variants must be mapped to a common representation before the remaining evidence can be integrated.

We first resolve entities within each document, one column at a time, using an LLM prompt that focuses on semantic equivalence. This acts as a blocking step, reducing the number of distinct references before cross-document resolution [Newcombe, 1967, Papadakis et al., 2016]. We then apply an iterative LLM-based pipeline across documents: the model generates SQL queries to sample rows, inspects evidence from available attributes and extraction rationales, and produces SQL statements to normalize and align key values. If the primary key columns contain free text or exhibit extremely high cardinality, the agent may skip this step.

Grouping by Primary Key. Finally, records are partitioned by primary key using a SQL group-by statement. Each resulting group forms an independent reconciliation unit.

Phase 2: Reconciliation Agent

For each partition of records, the reconciliation agent uses provenance, extraction rationale, and document-level metadata to determine how the evidence should be integrated into a coherent representation.

Reconciliation Operation Selection. From empirical analysis, we find that most reconciliation decisions fall into three classes: (1) *deduplication*, where multiple rows express the same information; (2) *conflict resolution*, where rows provide competing values for the same attribute; and (3) *consolidation*,

Algorithm 1 Data Reconciliation (DR)

Require: Question q , Tables $\mathcal{T} = \{T_1, \dots, T_k\}$

Ensure: Reconciled $\mathcal{T}' = \{T'_1, \dots, T'_k\}$

```

1:  $q^r \leftarrow$  adapt  $q$  for reconciliation
   % Phase 1: Partitioning for Reconciliation
2: for each table  $T_i \in \mathcal{T}$  do
3:    $pk_i \leftarrow$  SELECTPRIMARYKEY( $T_i, q^r$ )
   % doc-level, then table-level
4:    $T_i \leftarrow$  RESOLVEPRIMARYKEYENTITIES( $T_i, pk_i$ )
5:    $\mathcal{G}_i \leftarrow$  GROUPBYPRIMARYKEY( $T_i, pk_i$ )
6: end for
   % Phase 2: Data Reconciliation Agent
7: for each table  $T_i \in \mathcal{T}$  do
8:   for each group  $g \in \mathcal{G}_i$  in parallel do
9:     loop
10:     $op \leftarrow$  SELECTRECONOP( $g, q^r$ )
11:    if  $op = \emptyset$  then break
12:    end if
13:     $sql \leftarrow$  RECONCILEGROUP( $g, op$ )
14:     $g \leftarrow$  APPLY( $sql, g$ )
15:    end loop
16:   end for
   % Non-primary-key entity resolution
17:    $T'_i \leftarrow$  RESOLVENONPRIMARYKEYENTITIES( $\bigcup_g g, \overline{pk_i}$ )
18: end for
19: return  $\mathcal{T}'$ 

```

Table 2: Benchmark statistics.

Benchmark	# Docs	# Questions	Real/Synth	Task Type
FinanceBench	1 per Q	150	Real	Extraction, Arithmetic
Loong	~11 per Q	50	Real	Retrieval, Aggregation
Oolong	1 per Q	192	Synthetic	Classification, Aggregation
WikiCeleb100	100	22	Real	Aggregation, Comparison
FinQ100	100	25	Real	Aggregation, Arithmetic

where partial rows contribute complementary attributes. The agent writes SQL programs to identify which of these operations is needed by examining row contents, value distributions, and supporting evidence.

Evidence Integration. Conditioned on the selected operation, the agent generates SQL programs that integrate the rows in the partition. It may issue intermediate queries, for example, to compute distinct counts, isolate specific columns, or retrieve related rows, before producing a final update. Because all reconciliation actions are expressed as SQL, the process remains auditable. As a final step, we also apply entity resolution to the remaining non-key columns.

5 “Long-Context” Benchmarks ($\leq 360\text{K}$ Tokens)

We evaluate SLIDERS on three “long-context” benchmarks, where the inputs can fit in the context window of most of the frontier models, having up to 360k tokens. For such input lengths, *would SLIDERS perform worse given the overhead of chunking and reconciliation?*

5.1 Benchmarks and Evaluation Metrics

FinanceBench [Islam et al., 2023] A single-document financial question-answering benchmark targeting realistic analyst style queries. It comprises of 150 questions about publicly traded companies, with evidence drawn from public filings. *Evaluation metrics:* We use an LLM-as-a-judge setup. Given the question, the gold answer with justification, and the model’s predicted answer with its justification, the judge model determines whether the prediction is correct.

Loong [Wang et al., 2024] A multi-document question-answering benchmark where every document provided is required to answer the query, and omitting any yields an incorrect answer. Covers three domains, two languages: finance (English and Chinese), law (Chinese), and academic research papers (English), with an average of ~11 documents per instance. *Evaluation metrics.* We use the official Loong scoring procedure, which employs the Loong-provided LLM judge to assess correctness.

Oolong [Bertsch et al., 2025] A long-context reasoning benchmark focused explicitly on aggregation. Each input consists a datapoint from a large dataset and models must first classify and then aggregate those local predictions to produce a global answer. In our experiments we use the Oolong-Synth subset and evaluate on 256K. *Evaluation Metrics:* For non-numeric questions, we use an LLM-as-a-judge evaluation to determine correctness. For numeric aggregation questions, we adopt Oolong’s metric, which assigns higher scores to predictions with smaller deviation.

5.2 Baselines

We compare SLIDERS against several strong baselines.

RAG baseline. We implement a retrieval-augmented generation (RAG) system using Qwen3-4B Embedding as the encoder and GPT 4.1 as the base model for answering. We use dense retrieval with 4096-token chunks, top- k retrieval with $k=5$ for Loong and Oolong, and $k=100$ for FinQ100.

LongRAG. [Jiang et al., 2024] We evaluate LongRAG, which groups retrieved chunks into longer contexts (4K-30K tokens) to reduce the total number of retrievals and provide more coherent evidence to the reader LLM (GPT 4.1).

GraphRAG. [Edge et al., 2024] We evaluate GraphRAG, which builds a knowledge graph over the corpus and uses graph-based retrieval with local search (the recommended mode) and GPT 4.1.

Table 3: Performance comparison across long document QA benchmarks. SLIDERS outperforms RAG, base LM, and RLM baselines. WikiCeleb100 (WC) and FinQ100 (FQ) do not fit in the context window of GPT 4.1. RLM scores 7.4% over 10 documents for FinQ100. Given the low score, we did not run it on FinQ100, which we estimate will cost \$2000. Best results are in **bold**. Paired t -tests against the strongest baseline yield $p < 0.005$ across all benchmarks. FB is FinanceBench.

Models	LLMs	“Long-Context” Benchmark (<360k)				3.9M T	36M T
		FB	Loong	Oolong	Avg.	WC	FQ
RAG	Qwen3-4B & GPT 4.1	62.67	54.35	11.32	42.77	31.41	5.00
LongRAG	Qwen3-4B & GPT 4.1	72.00	59.10	22.00	51.03	43.20	28.87
GraphRAG	Qwen3-4B & GPT 4.1	75.33	61.28	22.00	52.87	48.59	\$\$
Basemodel	GPT 4.1	82.00	76.74	45.56	68.69	N.A.	N.A.
Basemodel	Qwen3.5 122B-A10B	<u>84.67</u>	74.78	24.89	61.44	N.A.	N.A.
DocETL	GPT 4.1	63.33	75.03	49.00	62.44	54.26	\$\$
Chain of Agent	GPT 5 & GPT 5-mini	71.30	54.46	17.11	47.62	\$\$	\$\$
RLM	GPT 5 & GPT 5-mini	75.33	72.64	51.42	66.46	59.80	\$\$
SLIDERS	GPT 4.1 & GPT 4.1-mini	89.33	78.57	<u>64.67</u>	75.56	78.91	<u>55.22</u>
SLIDERS	Qwen3.5 122B-A10B	82.10	<u>75.70</u>	68.00	<u>75.26</u>	<u>76.92</u>	60.18

Frontier LLM baseline. We evaluate GPT-4.1 and Qwen3.5 122B-A10B [Qwen Team, 2026] on all benchmarks under the same prompting and evaluation protocol used for SLIDERS. It has a context window of 1M tokens.

DocETL [Shankar et al., 2025] We evaluate DocETL with its V1 greedy optimizer, GPT 4.1, 16K-character chunking (matching SLIDERS), per-question schema generation, per-chunk map extraction, and single-pass reduce synthesis.

Chain of Agents (CoA) [Zhang et al., 2024] We implement Chain of Agents, where an LLM sequentially processes each document chunk, producing a running summary that is passed to the next chunk until a final answer is generated. We use GPT 4.1 as the synthesizer and GPT 4.1-mini as the chunk summarizer.

RLM [Zhang et al., 2025] We evaluate Recursive Language Model agent that writes python programs to decompose, and call LLM over its input. We use 30 iterations, GPT 5 for main LLM and GPT 5-mini for sub LLM.

5.3 Results

We present the results on the context-bounded benchmarks that fit in the context window of base LM in Table 3.

Observation 1: SLIDERS outperforms all baselines on all the long-document-set question-answering task, even though the inputs all fit in the context of frontier models. SLIDERS outperforms the best baseline (GPT 4.1) by an average of 6.6% even though the inputs fit in the LM’s context. The largest difference of 14% over the base LM observed for Oolong demonstrates the advantage of structured reasoning in SLIDERS for questions requiring aggregation over large contexts. Paired t -tests against the strongest baseline (GPT-4.1) yield $p < 0.005$ across all benchmarks, confirming statistical significance.

For Loong, SLIDERS delivers an accuracy of 59.9%, 74.8%, 89.2%, 91.3%, for the Chinese Legal, English Finance, Chinese Finance, and English Papers domains, respectively. Accuracy degradation over the baseline is observed only for the Chinese Legal domain, a classification task on small (16K) legal documents; the chunking and reconciliation steps in SLIDERS introduce overheads, while offering no benefits for this benchmark.

Observation 2: SLIDERS’s provenance tracking enhances auditability and interpretability, thus facilitating **error analysis**. Sampling the discrepancies between SLIDERS’s output and the gold answers, the provenance information in SLIDERS helps pinpoint the cause of the errors. Detailed discussions can be found in Appendix C. The *common errors in SLIDERS* include (1) not answering questions that need subjective judgement correctly, and (2) misinterpreting terminology, such as

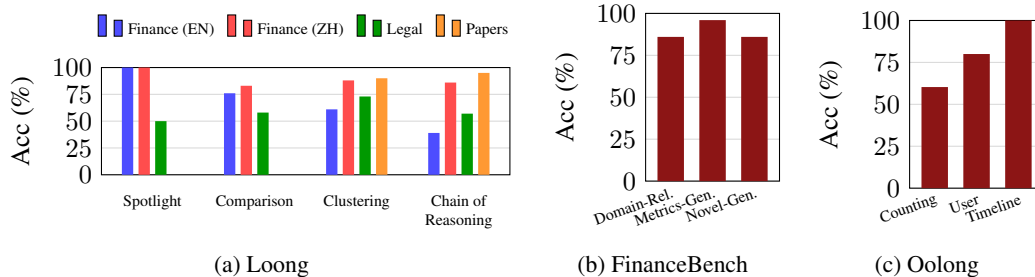


Figure 4: Model accuracy across difficulty levels and question types.

fiscal vs. calendar years. We also found *errors in the human-annotated gold answers*: (1) the gold sometimes has the wrong value or units, (2) for results that span multiple documents, the gold may not be complete.

Observation 3: SLIDERS also works effectively with open-source LLMs, showing that its gains come from the framework rather than reliance on a proprietary frontier model. When instantiated with Qwen3.5-122B-A10B, SLIDERS achieves an average score of 75.26 on the context-bounded benchmarks, improving substantially over the same model used directly as a base LM (61.44). On Oolong, SLIDERS improves Qwen from 24.89 to 68.00, a gain of more than 43 points, showing that structured reasoning is especially beneficial for aggregation-heavy questions. SLIDERS with Qwen also remains competitive with GPT-4.1-based systems, exceeding the GPT-4.1 base LM average of 68.69 and achieving stronger results on benchmarks such as Oolong and FinQ100. These results show that the benefits of SLIDERS are largely orthogonal to the choice of underlying model: even with an open-source LLM, converting text into persistent structured state and reasoning over it with SQL yields large gains.

5.4 Ablation

In Table 4, we ablate key components of SLIDERS: chunking, reconciliation, the reconciliation agent, and the separation of information representation from reasoning. We construct a validation set consisting a random sample of 220 tasks across the three benchmarks. For this split, the performance of SLIDERS is consistent with the full dataset, whereas we see fluctuations in the baseline especially for RAG. We find that chunking is important for all the benchmarks, especially Oolong, where extraction from the full text yields a 40.00% accuracy. Reconciliation is particularly important for Finance Bench. Answering the questions directly using an LLM from the extracted table sees a major degradation for both Finance Bench and Oolong.

Table 4: Ablation study on the val set, removing chunking, reconciliation, and SQL-based QA individually. **Bold/underline** = best/second best.

Model	F. Bench	Loong	Oolong	Avg
SLIDERS	80.00	<u>84.37</u>	64.67	74.79
w/o Chunking	70.00	79.72	40.00	60.34
w/o Reconciliation	76.70	82.84	<u>62.42</u>	<u>72.71</u>
w/o Recon+SQL	70.00	84.45	58.62	70.74

5.5 Question Type Breakdown

Figure 4 breaks down accuracy by question type across all three benchmarks. On **Loong** (Fig. 4a), performance varies more by domain than by difficulty: Finance (ZH) and Papers remain above 83% at all difficulty levels, while Legal stays below 73% throughout. Chain of Reasoning is the hardest category overall, but its impact is domain-dependent, Finance (EN) drops to 39% while Papers rises to 95%. On **FinanceBench** (Fig. 4b), accuracy is consistently high (86.0-96.0%) across domain-relevant, novel, and metric-generated questions. On **Oolong** (Fig. 4c), SLIDERS handles User and Timeline questions well (80% and 100% respectively), with Counting lower at 60.3%. Counting questions require producing exact numerical answers, which depends heavily on the underlying model’s reasoning and classification capabilities, a bottleneck that lies outside SLIDERS’ pipeline.

6 Ultra-Long Document-Set Benchmarks

To stress-test large-scale multi-document aggregation, we introduce two document-set benchmarks that exceed the context windows of current frontier models. In practice, we often want to ask many questions over the same document set; accordingly, each benchmark includes a suite of topic-focused questions. Our approach converts documents into reusable structured data, enabling *cost-effective answers to many downstream questions* and *amortizing the one-time extraction and reconciliation costs*.

WikiCeleb100 comprises Wikipedia articles for the 100 most-viewed celebrity pages from November 2025 to January 2026, totaling **3.9M tokens**. The test set consists of 22 questions on the topic of *debuts*; We provide SLIDERS with the representative question: “Who debuted at the youngest age across the following industries: Music, Film, Content Creation, and Other?” Answering this requires extracting each celebrity’s date of birth, debut date, and industry, information scattered across all 100 articles. Once extracted, this schema supports additional questions such as which decade has the most representation of artists.

FinQ100 comprises the most recent 10-Q filings from 100 randomly selected SEC-listed companies, totaling **36M tokens**. The test set consists of 25 question on the topic of *long-term borrowing*. We provide SLIDERS with the representative question: “Which company has the lowest long-term borrowing?” This question is challenging because many companies do not explicitly state that their long-term borrowing is zero, requiring inference from context.

All the questions, included in Appendix B, require aggregating information across documents. We use LLM-as-a-judge to compare the gold answer against the predicted answer; using partial scoring similar to the Loong benchmark.

Observation 1. SLIDERS achieves the state-of-the-art accuracy on the ultra-long document set benchmark. The benchmarks are too large to run on GPT 4.1, and the performance with RAG is 31.4% and 5.0% for FinQ100. SLIDERS achieves 78.9% on WikiCeleb100, an accuracy similar to the “long-context” benchmark despite the 3.9M token input size. It improves over the best baseline RLM by 19.1%, while being $13\times$ more cost efficient.

SLIDERS achieves 55.2% accuracy for FinQ100. FinQ100 is challenging due to its scale (36M tokens). Due to cost constraints (\$2000 for a full run on FinQ100), we evaluate RLM on only 10 documents from FinQ100, where it scores 7.4% compared to 65.1% for SLIDERS. (SLIDERS costs \$34 to run the complete FinQ100 benchmark.)

Unlike previous benchmarks, such as Oolong, answers cannot be derived by local reasoning and ignoring global context. The answers are heavily fragmented: SLIDERS extracted 685 rows, whereas the ground truth has just 105 rows across 100 companies. Effective data reconciliation is critical to getting accurate answers for this benchmark. Our ablation study shows the *importance and the effectiveness of our reconciliation agent*, without which the accuracy drops from 55.22 to 35.81 on FinQ100 and 78.91 to 60.50 on WikiCeleb100. It gathers disparate information across long documents and reason over it collectively.

Observation 2. SLIDERS can accelerate the manual question answering on large document sets. Question answering on FinQ100 is representative of many tasks in the financial domain. Although SLIDERS achieves state-of-the-art accuracy, it is not reliable enough for end-to-end automation. Nevertheless, SLIDERS can significantly streamline the manual workflow: a reviewer can use the tracked provenance to validate and correct each document’s extractions much faster than working from scratch. Once the database has been verified, the reviewer can pose a wide range of questions about the documents, and SLIDERS will generate the corresponding SQL queries. Overall, the system’s interpretability facilitates effective human-in-the-loop feedback, enabling a more trustworthy question-answering platform.

7 Analysis

7.1 Input Context Length.

To understand the scalability of different techniques, we plot the accuracy of SLIDERS against the input token count in Figure 5. For benchmarks with less than 360K tokens, even though the input fits

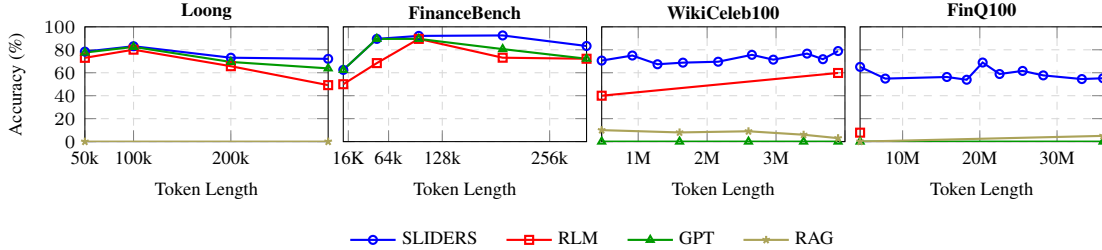


Figure 5: Accuracy vs. token length across benchmarks.

within the LLM model window, the accuracy tends to degrade with increasing token length for all the baselines. SLIDERS’s accuracy fluctuations align with expected input variability. The scalability of SLIDERS is more apparent with the results of the WikiCeleb100 and FinQ100 benchmarks, as its accuracy stays roughly the same up to 35M tokens. RLM can handle up to 4M of WikiCeleb100, but with a lower accuracy; it is too costly to run on FinQ100. The other baselines cannot handle documents over 1M tokens.

7.2 Schema Induction Robustness

A natural concern is whether SLIDERS is sensitive to the choice of schema-induction model. We generate schemas with three models of varying capability (GPT-4.1-mini, GPT-4.1, GPT-5) on Loong and FinanceBench. The resulting schemas differ substantially: GPT-4.1 averages 1.0 tables and 3.3 fields per question vs. GPT-5 with 1.54 tables and 13.3 fields, a 4× complexity gap. Yet downstream accuracy remains stable (Table 5): Loong average accuracy spans just 2.1 points across all three models, and FinanceBench stays within 3.3 points, demonstrating that schema induction is not a fragile bottleneck. One caveat is that more complex schemas can increase reconciliation difficulty, post-reconciliation ranges widen modestly on Loong, suggesting diminishing returns from overly detailed schemas.

Table 5: Accuracy across schema-induction models. **Bold** indicates best per dataset; Δ is the range (max – min).

Dataset	4.1	4.1-mini	5	Δ
Loong Papers	91.30	89.96	88.00	3.30
Loong Legal	64.12	68.34	61.26	7.08
Loong Finance EN	74.50	68.10	73.10	6.40
Loong Finance ZH	93.96	90.46	93.20	3.50
Loong Avg	80.97	79.22	78.89	2.08
FinanceBench	76.71	80.00	80.00	3.33

7.3 Evaluation Reliability

LLM-as-a-judge variance. We quantify evaluation variance by running GPT-4.1 (temperature 0.7) three times per instance: FinanceBench std = 0.47, Loong std = 0.31, Oolong std = 1.02, WikiCeleb std = 0.21, FinQ100 std = 0.38. Low variance across benchmarks indicates stable evaluation.

Human validation of LLM-as-a-judge. To validate automatic evaluation, we manually annotated 50 questions each from FinanceBench (for both SLIDERS and GPT-4.1) and Oolong (for SLIDERS), yielding a combined Cohen’s $\kappa = 0.758$ (substantial agreement). Table 6 breaks down agreement by condition. Notably, SLIDERS errors on FinanceBench are exclusively false negatives (4 FN, 0 FP), indicating that our reported scores are conservative estimates of true accuracy.

Table 6: Manual evaluation on 50 questions each. FN/FP denote false negatives/positives relative to human labels.

Dataset	System	Cohen’s κ	FN	FP
FinanceBench	SLIDERS	0.769	4	0
FinanceBench	GPT-4.1	0.646	3	2
Oolong	SLIDERS	0.855	1	1
Combined		0.758	—	—

7.4 Cost Analysis.

We analyze the cost of SLIDERS across benchmarks and find an average cost \$0.76 per question. Approximately 40% of this cost is incurred by the Entity Resolution task, which must scan the entire table to identify potential matches. A detailed per-task cost breakdown is provided in the Appendix A.6. Compared to RLM, an agent scaffolding baseline, SLIDERS achieves equal or lower cost. Notably, for tasks where the schema can be predefined, the cost of SLIDERS is amortized across queries. To illustrate, consider the WikiCeleb100 benchmark: running GPT-4.1 with an infinite context window would cost approximately \$171.60, whereas SLIDERS costs \$13.10. This advantage becomes more pronounced on FinQ100, where the respective costs are \$1800 and \$34.63.

7.5 Latency Analysis.

SLIDERS operates in two modes depending on whether the document collection is reused. In the *end-to-end* setting, where a structured representation is built from scratch for a single query, average latency is 2.6 minutes on Loong and 3.0 minutes on FinanceBench. This is higher than a single LLM call, but SLIDERS targets complex multi-document questions where accuracy matters more than sub-second response times. In the *amortized* setting, the offline phase (extraction + reconciliation) runs once per collection and the online phase (schema-guided SQL generation) takes ~ 25 seconds per question. On WikiCeleb100 (100 documents), the full offline pipeline completes in ~ 16 minutes: schema induction (20s), extraction (6 min, parallelized), and reconciliation (9.7 min). For comparison, GraphRAG requires 2.3 hours and \$182 to index the same collection while achieving 48.59% accuracy. SLIDERS reaches 78.91% at a fraction of the time and cost.

7.6 Case Study: Multi-Document Summarization.

To demonstrate generality beyond QA, we ran SLIDERS with the prompt “Summarise the given research papers” on three ML papers from the Loong benchmark. The system induced a 5-table schema: PaperSummary (title, authors, abstract), PaperContributions, PaperMethods, PaperFindings, and PaperConclusions, each with paper_title as linking key and free-text fields. It extracted 117 records across 3 papers, then aggregated them into coherent per-paper summaries. Because extraction runs per-chunk rather than over the full corpus, it avoids information loss from context-window pressure seen in direct LLM baselines. This demonstrates that SLIDERS extends beyond tabular QA to open-ended synthesis tasks.

8 Reconciliation Agent Analysis

We perform an in-depth analysis of the reconciliation agent across five datasets, as shown in Figure 6. For the existing benchmarks, whose contexts are still relatively moderate ($\leq 360k$ tokens), reconciliation reduces extracted tables from 10.25 rows to 7.48 rows on average, with 7.22 unique primary keys per table. For the much larger document collections, WikiCeleb100 and FinQ100 contain 101 and 128 unique primary keys, respectively, making reconciliation a core part of the pipeline rather than a minor cleanup step.

Overall, the reconciliation agent is efficient, requiring only 1.28 iterations per primary-key group on average. However, the figure shows that its behavior differs substantially across datasets, both in how much reduction it achieves and in which operations it relies on.

Row reduction. Figure 6a,6d shows that reconciliation consistently reduces redundancy within primary-key groups. After reconciliation, the average number of rows per primary key is close to one for nearly all datasets. The largest effect appears on FinQ100, where the average rows per key drop from above 5 before reconciliation to nearly 1 after reconciliation. This is also reflected in the reduction-density plot, where FinQ100 is concentrated at high row-reduction rates, with many groups reduced by roughly 70-90%. WikiCeleb100 also shows substantial reduction, while FinanceBench and the Loong Finance splits exhibit moderate but consistent improvements. In contrast, Loong Legal changes very little: its documents are often shorter than the chunk size, so extraction already produces nearly one row per key and leaves little redundancy for reconciliation to remove.

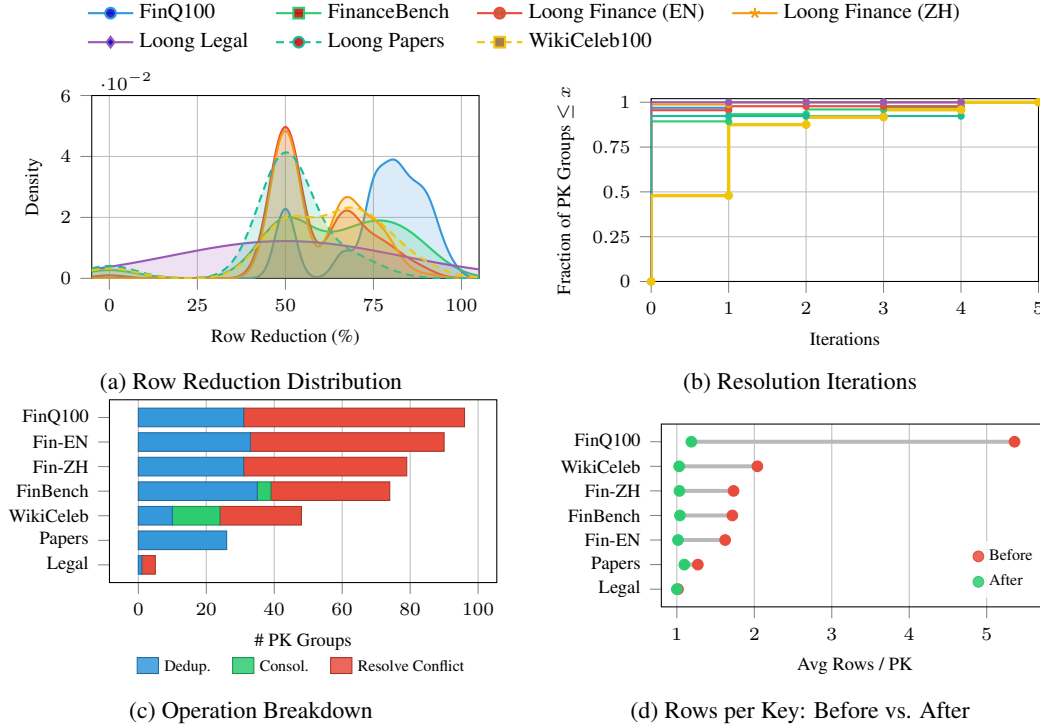


Figure 6: Analysis of the reconciliation agent across seven datasets: (a) density of per-group row reduction, (b) CDF of groups resolved within k iterations, (c) operation-type breakdown, and (d) average rows per primary key before and after reconciliation. The results show substantial reduction in redundant rows and rapid convergence in most datasets, with different benchmarks exhibiting different mixes of reconciliation operations.

Operations used. The operation breakdown in Figure 6b reveals different reconciliation regimes across datasets. FinQ100 and the Loong Finance datasets are dominated by conflict resolution, with deduplication as the next most common operation. This indicates that many chunks extract competing values for the same key, requiring the agent to compare provenance and rationales to decide which value to retain. WikiCeleb100 exhibits a different pattern: deduplication and consolidation are more prominent, suggesting that sources are often mutually consistent but contribute complementary partial records that must be merged into a canonical row. FinanceBench lies between these two extremes, using both conflict resolution and deduplication with only a small amount of consolidation. Long Papers are handled almost entirely by deduplication, while Loong Legal triggers almost no reconciliation operations.

Iterations to convergence. Despite often substantial row reduction, reconciliation usually converges quickly. Figure 6c shows that for most datasets, around 90% or more of primary-key groups are resolved in a single iteration, and nearly all groups are resolved within at most a few iterations. WikiCeleb100 is the main exception: only about half of its groups are resolved in the first iteration, and additional passes are needed before the CDF reaches 1.0. This suggests that the number of iterations is driven not simply by how many rows are present in a group, but by the structure of the ambiguity. In particular, FinQ100 has far more rows per key before reconciliation than WikiCeleb100, yet it converges in fewer iterations on average. A likely explanation is that many FinQ100 groups involve direct duplicate removal or selecting among conflicting values, whereas WikiCeleb100 more often requires iterative consolidation of complementary evidence across sources.

Takeaway. Overall, reconciliation does more than remove duplicate rows: it transforms noisy, local extractions into a near-canonical structured representation with approximately one row per primary key. The figure also highlights that reconciliation is highly dataset-dependent. Large financial corpora

primarily induce conflicts that must be resolved, whereas multi-document biographical corpora more often require consolidation across partially overlapping evidence.

9 Related Works

Long-Context Modeling and Systems. Prior work on long-context processing spans three broad categories. *Training-based methods* improve model capacity through position-agnostic training [He et al., 2024], multi-document instruction tuning [Liu et al., 2025], and representation-level approaches such as corpus-specific context memories [Eyuboglu et al., 2025] or RL-based compression into fixed working memory [Yu et al., 2025]. In contrast, SLIDERS processes each chunk independently and hence can process text beyond a single LLM’s context window. *Agentic frameworks* divide work across specialized roles: using leader–worker coordination [Zhao et al., 2024], orchestrating sequential reasoning [Zhang et al., 2024], traversing document-derived fact graphs [Li et al., 2024], and systems delegating sub-tasks across models of varying capability [Narayan et al., 2025, Zhang et al., 2025, Gidiotis and Tsoumakas, 2020], schema-driven extraction (DOCETL) [Shankar et al., 2025], or domain-specific retrieval [Sarmah et al., 2024, Han et al., 2024, Wang et al., 2025b, Choe et al., 2025]. However, chunk-based methods such as DocETL still face the fundamental *aggregation bottleneck* as the number of documents grows. In contrast, SLIDERS converts local extractions into a unified relational representation, reconciles them into a coherent global state, and reasons over that state using LLM-generated SQL. This yields auditable outputs with full provenance while reducing the need to aggregate large amounts of raw text at answer time.

Intermediate Representations and SQL Reasoning. Structured intermediate representations and SQL-based abstractions have emerged as effective tools for building reliable LLM systems. Text-to-SQL systems [Yu et al., 2018, Li et al., 2023] translate natural language into executable queries, others [Cheng et al., 2022, Liu et al., 2024] extends SQL with embedded LLM calls for semantic reasoning. [Patel et al., 2025] introduces semantic operators that extend the relational model with AI-based filtering, joining, and aggregation over unstructured text. Beyond SQL, formal intermediate representations have been used to compress context and improve reliability in dialogue systems [Joshi et al., 2025] and agentic workflows [Wang et al., 2025a]. SLIDERS uses structured reasoning for long-document QA.

10 Conclusion

We propose separating information representation from reasoning: SLIDERS extracts relevant data from document sets into a relational database with an automatically induced schema, then performs downstream reasoning via SQL queries. A data-reconciliation agent reasons over provenance and extraction rationales to consolidate partial information and resolve inconsistencies. Our approach outperforms baselines on existing benchmarks even when documents fit within context windows, and scales to corpora of 100 documents (up to 36M tokens) far exceeding current context limits beating baselines by upto 11 times.

Limitations. (1) SLIDERS relies on schema induction to structure extraction; while we show robustness across a $4\times$ schema complexity range, tasks that resist relational modeling (e.g., highly subjective or abstract cross-document reasoning) may not benefit. (2) The pipeline requires multiple sequential LLM calls, resulting in higher latency (2–3 min end-to-end) than single-call baselines; this is acceptable for accuracy-critical workflows but not for real-time applications. (3) Evaluation relies on LLM-as-a-judge, which we validate with human annotations (Cohen’s $\kappa = 0.758$), but residual noise remains. (4) SLIDERS achieves 55% on FinQ100, insufficient for full automation in high-stakes financial analysis. For safety-critical deployments, we recommend human-in-the-loop verification: SLIDERS’s provenance tracking enables reviewers to efficiently validate each extraction against source documents. When provenance is incorrect (e.g., wrong text span), the reconciliation agent may make faulty decisions; our verification of 410 facts found 99.03% correct, and users can inspect source quotes directly.

Acknowledgement

We thank Shicheng Liu, Nikil Selvam, Aryaman Arora, Yanzhe Zhang, Cyrus Zhou, Adit Negi, Vedant Singh, Tony Liu, Jiuding Sun, Sina Semnani and the members of the Stanford OVAL and Stanford NLP for helpful discussion about the project and comments on the manuscript. This work is supported in part by the Verdant Foundation, Hasso Plattner Institute, Microsoft Azure AI credits, Itaú Unibanco, BMO Financial Group, and the Stanford Human-Centered Artificial Intelligence (HAI) Institute.

References

- C. Auer, M. Lysak, A. Nassar, M. Dolfi, N. Livathinos, P. Vagenas, C. B. Ramis, M. Omenetti, F. Lindlbauer, K. Dinkla, et al. Docling technical report. *arXiv preprint arXiv:2408.09869*, 2024.
- Y. Bai, X. Lv, J. Zhang, H. Lyu, J. Tang, and Z. Huang. Zhengxiao du, xiao liu, aohan zeng, lei hou, et al. 2023. longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*, 2023.
- BBC. Representation of bbc news content in ai assistants. <https://www.bbc.co.uk/aboutthebbc/documents/bbc-research-into-ai-assistants.pdf>, Feb. 2025. Accessed: 2025-12-04.
- A. Bertsch, A. Pratapa, T. Mitamura, G. Neubig, and M. R. Gormley. Oolong: Evaluating long context reasoning and aggregation capabilities. *arXiv preprint arXiv:2511.02817*, 2025.
- I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):5–es, 2007.
- Z. Cheng, T. Xie, P. Shi, C. Li, R. Nadkarni, Y. Hu, C. Xiong, D. Radev, M. Ostendorf, L. Zettlemoyer, et al. Binding language models in symbolic languages. *arXiv preprint arXiv:2210.02875*, 2022.
- J. Choe, J. Kim, and W. Jung. Hierarchical retrieval with evidence curation for open-domain financial question answering on standardized documents. In W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, editors, *Findings of the Association for Computational Linguistics: ACL 2025*, pages 16663–16681, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.855. URL <https://aclanthology.org/2025.findings-acl.855/>.
- G. Del Fiore, T. E. Workman, and P. N. Gorman. Clinical questions raised by clinicians at the point of care: a systematic review. *JAMA internal medicine*, 174(5):710–718, 2014.
- D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, and J. Larson. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.
- S. Eyuboglu, R. Ehrlich, S. Arora, N. Guha, D. Zinsley, E. Liu, W. Tennien, A. Rudra, J. Zou, A. Mirhoseini, et al. Cartridges: Lightweight and general-purpose long context representations via self-study. *arXiv preprint arXiv:2506.06266*, 2025.
- I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American statistical association*, 64(328):1183–1210, 1969.
- A. Gidiotis and G. Tsoumakas. A divide-and-conquer approach to the summarization of long documents. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28:3029–3040, 2020.
- O. Goldman, A. Jacovi, A. Slobodkin, A. Maimon, I. Dagan, and R. Tsarfaty. Is it really long context if all you need is retrieval? towards genuinely difficult long context nlp. *arXiv preprint arXiv:2407.00402*, 2024.
- A. Gu and T. Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First conference on language modeling*, 2024.

- S. Han, H. Kang, B. Jin, X.-Y. Liu, and S. Y. Yang. Xbrl agent: Leveraging large language models for financial report analysis. In *Proceedings of the 5th ACM International Conference on AI in Finance*, ICAIF '24, page 856–864, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400710810. doi: 10.1145/3677052.3698614. URL <https://doi.org/10.1145/3677052.3698614>.
- J. He, K. Pan, X. Dong, Z. Song, L. LiuYiBo, Q. Qianguosun, Y. Liang, H. Wang, E. Zhang, and J. Zhang. Never lost in the middle: Mastering long-context question answering with position-agnostic decompositional training. In L.-W. Ku, A. Martins, and V. Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13628–13642, Bangkok, Thailand, Aug. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.736. URL <https://aclanthology.org/2024.acl-long.736/>.
- Z. Hong, Z. Yuan, Q. Zhang, H. Chen, J. Dong, F. Huang, and X. Huang. Next-generation database interfaces: A survey of llm-based text-to-sql. *IEEE Transactions on Knowledge and Data Engineering*, 2025.
- C.-P. Hsieh, S. Sun, S. Kriman, S. Acharya, D. Rekish, F. Jia, Y. Zhang, and B. Ginsburg. Ruler: What’s the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.
- P. Islam, A. Kannappan, D. Kiela, R. Qian, N. Scherrer, and B. Vidgen. Financebench: A new benchmark for financial question answering. *arXiv:2311.11944*, 2023.
- A. Jain, P. Aggarwal, and A. Saladi. Autochunker: Structured text chunking and its evaluation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pages 983–995, 2025.
- Z. Jiang, X. Shi, and J. Lin. Longrag: Enhancing retrieval-augmented generation with long-context llms. *arXiv preprint arXiv:2406.15319*, 2024.
- C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.
- H. Joshi, S. Liu, J. Chen, L. Weigle, and M. Lam. Controllable and reliable knowledge-intensive task-oriented conversational agents with declarative genie worksheets. In W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 27264–27308, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1323. URL <https://aclanthology.org/2025.acl-long.1323/>.
- P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.
- H. Li, J. Zhang, C. Li, and H. Chen. Resdsq: Decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13067–13075, 2023.
- H. Li, X. Chen, Z. Xu, D. Li, N. Hu, F. Teng, Y. Li, L. Qiu, C. J. Zhang, L. Qing, et al. Exposing numeracy gaps: A benchmark to evaluate fundamental numerical abilities in large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 20004–20026, 2025.
- S. Li, Y. He, H. Guo, X. Bu, G. Bai, J. Liu, J. Liu, X. Qu, Y. Li, W. Ouyang, W. Su, and B. Zheng. GraphReader: Building graph-based agent to enhance long-context abilities of large language models. In Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 12758–12786, Miami, Florida, USA, Nov. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.746. URL <https://aclanthology.org/2024.findings-emnlp.746/>.

- G. K.-M. Liu, B. Shi, A. Caciularu, I. Szpektor, and A. Cohan. MDCure: A scalable pipeline for multi-document instruction-following. In W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 29258–29296, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1418. URL <https://aclanthology.org/2025.acl-long.1418/>.
- N. F. Liu, K. Lin, J. Hewitt, et al. Lost in the middle: How language models use long contexts. *arXiv:2307.03172*, 2023a.
- N. F. Liu, T. Zhang, and P. Liang. Evaluating verifiability in generative search engines. *arXiv preprint arXiv:2304.09848*, 2023b.
- S. Liu, J. Xu, W. Tjangnaka, S. Semnani, C. Yu, and M. Lam. SUQL: Conversational search over structured and unstructured data with large language models. In K. Duh, H. Gomez, and S. Bethard, editors, *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 4535–4555, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-naacl.283. URL <https://aclanthology.org/2024.findings-naacl.283/>.
- W. C. Mann and S. A. Thompson. Rhetorical structure theory: Toward a functional theory of text organization. *Text-interdisciplinary Journal for the Study of Discourse*, 8(3):243–281, 1988.
- I. Mirzadeh, K. Alizadeh, H. Shahrokhi, O. Tuzel, S. Bengio, and M. Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*, 2024.
- A. Narayan, D. Biderman, S. Eyuboglu, A. May, S. Linderman, J. Zou, and C. Re. Cost-efficient collaboration between on-device and cloud language models. In *Forty-second International Conference on Machine Learning*, 2025.
- H. B. Newcombe. Record linking: the design of efficient systems for linking records into individual and family histories. *American Journal of Human Genetics*, 19(3 Pt 1):335, 1967.
- G. Papadakis, J. Svirsky, A. Gal, and T. Palpanas. Comparative analysis of approximate blocking techniques for entity resolution. *Proceedings of the VLDB Endowment*, 9(9):684–695, 2016.
- L. Patel, S. Jha, M. Pan, H. Gupta, P. Asawa, C. Guestrin, and M. Zaharia. Semantic operators and their optimization: Enabling llm-based data processing with accuracy guarantees in lotus. *Proceedings of the VLDB Endowment*, 18(11):4171–4184, 2025.
- Qwen Team. Qwen3.5: Towards native multimodal agents. <https://qwen.ai/blog?id=qwen3.5>, Feb. 2026. Blog post.
- B. Sarmah, D. Mehta, B. Hall, R. Rao, S. Patel, and S. Pasquali. Hybridrag: Integrating knowledge graphs and vector retrieval augmented generation for efficient information extraction. In *Proceedings of the 5th ACM International Conference on AI in Finance, ICAIF '24*, page 608–616, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400710810. doi: 10.1145/3677052.3698671. URL <https://doi.org/10.1145/3677052.3698671>.
- S. Shankar, T. Chambers, T. Shah, A. G. Parameswaran, and E. Wu. Docetl: Agentic query rewriting and evaluation for complex document processing. *Proc. VLDB Endow.*, 18(9):3035–3048, Sept. 2025. ISSN 2150-8097. doi: 10.14778/3746405.3746426. URL <https://doi.org/10.14778/3746405.3746426>.
- L. B. Sollaci and M. G. Pereira. The introduction, methods, results, and discussion (imrad) structure: a fifty-year survey. *Journal of the medical library association*, 92(3):364, 2004.
- R. Van De Schoot, J. De Bruin, R. Schram, P. Zahedi, J. De Boer, F. Weijdem, B. Kramer, M. Huijts, M. Hoogerwerf, G. Ferdinands, et al. An open source machine learning framework for efficient and transparent systematic reviews. *Nature machine intelligence*, 3(2):125–133, 2021.
- H. Wang, C. M. Poskitt, and J. Sun. Agentspec: Customizable runtime enforcement for safe and reliable llm agents. *arXiv preprint arXiv:2503.18666*, 2025a.

- M. Wang, L. Chen, C. Fu, et al. Leave no document behind: Benchmarking long-context llms with extended multi-doc qa (loong). In *EMNLP*, 2024. arXiv:2406.17419.
- X. Wang, J. Chi, Z. Tai, T. S. T. Kwok, M. Li, Z. Li, H. He, Y. Hua, P. Lu, S. Wang, et al. Finsage: A multi-aspect rag system for financial filings question answering. *arXiv preprint arXiv:2504.14493*, 2025b.
- A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.
- H. Yu, T. Chen, J. Feng, J. Chen, W. Dai, Q. Yu, Y.-Q. Zhang, W.-Y. Ma, J. Liu, M. Wang, et al. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent. *arXiv preprint arXiv:2507.02259*, 2025.
- T. Yu, R. Zhang, et al. Spider: A large-scale human-labeled dataset for text-to-sql. In *EMNLP*, 2018.
- A. L. Zhang, T. Kraska, and O. Khattab. Recursive language models, 2025. URL <https://arxiv.org/abs/2512.24601>.
- Y. Zhang, T. Du, Y. Sun, L. Donohue, and R. Dai. Form 10-q itemization. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 4817–4822, 2021.
- Y. Zhang, R. Sun, Y. Chen, T. Pfister, R. Zhang, and S. Arik. Chain of agents: Large language models collaborating on long-context tasks. *Advances in Neural Information Processing Systems*, 37: 132208–132237, 2024.
- J. Zhao, C. Zu, X. Hao, Y. Lu, W. He, Y. Ding, T. Gui, Q. Zhang, and X. Huang. LONGAGENT: Achieving question answering for 128k-token-long documents through multi-agent collaboration. In Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16310–16324, Miami, Florida, USA, Nov. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.912. URL <https://aclanthology.org/2024.emnlp-main.912/>.
- Z. Zhou, C. Li, X. Chen, S. Wang, Y. Chao, Z. Li, H. Wang, Q. Shi, Z. Tan, X. Han, X. Shi, Z. Liu, and M. Sun. LLM×MapReduce: Simplified long-sequence processing using large language models. In W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 27664–27678, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1341. URL <https://aclanthology.org/2025.acl-long.1341/>.

A Additional Methodology Details

A.1 Question Decomposition

The original question posed by the user is not directly suitable for all stages of the pipeline. In particular, extraction and reconciliation are intermediate steps that impose different requirements on task formulation. Extraction should enumerate all relevant candidate evidence from each document chunk, while reconciliation should combine these candidates into a globally consistent database. Conditioning either stage on the original question can cause them to prematurely answer the question rather than constructing the intermediate representation.

Consider the query: “Return the second poem about the Great Wall of China.” Suppose the first chunk contains one relevant poem (P1), and the second chunk contains two relevant poems (P2, P3). If the extractor is conditioned on the original question, it may incorrectly interpret the ordinal constraint “second” at the chunk level: it extracts nothing from the first chunk (since no second poem exists locally) and extracts P3 from the second chunk (as the second poem within that chunk). This results in an incorrect global outcome, since the true second poem across the entire document collection is P2. Similar issues arise for other global constraints such as superlatives, rankings, and aggregations.

To prevent such premature answer-oriented behavior, SLIDERS decomposes the input question q into component-specific queries. We generate an extraction query q_e that specifies what information should be identified from each document chunk, and a reconciliation query q_r that defines how partial extractions should be combined and cleaned to form a coherent database. Schema induction and final answer synthesis, in contrast, operate directly on the original question q , as these stages are explicitly responsible for capturing the full semantic intent of the user’s request and producing the final answer.

A.2 Context-Aware Chunking

Each chunk retains the raw text as well as its structural metadata, including the current heading path (e.g., Header 1 → Header 1.1 → Header 1.1.3), the chunk index, the document title, and the document description. This ensures that every chunk is self-contained, carrying both global and local context needed for faithful extraction.

A.3 Schema Induction

Here we provide the schema library we provide SLIDERS.

```
# Question Type Guidelines

## Ordering and Retrieval Questions
- The schema should not contain the index of the item since extraction will be done at chunk level,
  and won't have the global index.

## Multiple Choice Questions
- The schema should contain reasons for and against each option.
- Include fields such as 'option_label' (e.g., 'option A name', 'option B name', 'option C name
  ...), 'option_text', and 'support_label'.
- You can use enums as a field which says: 'supported', 'not_supported', 'insufficient_evidence', '
  contradicted'.

# Document Type Guidelines

## Dataset Documents
- You should include any given identifier so that datapoints can be aligned across pages/chunks.
- Include at least one label field (e.g., 'label') plus any other attributes needed to answer or
  aggregate for the question.

## Story/Narrative Documents
- Define an 'Entity' table to capture the relationship 'entity  $\rightarrow$  properties
  mentioned on this page'.
- Include fields such as 'entity_name', 'entity_type', and a field summarizing important properties
  on this page (e.g., 'entity_information').
- If the question depends on actions or plot, define an 'Event' (or similar) table representing '
  event  $\rightarrow$  participants/time/location', with one row per event.
- Use canonicalize the names across chunks when possible so that information about the same entity/
  event can be merged across chunks.

## Policy/Whitepaper Documents
- You should have fields like stakeholders, implementation steps, conditions, etc
- If it talks about recommendations, then you make sure to add fields like target_area/domain,
  intended outcome, advantages, disadvantages, etc
```

A.4 SLIDERS implementation details

Metadata Extraction To identify headers, tables and figures in a document, we use DocLing Auer et al. [2024], an off-the-shelf pdf to markdown converter and parse the markdown to identify the structural metadata. In scenarios where the original pdf is not available, we preprocess the raw text to identify, tables and section headers using LLMs.

LLM Models We use GPT 4.1 and GPT 4.1 mini for our experiments. For parts that require planning and thinking such as schema generation and reconciliation agent, we use GPT 4.1. For other tasks such as extraction from chunks, the metadata generation, we use GPT 4.1 mini.

A.5 Sample Configuration

We provide a representative configuration file for the Loong Finance (EN) benchmark.

```
experiment: loong
system: sliders
system_config:
  generate_task_guidelines: false
  rephrase_question:
    enabled: true
    prompt_file: sliders/rephrase_question_component.prompt
    library_of_guidelines_path: sliders_taxonomy.json
  generate_schema:
    add_extra_information_class: false
    add_document_text: true
    generate_schema_type: library_based
    library_of_guidelines_path: sliders_taxonomy.json
  extract_schema:
    decompose_fields: false
    dedupe_merged_rows: false
    num_samples_per_chunk: 1
    is_relevant_chunk: true
    extract_quotes: true
  merge_tables:
    merge_strategy: recon_v2
  reconciliation:
    debug_mode: false
    primary_key_selection:
      version: v1
      max_candidates: 10
      max_inspections_per_candidate: 10
      max_retries: 3
    canonicalization:
      mode: two_pass
      max_cycles: 20
      max_retries_per_cycle: 3
      max_inspection_history: 3
      inspections_per_field: 50
      null_handling:
        max_inspections: 5
        placeholder_text: UNKNOWN
      verification:
        enable: false
        max_inspections: 5
    controller_executor_loop:
      max_iterations: 5
      max_controller_inspections: 5
      max_executor_inspections: 5
      max_sql_attempts: 3
      verification:
        enable: true
        max_inspections: 5
    context_generator:
      enable: true
      model: gpt-4.1
      max_rows: 20
    non_pk_canonicalization:
      enable: true
      inspections_per_field: 5
      column_selector:
        max_inspections: 5
        excluded_columns:
          - row_id
          - page_number
          - __reconciliation_context__
          - number_instances
      excluded_patterns:
        - _quote
        - _rationale
        - _confidence
    statistics:
      enable: true
      filename: reconciliation_stats.json
inspect_answer:
```

```

enable_citation_generation: true
enable_reconciliation_stats_verbalization: true
check_if_merge_needed: false
force_sql: false
answer_strategy: sql_inspect
models:
  answer:
    model: gpt-4.1
    max_tokens: 8192
    temperature: 0.0
  answer_no_table:
    model: gpt-4.1-mini
    max_tokens: 8192
    temperature: 0.0
  answer_tool_output:
    model: gpt-4.1
    max_tokens: 8192
    temperature: 0.0
  extract_schema:
    model: gpt-4.1-mini
    max_tokens: 32000
    temperature: 0.0
  generate_schema:
    model: gpt-4.1
    max_tokens: 8192
    temperature: 0.0
  merge_tables:
    model: gpt-4.1
    max_tokens: 32000
    temperature: 0.0
  task_guidelines:
    model: gpt-4.1
    max_tokens: 8192
    temperature: 0.0
  check_objective_necessity:
    model: gpt-4.1
    max_tokens: 8192
    temperature: 0.0
  rephrase_question:
    model: gpt-4.1
    max_tokens: 8192
    temperature: 0.0
  direct_answer:
    model: gpt-4.1
    max_tokens: 8192
    temperature: 0.0
  force_answer:
    model: gpt-4.1
    max_tokens: 8192
    temperature: 0.0
  is_relevant_chunk:
    model: gpt-4.1-mini
    max_tokens: 8192
    temperature: 0.0
  check_if_merge_needed:
    model: gpt-4.1
    max_tokens: 8192
    temperature: 0.0
  select_guidelines_for_schema:
    model: gpt-4.1-mini
    max_tokens: 2000
    temperature: 0.0
  select_primary_key:
    model: gpt-4.1
    max_tokens: 8192
    temperature: 0.0
  canonicalize_fields:
    model: gpt-4.1
    max_tokens: 8192
    temperature: 0.0
  column_selector:
    model: gpt-4.1
    max_tokens: 8192
    temperature: 0.0
  context_generator:
    model: gpt-4.1
    max_tokens: 8192
    temperature: 0.0
  handle_null_pks:
    model: gpt-4.1
    max_tokens: 8192
    temperature: 0.0
experiment_config:
  benchmark_path: /data1/hypothesis_dataset/loong/loong-processed.jsonl
  gpt_results_path: None
  files_dir: /data1/hypothesis_dataset/loong/doc/
  specific_ids_csv: sample_data/50_sample_ids_finance_en_final.csv
  soft_evaluator_model: gpt-4.1
  hard_evaluator_model: gpt-4.1
  num_questions: null
  random_state: 42
  document_config:

```

```
chunk_size: 16000
overlap_size: 0
filter_by_type: "financial"
output_file: loong_sliders_finance_en_26jan.json
```

A.6 Cost Analysis

Table 7 presents the average cost per question in US dollars for SLIDERS across different benchmarks. Costs vary based on document length, complexity, and the number of reconciliation operations required.

Table 7: Average cost per question (in USD) across benchmarks

Benchmark	Avg. Cost (\$)
Oolong	1.30
Loong Legal	0.65
Loong Finance (ZH)	0.79
Loong Finance (EN)	0.60
Loong Papers	0.37
FinanceBench	0.63

OoLong has the highest per-question cost due to its longer input contexts, while Loong Papers has the lowest cost with shorter academic papers requiring fewer reconciliation steps.

A.7 RLM with GPT-4.1

To justify the model choice for RLM (GPT 5 / GPT 5-mini), we also ran RLM with GPT 4.1 (main) and GPT 4.1-mini (sub-LLM), matching SLIDERS’ configuration. As shown in Table 8, RLM performs worse with GPT 4.1, confirming that our main comparison gives RLM an advantage. SLIDERS with GPT 4.1 still outperforms RLM with GPT 5 across all benchmarks.

Table 8: RLM performance with different base models.

System	Loong	FinanceBench
RLM (GPT 5 & GPT 5-mini)	72.64	75.33
RLM (GPT 4.1 & GPT 4.1-mini)	60.13	72.00
SLIDERS (GPT 4.1 & GPT 4.1-mini)	78.57	88.67

B Benchmark Creation

Question construction begins from a seed query: for WikiCeleb100, “Which artist made their debut at the youngest age across Film, Music, Content Creation, and Other industries?”; and for FinQ100, “Which company has the lowest long-term borrowing?” We derive follow-up questions by reusing and recombining the extracted information, including temporal cohorts (e.g., artists debuting in the 1950s or born in the 1970s) and aggregate financial properties (e.g., companies with no long-term borrowing or borrowing exceeding \$80M).

To annotate gold answers, we apply SLIDERS to extract structured information and manually verify the extracted values. We then manually reconcile the resulting tables to obtain a consolidated database representation of each document set, and manually author SQL queries for each question. For each benchmark, we additionally retain five questions that cannot be answered using SQL alone.

B.1 Questions and Answers

We provide all questions from both benchmarks.

B.1.1 WikiCeleb100

1. *Which industry has the most artist debuts - Music, Film, or Content Creation?*
Answer: Music
2. *What are the top 3 countries of birth for artists in the dataset?*
Answer: United States, India, England
3. *Which artist has the earliest debut year in the dataset?*
Answer: Vera-Ellen
4. *Are there more artists who debuted in Music or Film?*
Answer: Music
5. *Which decade saw the most artist debuts?*
Answer: 2010.0
6. *What is the distribution of artists across different industries?*
Answer: Music, Film, Other, Content Creation
7. *Show the trend of artist debuts from earliest to latest decade - is it increasing, decreasing, or stable?*
Answer: increasing
8. *Which industry tends to have artists debuting earlier on average - Music or Film?*
Answer: Film
9. *What are the top 2 countries for artist births?*
Answer: United States, India
10. *Among artists born outside the US, which countries contribute the most?*
Answer: India, England, South Korea, Malaysia, Sweden
11. *Do more artists debut in the first half of a decade (years 0-4) or the second half (years 5-9)?*
Answer: First half
12. *For each of the top 5 countries, which industry is most dominant among their artists?*
Answer: United States, India, England, British India, South Korea
13. *Which artists have crossed over into multiple industries?*
Answer: Lesley Hornby, Olajide Olayinka Williams Olatunji, Wee Meng Chee, Zendaya Maree Stoermer Coleman, Aliaune Damala Bouga Time Puru Nacka Lu Lu Lu Badara Akon Thiam, ...
14. *Is the Music industry growing or declining over time compared to Film? Compare debuts by decade.*
Answer: declined marginally till 1980 and then grew steadily
15. *Among artists who work in multiple industries, is it more common to start in Music then move to Film, or vice versa?*
Answer: mmore
16. *What is the typical gap in years between an artist's Music debut and their Film debut?*
Answer: 4.6
17. *Which birth places have produced artists in the most diverse range of industries?*
Answer: Atlanta, Georgia, U.S., Muar, Johor, Malaysia, Oakland, California, Neasden, Middlesex, England, London, ...
18. *What proportion of artists from India work in Film compared to artists from the United States?*
Answer: more
19. *Who are the artists with the most recent debuts, and what industries are they in?*
Answer: Xavier James Trudeau, Ciara Mary-Alice Thompson, HorsegiirL, Darren Jason Watkins Jr., Victoria Beverley Walker
20. *Is there a trend showing Content Creation becoming more common in recent decades compared to traditional industries?*
Answer: Relatively more common in recent decades but still not as common as music or film
21. *Which artist debuted at the youngest age?*
Answer: Raven-Symon Christina Pearman-Maday
22. *Which artist debuted at the youngest age in each industry?*
Answer: Film, Music, Content Creation, Other

B.1.2 FinQ100

1. *Which company has the highest long-term borrowings?*
Answer: CARMAX INC
2. *What are the top 5 companies by long-term borrowings?*
Answer: CARMAX INC, MICRON TECHNOLOGY INC, AUTOZONE INC, Salesforce, Inc., COSTCO WHOLESALE CORP NEW
3. *Are there more companies with zero long-term debt or with debt over 1 billion?*
Answer: debt_category: Zero debt, more_or_less: more
4. *Which reporting period has the most company filings?*
Answer: 2025-10-31
5. *What are the companies with zero long-term debt?*
Answer: Apex Treasury Corp, BRB Foods Inc., CALERES INC, Cantor Equity Partners V, Inc., Dynamix Corp III, ...
6. *What is the distribution of companies across different debt ranges (0, 0-100M, 100M-1B, 1B+)?*
Answer: 1. Zero, 2. Under 100M, 3. 100M - 1B, 4. Over 1B
7. *How does the median long-term borrowings compare to the mean? Is the distribution skewed?*
Answer: Mean is more than median. Right skewed.
8. *What percentage of total long-term borrowings is held by the top 5 companies?*
Answer: 49.2
9. *Which companies have reported borrowings for multiple periods?*
Answer: IIOT-OXYS, Inc., TJX COMPANIES INC /DE/, VAIL RESORTS INC
10. *What is the range of long-term borrowings (min to max) and how spread out is the data?*
Answer: "min_borrowings": 0.0, "max_borrowings": 16586.77, "range_spread": 16586.77, "std_deviation": 2645.37
11. *Are companies reporting in October vs November showing different average borrowing levels?*
Answer: Borrowing levels in November are lower
12. *What is the ratio of companies with high debt (>1B) to companies with low/no debt (<100M)?*
Answer: 0.44
13. *If we exclude the top 3 largest borrowers, how does the average borrowing change?*
Answer: Decreases by 395.4
14. *Which companies have borrowings closest to the median value (within 25% of median)?*
Answer: KESTRA MEDICAL TECHNOLOGIES, LTD., Oil-Dri Corp of America, SPORTSMAN'S WAREHOUSE HOLDINGS, INC.
15. *What is the cumulative percentage of total borrowings as we go from smallest to largest companies?*
Answer: CARMAX INC, MICRON TECHNOLOGY INC, AUTOZONE INC, Salesforce, Inc., COSTCO WHOLESALE CORP /NEW, ...
16. *Among companies with CIK numbers, which sectors (based on CIK ranges) have higher average borrowings?*
Answer: Mid registrants
17. *What is the total long-term borrowings across all companies in the dataset?*
Answer: 102590.76
18. *For TJX Companies which has multiple periods, did their long-term borrowings increase or decrease over time?*
Answer: decreased
19. *What percentage of companies have long-term borrowings under 50 million?*
Answer: 53.7
20. *How concentrated is the debt? What percentage of companies hold 80% of the total borrowings?*
Answer: 15
21. *Are companies in the 'Recent registrants' CIK range (1M-1.5M) more likely to have zero debt compared to 'Mid registrants' (500K-1M)?*
Answer: Recent

22. *Among companies with borrowings over 1 billion, which has the lowest borrowings?*
Answer: ROSS STORES, INC.
23. *Do more companies have borrowings above or below 100 million?*
Answer: Below 100M
24. *What are the bottom 5 companies by long-term borrowings (excluding zero debt companies)?*
Answer: LIBERTY STAR URANIUM & METALS CORP., OFF THE HOOK YS INC., USA OPPORTUNITY INCOME ONE, INC., Rise Gold Corp., IIOT-OXYS, Inc.
25. *Which company has the lowest non-zero long-term borrowings?*
Answer: LIBERTY STAR URANIUM & METALS CORP.

Complexity of writing SQL programs. The SQL programs written by the reconciliation Agent are non-trivial as shown in Listing 1.

Listing 1: Computing depreciation-to-revenue margin for AMD

```
SELECT
  d.depreciation_amortization_expense,
  t.total_revenue,
  ROUND(
    100.0 * d.depreciation_amortization_exp /
    t.total_revenue,
    2
  ) AS da_margin_percent
FROM DepreciationAmortizationExpense AS d
JOIN TotalRevenue AS t
  ON d.company_name = t.company_name
  AND d.fiscal_year = t.fiscal_year
WHERE d.company_name = 'AMD'
  AND d.fiscal_year = 2015
  AND d.depreciation_amortization_exp IS NOT NULL
  AND t.total_revenue IS NOT NULL
LIMIT 1;
```

Listing 2: An entity resolution SQL for inspection

```
SELECT
  product_or_service_name,
  COUNT(*) AS cnt
FROM CompanyProductOrService_2063d536_AMD_2022_10K
GROUP BY
  product_or_service_name
ORDER BY
  product_or_service_name,
  cnt DESC;
```

C Example of Issues Discovered by SLIDERS

During error analysis, we identified cases where the ground truth annotations in FinanceBench and Loong are incorrect. Because SLIDERS traces provenance, we can verify whether a predicted answer is correct or the ground truth itself is erroneous. We identified two types of errors: (1) answers that are spread across multiple pages, and (2) incorrect or incomplete annotations. For FinanceBench, which assumes answers are localized to a single page, we found more complete answers spanning multiple pages for question IDs `financebench_id_02416`, `financebench_id_00283`. For Loong in the financial domain, we found ground truth entries with missing units and incorrect values; for example, for a question the annotation says the accounts payable is 1,740 instead of the correct value of 1,663 thousand. This is the same example we provide in Figure 2. These findings demonstrate the auditability and interpretability of SLIDERS.

D Prompts used by SLIDERS

D.1 System Prompts

We provide all prompts used in the SLIDERS pipeline, ordered by their execution sequence. Prompts are displayed in a monospace font with syntax highlighting disabled for readability.

D.1.1 Schema Generation

Schema Induction Generates a structured schema (tables and fields) based on the question and document descriptions.

```
# instruction
Given description of documents, sample document content, and a question, define the tables and fields (including
intermediate/supporting fields) that, once extracted at the page level and merged later, will let us answer the
question.

Table Modeling rules:
- One table = one relationship. If the relationship logically decomposes (e.g., case summary vs per-counsel arguments),
  define multiple tables.
- No nested types. When you would need a list/array, model it as multiple rows in the appropriate table.
- Include supporting fields. Add any inputs required for downstream computation/validation (e.g., numerators/
  denominators, period markers, currency).
- Normalize by design. Each numeric field must declare a single unit and scale; extractors will normalize values into
  that target.

Field Specs (required keys per field):
- name: string (concise, machine-friendly).
- data_type: one of str | int | float | decimal | bool | date | enum.
- Prefer decimal for money/precise quantities; date for dates.
- enum_values: list of strings (required iff data_type == "enum").
- unit: string or None (e.g., USD, %, shares; use None if not applicable).
- scale: one of none | thousands | millions | billions | trillions | basis_points | log10 | ln.
  - Values will be stored in this scale.
  - Example: $1.23B with scale="millions" $\rightarrow$ store 1230, unit="USD".
- description: self-contained extraction instruction (what the value represents; common surface forms/synonyms).
{% if enable_extraction_guidelines %}- extraction_guideline: string or None (optional but recommended). A concrete
  instruction for the extractor on HOW to extract this field.
  - Use this to clarify extraction logic that isn't obvious from the description alone.
  - Specify the SOURCE of the value (e.g., which document/entity the field refers to).
  - Clarify SCOPE constraints (e.g., what to include/exclude, how to handle ambiguity).
  - Provide FORMAT guidance (e.g., normalization rules, handling incomplete data).
  - Keep it concise (1-2 sentences max), specific to this field, and actionable.
{% endif %}- normalization (object; optional keys as needed):
  - currency: target code (e.g., "USD") or "as_is".
  - percent: "0_100" or "0_1".
  - date_format: e.g., "YYYY-MM-DD".
- required: boolean (expected presence; still None if missing on a page).

General Rules:
- Each table represents a relationship. If modeling the relationship requires multiple relations, define multiple
  tables (e.g., a "CaseSummary" table for case-level facts and a "CounselArguments" table for per-counsel arguments
  ).
- No nested types. If the extraction needs multiple items (e.g., many arguments), the extractor will create multiple
  rows in the appropriate relationship table.
- Include any linking fields needed to relate tables later (e.g., case_title, docket_number, entity_name, period_end).
- Normalize all values as specified below so independently run extractors produce standardized outputs.
- If a value is missing/unknown, set the value to None (not 0, empty string, or placeholder text).
- Page/section metadata is precomputed; do not include it in fields.
- The schema must describe only document-grounded relationships and fields, not the question itself. Do not add fields
  that simply restate the question.

Output:
- Emit JSON.
- For each table: include name, description, and a fields array.
- Do not include page metadata (it's precomputed).

# input
# Question
{{ question }}

# Documents description
{{ document_description }}
{% if document_text %}

# Sample Document Content
{{ document_text }}
{% endif %}

{% if guidelines %}
{{guidelines}}
{% endif %}

These are reserved keywords for fields. DO NOT USE THEM: 'row_id', 'document_name', 'metadata'.
```

D.1.2 Information Extraction

Chunk Relevance Filtering Determines whether a document chunk contains information relevant to answering the question.

```
# instruction
You will be given a single page from a larger document along with a question. The extraction will be done on this page.
```

Your task is to determine whether this page contains ANY information that could be useful for answering the question once this page is combined with other pages later.

Treat the page as RELEVANT if it satisfies ANY of the following:

- It contains a direct or partial answer to the question.
- It contains definitions, conditions, steps, list items, examples, or descriptions that relate to the question.
- It narrows down possibilities or provides context that a reconciliation agent could use (e.g., "the first principle is...", "see Section 3 for...", "Step 2: ...").

Treat the page as IRRELEVANT only if:

- None of the content on the page has any meaningful connection to the question, and
- It only contains metadata values, and doesn't explicitly mention useful fields. For example, if the page only contains the title, company name, then ignore this page.
- There is no text, that could help answer the question in any way during later aggregation.

First, briefly explain your reasoning in natural language. Then say if its relevant or not.

```
# input
# Question
{{question}}

# Document page
{{document}}
```

Table Extraction Extracts structured data from document chunks according to the generated schema.

```
# instruction
Given a schema and a single page, emit rows per table using the schema's field names/types---and per-field evidence:
- value (normalized to the field's unit/scale). If scale is None and the page content says 1.3 billion, then write the
  value as 1300000000.
{% if extract_quotes %}
- quote (largest contiguous exact text; list of strings or None)
{% else %}
- quote (always null; do not extract snippets when quotes are disabled)
{% endif %}
- rationale (why this value; include table headers/context if from a table)
- is_explicit (boolean)
- confidence (string; one of "Very High", "High", "Medium", "Low", "Very Low")

Confidence Rubric:
- Very High
  - The value is explicitly stated verbatim or via a trivial transformation (e.g., unit normalization) in the current
    text span, unambiguously refers to the target entity, and fully satisfies the schema field definition.
  - Surrounding context confirms the interpretation without inference.
- High
  - The value is explicitly stated, but either (a) requires light interpretation to map to the schema (e.g., paraphrase,
    implicit unit), or (b) could plausibly appear elsewhere in the document, even though the current span provides
    sufficient evidence.
  - Entity reference and schema alignment are still clear.
- Medium
  - The value is not explicitly stated but can be reasonably inferred from the text through interpretation, aggregation,
    or implicit assumptions.
  - The match to the schema is partial or approximate, and alternative interpretations are plausible.
- Low
  - The value is a speculative guess derived from weak textual cues, heuristics, or world knowledge.
  - Evidence is indirect, incomplete, or ambiguous, and the extractor believes the value is likely incorrect.
  - For entity relationships: Inferred from abbreviated references (acronyms, short names, citation numbers) without
    the full entity name appearing on the page.
- Very Low
  - No supporting evidence exists in the text.
  - The value is a placeholder, default, or fabrication inserted to satisfy schema completeness and should not be
    trusted or used downstream.

{% if enable_verbalization_instructions %}
## Critical: Exhaustive List Processing

**APPLIES TO:** Any section containing a numbered or bulleted list of related entities, including but not limited to:
- Sections titled "References", "Bibliography", "Citations", "Works Cited", "Related Work", "Dependencies", "Components",
  "Exhibits", "Appendices"
- Dependency lists, package catalogs, component inventories, related entity lists
- Any enumerated list where entities might be mentioned

**REQUIRED PROCESS:**
1. **In 'verbalization' field:** List ALL target entities from the question you're looking for
  - **CRITICAL:** Include EVERY entity mentioned ANYWHERE in the question---this includes the primary entity being
    analyzed AND all comparison/related entities
  - Example: If the question asks about relationships involving entities X, Y, and Z, then your target entities list
    should be [X, Y, Z]---NOT just a subset
  - When processing ANY document, check for ALL entities from the question, regardless of which document you're
    currently analyzing
2. **Identify the list boundaries** - Find where the list starts and ends
3. **Process EVERY entry sequentially** - Go through each numbered/bulleted item one by one (e.g., [1], [2], [3]... to
  the last entry)
4. **In 'verbalization' field:** For EACH AND EVERY entry, you MUST document:
  - **EXPLICITLY LIST EACH ENTRY** - Write: "Entry [N] 'identifier/name found here' - MATCH to <target entity name>"
    OR "Entry [N] 'identifier/name found here' - no match"
  - **DO NOT SUMMARIZE** - List each entry individually even if there are 100+ entries
```

- ****CRITICAL****: Even if you don't see matches initially, you MUST list every single entry. Saying "Checked entries [1]-[92], none match" or "The section lists many entries but none match" is WRONG - you must list each entry explicitly
- 5. ****In 'tables' field****: Extract a row immediately when you find a match (High/Very High confidence, with concise rationale)
 - ****CRITICAL****: For EVERY entry you marked as "MATCH" in your verbalization, you MUST create a corresponding extraction row. The verbalization is your working notes; the tables section is where you record the actual data. DO NOT identify matches in verbalization and then fail to extract them.
- 6. ****Continue to the end**** - Do not stop after finding matches; process every remaining entry
- 7. ****In 'verbalization' field****: Confirm you reached and checked the final entry (e.g., "Completed all 92 entries")
- 8. ****CRITICAL - Understanding Entity Relationships in Lists****: When processing a source entity's list of related entities:
 - Finding target entity Y in source entity X's list means "X relates to Y" - EXTRACT THIS
 - Whether Y appears in the question is IRRELEVANT to whether this is a valid relationship
 - ONLY skip extraction when source equals target (i.e., X's list contains X itself - a self-loop, likely a data error)
 - [OK] Correct: Processing "Entity Alpha"'s related items list, found "Entity Beta" (where Beta is mentioned in the question) \rightarrow Extract "Alpha relates to Beta"
 - [OK] Correct: Processing "Component A"'s dependency list, found "Component B" (from question) \rightarrow Extract the dependency relationship
 - [OK] Correct: Processing "Site 001"'s supplier catalog, found "Supplier Corp" (from question) \rightarrow Extract the supplier relationship
 - [X] Incorrect: Processing "Item X"'s related items list, found "Item X" itself \rightarrow Skip (self-loop)
 - ****WRONG reasoning****: "The found entity is mentioned in my question, therefore this is self-referential" - NO. Self-reference means the source entity points to itself, not that the target appears in the question.

Examples:

- If you see a section with 50 entries (e.g., a list with [1] through [50]), and the question asks about 5 specific entities:
 - [OK] Correct in 'verbalization': "Target entities: [Entity A, Entity B, Entity C, Entity D, Entity E]. Entry [1] 'Author et al., 2020. Some description about topic X' - no match. Entry [2] 'Source, 2021. Entity A: A comprehensive analysis' - MATCH to Entity A. Entry [3] 'Creator, 2019. Another description' - no match. ... Entry [50] 'Final Author, 2023. Final description' - no match. Completed all 50 entries."
 - [OK] Correct in 'tables': Extract rows for all matches found
- If you see an exhibits list with 15 items and question asks about 3 companies:
 - [OK] Correct in 'verbalization': "Target entities: [Acme Corp, Widget Inc, Gadget LLC]. Entry [1] 'Exhibit 1.1: Lease Agreement with Property Owner' - no match. Entry [2] 'Exhibit 2.1: Supply Contract with Acme Corp' - MATCH to Acme Corp. ... Entry [15] 'Exhibit 5.4: Insurance Policy' - no match. Completed all 15 entries."
 - [X] Wrong: "The section lists many entries, but none of the target entities appear explicitly" (This is a SUMMARY without listing entries)
 - [X] Wrong: "Checked entries [1]-[50]. Found matches for Entity A and Entity B." (This is a SUMMARY, not entry-by-entry verbalization)
 - [X] Wrong: Stop after finding 2 matches in first 10 entries
 - [X] Wrong: "List entries [1]-[92]. None match." (Must list each entry explicitly: "[1] ... - no match. [2] ... - no match.")

```
{% endif %}
```

Output guidelines:

- Produce one row per real item visible on the page (e.g., per counsel-argument, per entity-period-metric).
- Strictly follow the schema, do not add any new field/key.
- If the page contains multiple distinct items, output multiple rows.

```
{% if extract_quotes %}
```

- For any field, if the given page does not have any relevant information, set the value and quote to null.

```
{% else %}
```

- For any field with no relevant information, set value to null. Quote extraction is disabled, so leave "quote": null' for every field.

```
{% endif %}
```

- Consider the summary if provided to contextualize the extraction.
- **IMPORTANT**: If the question below contains a "FOCUS:" directive specifying a particular row or item to extract, you MUST extract ONLY that specific row/item. Ignore all other rows/items in the page. This focused extraction is used when processing information-dense pages to ensure accuracy and prevent confusion between similar items.

Extraction and Normalization Rules:

- Page-local only. Use only the current page's content.
- Rationale.
 - Start with rationale, and reason about which parts of the text correspond to the field
 - 1--3 sentences: how you located/parsed it; if from a table, include headers and relevant table context.
 - Note any normalization you performed (e.g., "converted \$1.23B to 1230 millions USD").
- Normalization is mandatory.
 - Convert magnitudes into the field's target scale (e.g., target millions: 1,200,000 \rightarrow \$1.2; 1.23B \rightarrow \$1230).
 - Convert unit (e.g., currency) to the target if specified; else keep as_is.
 - Normalize percent per the field's rule (0_100 vs 0_1) and date per date_format.
- Type safety. Coerce to the declared data_type; on failure, set value = null.

```
{% if extract_quotes %}
```

- Quote (evidence).
 - Provide a list with the largest contiguous exact substring that supports the value (often a single element).
 - Preserve punctuation/case; no ellipses or paraphrase.
 - If truly not present on the page \rightarrow quote = null.
 - Remember if quote is not present, then the value should also be None.

```
{% else %}
```

- Quote (evidence) is disabled. Always emit "quote": null' for every field; do not copy supporting text into the output.

```
{% endif %}
```

- is_explicit.
 - True if the underlying value appears explicitly on the page (including trivial deterministic normalization like unit/scale conversion or parentheses-to-negative).
 - False if it requires arithmetic beyond normalization (e.g., sums, differences, ratios) or approximations/inference.
- Duplicates vs distinct items.
 - Multiple mentions of the same item: choose the most precise/authoritative instance on that page (e.g., structured table over prose).

```

- Distinct items: separate rows.
- Relationship schemas with main entity:
  - If the schema has a primary entity field (e.g., "entity_name", "source_identifier"), populate it consistently in every row.
  - For relationship fields: never leave the primary/source entity field null. Only set secondary/target fields to null when no such relationship is present.
{% if enable_verbalization_instructions %}
- **ACTION REQUIRED for structured list sections**: When you encounter enumerated lists containing entity relationships (such as dependency catalogs, related entity lists, component inventories, reference lists, or bibliographic entries), follow the "Critical: Exhaustive List Processing" instructions above. This includes listing all target entities, processing every entry individually without summarization, and understanding that finding a question entity in another document's list is a valid relationship to extract.{% endif %}

# Example
Suppose schema has metric_value (decimal, unit=USD, scale=None) and the page says: "Revenue was $1.23 billion in 2023."

'''JSON
{
  "verbalization": "Not a references section. Extracting direct field values from page content.",
  "tables": [
    {
      "name": "Financials",
      "rows": [
        {
          "fields": {
            "entity_name": {
              "rationale": "The page header mentions the information hence I can use it to extract the entity name. Taken from page header.",
              "value": "Acme Inc.",
              "quote": ["ACME INC."],
              "is_explicit": true,
              "confidence": "Very High"
            },
            "period_end": {
              "rationale": "The period end is mentioned but it is not in the required format. I need to normalized to YYYY-MM-DD. The period was mentioned in the page title",
              "value": "2023-12-31",
              "quote": ["For the year ended December 31, 2023"],
              "is_explicit": true,
              "confidence": "Very High"
            },
            "metric_name": {
              "rationale": "Table 2 mentions the metrics. The Metric label in the caption of the table talking about balance sheet.",
              "value": "revenue",
              "quote": ["Revenue"],
              "is_explicit": true,
              "confidence": "Very High"
            },
            "metric_forecast": {
              "rationale": "The text mentions the current and past revenue. However, there is no mention of future revenue. Hence I should set the value and quote to None.",
              "value": null,
              "quote": null,
              "is_explicit": false,
              "confidence": "Very Low"
            },
            "metric_value": {
              "rationale": "The revenue is mentioned in in section Commentry. However, it says $1.23 billion. Since the scale is None, I should make it 1230000000. Converted $1.23B to 1230000000 with scale='None', unit='USD'.",
              "value": 1230,
              "quote": ["Revenue was $1.23 billion in 2023"],
              "is_explicit": true,
              "confidence": "Very High"
            }
          }
        }
      ]
    }
  ]
}
'''

# input
The following question has been asked by the user over the whole document set:
<Question Start>
{{question}}
<Question End>

The Relationship Schema for this question has already been defined.
Your task is to populate this schema using only the information available in the document. The extracted data will later be aggregated with information from other pages to generate the final answer.

{% if previous_chunk_summary %}
# Summary of the pages till now
{{previous_chunk_summary}}
{% endif %}

<Document Start>
# The name of the CURRENT DOCUMENT you are extracting from is:
{{document_name}}

```

```

# Document Descriptions
{{document_description}}

# Page Content
{{document}}
<Document End>

# Relationship Schema
{{schema}}

# Field Extraction Guidelines
{% if extraction_guidelines %}
The schema includes specific extraction guidelines for certain fields. Follow these instructions when extracting:

{{extraction_guidelines}}
{% endif %}

# Output Format

Your output should be a JSON object with these fields:
{% if enable_verbalization_instructions %}
**IMPORTANT**: Verbalization is your working notes to show your reasoning. The 'tables' array is where you record your
actual extractions. Any match you identify in verbalization MUST have a corresponding row in tables.

1. **verbalization** (optional string): For sections with enumerated lists (such as dependency lists, related entity
lists, "References", "Bibliography", "Citations", component inventories, or any numbered/bulleted entity listings
), describe your systematic checking process:
- List ALL entities mentioned ANYWHERE in the question---include the primary entity AND all comparison/related
entities (e.g., if the question asks about relationships involving entities X, Y, and Z, list [X, Y, Z])
- For each entry in the list: entry number/identifier, name/title found, match determination (which target entity it
matches, or "no match")
- **DO NOT SUMMARIZE**: You must list each entry individually, even if none match. Saying "checked all entries, none
match" is insufficient.
- Confirm you reached the final entry

**IMPORTANT for inline references (non-list sections):**
- When processing abbreviated inline identifiers (e.g., short codes, numeric references), DO NOT verbalize
speculative mappings to full entity names.
- Only report matches when the entity name itself appears on the page.
- [X] WRONG: "The document mentions identifier X in context Y, which corresponds to 'Full Entity Name Z'"
- [OK] CORRECT: "The document mentions identifier X but does not include the full entity name - no match"

Examples:
- "Target entities: ['Entity A Title', 'Entity B Title', 'Entity C Title']. Reference list section entries [1]-[25].
Entry [1] 'ID-001. Item X description' - no match. Entry [2] 'ID-002. Entity A Title: A comprehensive
analysis' - MATCH to 'Entity A Title'. Entry [3] 'ID-003. Another item description' - no match. ... Entry [25]
'ID-025. Final item' - no match. Completed all 25 entries."
- "Target entities: ['Supplier A', 'Supplier B', 'Supplier C']. Dependencies section entries [1]-[18]. Entry [1] '
Component X from Vendor Alpha' - no match. Entry [2] 'Raw materials from Supplier A' - MATCH to 'Supplier A'.
Entry [3] 'Packaging from Provider Beta' - no match. ... Entry [18] 'Logistics from Carrier Gamma' - no match.
Completed all 18 entries."

2. **tables** (required array): The extracted data following the schema above, with concise rationales per field.

**CRITICAL**: If you identified ANY matches (marked as "MATCH") in your verbalization, you MUST create a
corresponding extraction row in the tables array for EACH match. The verbalization is your working notes---the
tables section is where you record the actual extractions. Do NOT skip creating rows for matches you
identified during verbalization.

{% else %}
1. **tables** (required array): The extracted data following the schema above, with concise rationales per field.
{% endif %}

Fill out the schema above based on the Document content.

```

D.1.3 Data Reconciliation

Primary Key Selection Selects primary key fields which are used to group data rows for further reconciliation.

```

# instruction
You are selecting **primary keys** for database tables that have been automatically extracted from documents.

## Purpose
Your selected primary key will be used for **downstream data consolidation**:
1. **Deduplication**: Rows with the same primary key will be identified as duplicates
2. **Conflict Resolution**: When duplicates exist, we'll merge them by selecting the best value for each field
3. **Table Consolidation**: The final cleaned table will have one row per unique primary key

## Context
- The data was automatically extracted from unstructured documents (PDFs, text, etc.)
- Extraction errors create duplicate rows for the same real-world observation
- Multiple extractions of the same data point may have conflicting values
- Your primary key defines which rows represent the "same thing" and should be merged

## Your Task
Select the **conceptual primary key** that identifies unique observations in the ideal cleaned table. When two rows
have the same primary key, they should represent the same real-world observation and be consolidated into one row

```

```

## Key Principles

### The Key Defines What Gets Consolidated
Your primary key determines which rows will be merged together:
- Rows with identical primary key values  $\rightarrow$  consolidated into ONE row
- Rows with different primary key values  $\rightarrow$  kept as separate rows
- During consolidation, we merge conflicting field values by selecting the most reliable extraction

### Separate Identifiers from Values
Identifiers define WHAT is being observed - these form the primary key:
- Entity identifiers: company, user, product, document
- Temporal identifiers: date, period, timestamp
- Categorical identifiers: metric_type, event_type, category
- Relationship identifiers: source_entity, target_entity

Values are the observed data - these get consolidated during conflict resolution:
- Measurements: amounts, counts, percentages, scores
- Content: text, descriptions, labels, classifications
- Attributes: status, flags, properties

Rule: Primary keys should contain ONLY identifiers, never measured values.

Examples:
- Metrics over time: Key = entity + time_period (metric values get consolidated)
- Events with content: Key = entity + timestamp (content fields get consolidated)
- Entity relationships: Key = source + target (the relationship itself IS the identifier)

### Field Preferences
STRONGLY PREFER business/domain identifier fields as primary keys.

Use 'row_id' or 'document_name' as the primary key in the scenarios where other fields are allowed to have duplicates as in text that represents dataset.

## Analysis Guidelines

### 1. Understand the Data Domain
Ask yourself: What is this table measuring or recording?
- Financial data: Observations about companies at points in time
- Event logs: Actions by users at timestamps
- Relationships: Connections between entities
- Classifications: Labels applied to items

### 2. Identify the Identifier Fields
Which fields define WHAT is being measured (not the measurement itself)?
- Entity identifiers: company_name, user_id, paper_title, product_id
- Temporal identifiers: date, period_end, timestamp, fiscal_year
- Categorical identifiers: metric_name, event_type, category
- Relationship identifiers: source_entity, target_entity

Exclude value fields from the key: amounts, counts, text content, computed metrics, labels

### 3. Build Composite Keys
Most tables need multiple dimensions to identify observations:
- Financial: entity + time (+ metric_name if multiple metrics per row)
- Events: entity + time (+ event_type if multiple events per entity-time)
- Relationships: source + target
- Time series: entity + time + measure_type

### 4. Validate with Consolidation Logic
Ask yourself:
- If two rows have the same key, should they be merged into one row? If yes, good key.
- If two rows have different values but the same key, is one a duplicate/error? If yes, good key.
- Are you including measured values in the key? If yes, reconsider - those should be consolidated, not part of the key.
- Does the key define the "grain" of the consolidated table? If yes, good key.

### 5. Use Statistics as Supporting Evidence
The table statistics (null counts, distinct values) can help validate your choice:
- Key fields should have low null rates
- Key combinations should have reasonable cardinality
- But don't let statistics override semantic meaning

### 6. Expect Duplicates - That's Why We Need the Key
The current extracted data will have duplicates with the correct key - this is by design:
- Multiple extractions of the same observation create duplicate rows
- These duplicates will be consolidated using your primary key
- Text variations (spacing, punctuation) will exist in both keys and values
- Your job: identify the conceptual key that groups rows for consolidation, not ensure perfect uniqueness now

## SQL Query Capabilities
You have access to SQL queries to inspect the data. Use queries to:
- Check uniqueness: 'SELECT field, COUNT(*) as cnt FROM table GROUP BY field HAVING cnt > 1'
- Check null rates: 'SELECT COUNT(*) - COUNT(field) as null_count FROM table'
- Check cardinality: 'SELECT COUNT(DISTINCT field) FROM table'
- Check composite uniqueness: 'SELECT field1, field2, COUNT(*) FROM table GROUP BY field1, field2 HAVING COUNT(*) > 1'
- Sample values: 'SELECT field FROM table LIMIT 10'

Focus queries on business fields first. Only query system fields if you're considering them as a fallback.

## Query Budget
You have a budget of {max_queries} SQL queries. Use them strategically:
- First query: Get overview statistics for all fields

```

```

- Subsequent queries: Drill down on promising primary key candidates
- Final queries: Validate your proposed primary key choice

## Input
**Question:** {{question}}

**Table Name:** {{table_name}}

**Schema:**
{{schema}}

**Table Statistics:**
{{table_stats}}

{% if query_history %}
**Previous Query Results:**
{{query_history}}
{% else %}
**Previous Query Results:**
No queries executed yet
{% endif %}

{% if sql_error_feedback %}
**SQL Error Feedback:**
{{sql_error_feedback}}
{% endif %}

## Output Format
Based on your analysis, provide your decision:

**Action:** Choose either "query" (run another SQL query to gather more information) or "finalize" (you have enough information to recommend a primary key)

**Reasoning:** Explain your thinking (2-3 sentences)
- If action is "query": Why do you need this information? What will it tell you about the consolidation key?
- If action is "finalize": Why is this the best primary key for consolidation? Explain what defines a unique observation and what fields will be consolidated during conflict resolution.

**SQL** (only if action is "query"): The SQL query to run

**Primary Key** (only if action is "finalize"): List of field names that form the primary key, e.g., ["company_name", "period_end_date"] or ["citing_paper_title", "cited_paper_title"]. Use null if no semantic primary key exists.

## Examples

### Example 1: Single Metric Per Row
'''
Action: finalize
Reasoning: This table records a single metric for entities over time. The consolidation key is entity + time_period.
Multiple extractions of the same entity-period will have the same key and be merged into one row. The
metric_value field will be consolidated during conflict resolution (selecting the most reliable extracted value).
Primary key excludes the value field, as values are what get consolidated, not what defines uniqueness.
Primary Key: ["entity_name", "time_period"]
'''

### Example 2: Evidence / Dataset Table
'''
Action: finalize
Reasoning: This table represents a dataset of extracted mentions, not canonical facts. Each row is a distinct evidence
instance that should be preserved even if it refers to the same underlying real-world observation. Consolidating
rows would destroy provenance and reduce the dataset to one representative mention, which is not the goal.
Primary Key: ["row_id"]
'''

### Example 3: Multiple Metrics Per Row
'''
Action: finalize
Reasoning: This table has multiple metric types per entity-period. The metric_type field IS an identifier (which
measurement) that defines uniqueness along with entity and time. During consolidation, rows with the same entity-
period-metric_type will be merged, with the amount field being consolidated via conflict resolution. Without
metric_type in the key, we'd incorrectly merge different metrics (revenue vs. expenses) into one row.
Primary Key: ["entity_name", "time_period", "metric_type"]
'''

### Example 4: Relationship/Association Data
'''
Action: finalize
Reasoning: This table records relationships between entities. The consolidation key is source + target, as each unique
relationship should appear once in the final table. Multiple extractions of the same relationship will be merged.
Any additional fields (relationship_strength, relationship_date, etc.) are attributes of the relationship that
get consolidated during conflict resolution, not part of the identifier.
Primary Key: ["source_entity", "target_entity"]
'''

### Example 5: Event/Transaction Data
'''
Action: finalize
Reasoning: Events are identified by entity + timestamp. Multiple extractions of the same event will have the same key
and be consolidated into one row. Fields like event_description, amount, and status are event attributes that
will be resolved during conflict resolution. If truly multiple distinct events can occur at the exact same

```

```

        timestamp for the same entity, and there's no event_id or event_type to distinguish them, then entity + timestamp
        is still the semantic consolidation key.
Primary Key: ["entity_id", "timestamp"]
'''

Now, provide your decision.

```

Document-Level Canonicalization Canonicalizes fields within each document.

```

# instruction
You are canonicalizing the field **{{field_name}}** in rows from a single document.

## Context
- **Document**: {{document_name}}
- **All rows are from the SAME document** - variations are likely extraction artifacts, not real distinctions
- Use OTHER COLUMNS to determine if different values refer to the same entity

## Key Principle: Check All Columns for Entity Relationships

**Before deciding if two values are different entities, inspect ALL columns.** Other columns may contain information
that reveals two seemingly different values actually refer to the same entity. Look for any column where:
- One value appears as an attribute of another
- The source text (quotes) mentions both forms together
- Context suggests equivalence

**Always start by examining the full row data:**
'''sql
SELECT * FROM {{table_name}} ORDER BY {{field_name}};
'''

If you see that value A appears in another column for rows with value B (or vice versa), they likely refer to the same
entity and should be canonicalized.

## Workflow

1. **First inspection**: Check ALL columns to find entity relationships
2. **Look for cross-references**: Does value X appear in any column of rows where {{field_name}} = Y?
3. **If values appear related**, canonicalize them to one form
4. **Apply normalization** (case, punctuation, whitespace) for remaining variations
5. **Inspect Again to verify**: Check if the table still contains rows to be canonicalized.
6. **Stop** when no more relationships or variations exist

## Skip If
- Free-text / natural language content
- High cardinality with unique values
- Long verbose strings (>50 chars average)

## Canonicalization SQL Format (CRITICAL)

**You must REPLACE the column, not add a new one.** Do NOT use 'SELECT *' for canonicalization.

**CORRECT** - explicitly list columns and keep the same column name:
'''sql
SELECT
  row_id,
  CASE WHEN {{field_name}} = 'ValueA' THEN 'ValueB' ELSE {{field_name}} END AS {{field_name}},
  other_col1,
  other_col2,
  ...
FROM {{table_name}}
ORDER BY row_id;
'''

**WRONG** - DO NOT do this:
'''sql
SELECT *, CASE ... END AS {{field_name}}_canonicalized -- WRONG: adds new column
SELECT *, CASE ... END AS {{field_name}} -- WRONG: duplicates column
'''

Requirements:
- List ALL columns explicitly (no 'SELECT *')
- The CASE expression must be named exactly '{{field_name}}' (same as original)
- End with 'ORDER BY row_id'
- DuckDB syntax (no '>' operators)
- Escape quotes: ''' not '\

## Input

**Table**: {{table_name}} | **Field**: {{field_name}} | **Document**: {{document_name}}

**Schema**:
{{schema}}

**Table Statistics**:
{{table_stats}}

{% if inspection_history %}
**Inspection History**:
{{inspection_history}}
{% endif %}

```

```

**Canonicalization History:** {{canonicalization_history}}

{% if canonicalization_error_feedback %}
**Error:** {{canonicalization_error_feedback}}
{% endif %}

**Inspections remaining:** {{inspections_remaining}}

## Actions

**inspect**: Run SQL to check column relationships
'''
action: "inspect"
sql: "<SQL>"
reasoning: "<what you're checking>"
'''

**canonicalize**: Merge related values (MUST replace column, not add new one)
'''
action: "canonicalize"
sql: "<SELECT with explicit columns, CASE for {{field_name}}, ORDER BY row_id>"
reasoning: "<which values are the same entity>"
'''

**stop**: After canonicalization has been performed and you have done inspection again: No more variations found
'''
action: "stop"
reasoning: "<confirm all values are now consistent>"
'''

**skip**: Field is not categorical
'''
action: "skip"
reasoning: "<why>"
'''

```

Error Detection Agent Agent that inspects data and decides which reconciliation operation to perform (deduplicate, aggregate, resolve conflicts, canonicalize).

```

# instruction
You are a data quality controller for a single primary key group during table reconciliation.

Your role is to analyze a subset of rows sharing the same primary key value and decide which reconciliation operation to perform next.

You are processing rows for **one specific primary key value only**. This is a focused reconciliation task for a single entity.

The table contains values extracted from different pages of documents. Each page was processed independently by extraction workers.

## Pipeline Context

**Preprocessing completed**:
1. [OK] **Primary Key Selected**: A semantic primary key has been identified based on the schema and data domain
2. [OK] **Primary Key Fields Canonicalized**: Values in the primary key fields have been standardized (case normalization, whitespace trimming, format consistency)
3. [OK] **Table Split by Primary Key**: Rows have been grouped by primary key values

**Primary Key for this table**: {{primary_key}}

**Your specific primary key value**: {{pk_value}}

You are seeing ONLY the rows with this specific primary key value (typically 2-5 rows, occasionally more).

## Your Position in the Loop

You are in iteration {{iteration}} of {{max_iterations}} for this primary key group.

**Operations performed so far**: {{operations_history}}

After each operation you chose, an executor agent:
- Inspected the rows
- Generated and executed SQL to perform the operation
- Returned updated rows (could be 0, 1, or all rows depending on the operation)

You now see the **current state** of rows for this primary key after those operations.

## Your Goal

**Primary objective**: Reduce this primary key group to **exactly 1 row** through reconciliation operations.

Multiple rows with the same primary key indicate extraction redundancy, complementary information, or conflicts that need resolution. The ideal end state is a single consolidated row containing all valid information for this entity.

**When to stop before reaching 1 row**:

```

- Only if the executor has attempted an operation and determined, after deep analysis of the data, that the remaining rows represent genuinely distinct observations that should NOT be merged
- The executor's refusal must be based on semantic evidence (e.g., different temporal contexts, distinct aspects that shouldn't be combined)
- Simply noticing that rows "look different" is NOT sufficient reason to stop - differences are exactly what reconciliation operations are designed to handle

****When executor does not reduce to 1 row, diagnose the reason**:**

1. ****Legitimate refusal****: The executor reasoned that rows represent distinct entities/events that should not be merged (e.g., multiple separate occurrences, different aspects that shouldn't be combined) \rightarrow route to ****stop****
2. ****Wrong operation chosen****: The executor tried to apply the operation but the current state actually requires a different operation (e.g., routed to consolidate but there are conflicts to resolve, or routed to deduplicate but there's complementary information to aggregate) \rightarrow route to the ****appropriate operation**** that addresses the actual data pattern

Your Decision Process

You iterate: ****inspect**** \rightarrow ****inspect**** \rightarrow ... \rightarrow ****route****

At each call, you decide:

1. ****inspect****: Run a SQL query to understand the current state better
2. ****route****: Route to a specific reconciliation operation (or stop)

Available Route Actions

****Important****: When deciding which operation to route to, focus on ****main data columns**** (measured values like names, dates, amounts, descriptions). Use metadata columns ('_quote', '_rationale', '_confidence', '_is_explicit') to understand the context and semantics of the main data. However, do NOT route to an operation if the issue exists ONLY in metadata columns while the main data columns are already clean. Route based on problems in main data columns; the executor will then handle corresponding metadata columns appropriately.

1. ****deduplicate****
 - Trigger when: Multiple rows with redundant information in non-primary-key fields
 - Purpose: Remove redundant rows, keeping the best representative
 - Executor will select the best row based on confidence, completeness, and data quality
 - Note: If a key column has non-redundant information (different values, NULLs to fill, or conflicts), route to aggregate, consolidate, or resolve_conflicts instead
2. ****aggregate****
 - Trigger when: The field semantics CLEARLY indicate multiple values should coexist (e.g., list of all products, sum across categories)
 - Use sparingly: Only when you're confident the field is meant to hold multiple values, not when you're trying to avoid making a choice
 - Purpose: Combine multiple valid values into a unified representation (arrays, lists, sums, ranges)
 - Executor will apply: SUM (additive quantities across categories), ARRAY_AGG (multiple items in a collection), MIN/MAX (range bounds), COALESCE (filling different fields)
 - Examples: Multiple phone numbers \rightarrow array, revenue across product lines \rightarrow sum, multiple categories \rightarrow array
3. ****consolidate****
 - Trigger when: Information is scattered across rows with incomplete data that can be filled by copying values from other rows
 - Purpose: Broadcast complete information to fill gaps where data is missing
 - Executor will propagate consistent values across rows without combining or transforming them
 - Note: Returns multiple rows; use when information should be copied, not merged
4. ****resolve_conflicts****
 - Trigger when: Different values exist for a field, and only ONE is correct or most appropriate
 - Default choice: When uncertain whether to aggregate or resolve, choose resolve_conflicts
 - Purpose: Select the single best value based on evidence, source quality, semantic fit, and specificity
 - Executor will evaluate which value to keep and discard the others
 - Examples: Different dates for same event, different names, different statuses, different measurements, conflicting attributes
5. ****canonicalize****
 - Trigger when: Non-primary-key fields have different formats or representations
 - Example: "5M" vs "5000000", "USA" vs "United States"
 - Purpose: Standardize non-primary-key column values to a canonical form
 - Note: Primary key fields are already canonicalized. NULL values should generally be preserved.
6. ****stop****
 - Trigger when:
 - * ****Ideal case****: Successfully reduced to 1 row
 - * ****Acceptable case****: After diagnosing the most recent executor result, you determine the executor legitimately refused to merge because rows represent genuinely distinct observations (not because the wrong operation was chosen)
 - Before stopping with >1 row remaining: Analyze the executor's reasoning from the last operation. Did it refuse because rows are truly distinct, or because the operation chosen wasn't suitable for the actual data pattern?
 - If the operation was wrong (e.g., tried to deduplicate when should aggregate, or tried to consolidate when should resolve conflicts), route to the correct operation instead
 - Do NOT stop just because: Rows look different, have some NULL values, or seem "mostly done" - these are exactly what reconciliation addresses
 - Purpose: End the reconciliation loop for this primary key group

Current State

Question to answer: {{question}}

Table name: {{table_name}}

Primary key value: {{pk_value}}

```

Current row count: {{current_row_count}}

## Schema
{{schema}}

## Table Statistics (for your specific PK group)
{{table_stats}}

## Your Inspection History (current controller call)
{{inspection_history}}

## Inspection Budget
{% if must_route_now %}
[WARNING] WARNING: No more inspections allowed. You MUST route to an action now (action: "route").
{% else %}
You can inspect {{inspections_remaining}} more time(s) before you must route.
{% endif %}

# Task

Decide: should you inspect more, or are you ready to route to an action?

**If you need more information**, output:
- action: "inspect"
- sql: Your inspection SQL query (e.g., check row differences, compare confidence scores)
- reasoning: Why you need this inspection

**Example inspections:**
```sql
-- View all rows for this PK
SELECT * FROM {{table_name}};

-- Compare measured values across rows
SELECT user_id, message_text, date, _confidence FROM {{table_name}};

-- Check metadata
SELECT _quote, _rationale, _is_explicit FROM {{table_name}};
```

**If you're ready to route**, output:
- action: "route"
- route_to: One of [deduplicate, aggregate, consolidate, resolve_conflicts, canonicalize, stop]
- reasoning: Why you're routing to this action based on the current state

Think about:
- Target: Are we at 1 row yet? If not, what operation will move us closer?
- Are rows redundant (same information), complementary (different columns populated), or conflicting (same column with different values)?
- For conflicting values: Default to resolve_conflicts to pick the best value. Only use aggregate if the field clearly needs to hold multiple values
- Are there format inconsistencies to canonicalize?
- Can consolidate or deduplicate reduce row count? Can resolve_conflicts eliminate value conflicts?
- If last operation didn't reduce to 1 row: Was it a legitimate refusal (rows are truly distinct) or was the wrong operation applied? If wrong operation, route to the correct one.
- Only stop early if: Executor has legitimately refused to merge after analysis because rows represent distinct entities, NOT because the chosen operation wasn't suitable for the data pattern

Important:
- You're only seeing rows for ONE primary key value
- After you route, an executor will perform that ONE operation
- You'll be called again to decide the next operation (up to {{max_iterations}} total iterations)
- Each operation should make meaningful progress toward a clean, consolidated result

```

Reconciliation agent Agent that generates and executes SQL to perform the reconciliation operation chosen by the controller.

```

# instruction
You are an executor agent for reconciling rows that share the same primary key values.

You are processing a small subset of the full table - specifically, all rows that share a specific primary key value.

{% if objective == "consolidate" %}Your task is to fill NULL values by broadcasting non-NULL values across these rows, returning multiple rows with gaps filled.{% elif objective == "aggregate" %}Your task is to aggregate values across these rows where appropriate, returning multiple rows with aggregated values broadcasted.{% elif objective == "deduplicate" %}Your task is to identify and remove redundant rows, returning 1 or more rows depending on redundancy patterns.{% else %}Your task is to consolidate these rows into 1 output row that best represents this entity.{% endif %}

The table contains values extracted from a document based on the provided schema. Each row was extracted independently by workers processing different document pages, which may have had partial context.

## Pipeline Context

Preprocessing completed:
1. [OK] Primary Key Selected: A semantic primary key has been identified
2. [OK] Primary Key Fields Canonicalized: Primary key values have been standardized

Primary Key for this table: {{primary_key}}

```

```

**Your specific primary key value**: {{pk_value}}

You are only seeing rows with this specific primary key value. Typically 2-5 rows, but occasionally more.

## Your Task

You have been assigned the objective: **{{objective}}**

You iterate: inspect $ \rightarrow$ inspect $ \rightarrow$ ... $ \rightarrow$ generate_merge_sql

{% if objective == "consolidate" %}Your merge SQL should return **multiple rows** (all input rows with NULLs filled where appropriate).{% elif objective == "aggregate" %}Your merge SQL should return **multiple rows** (all input rows with aggregated values broadcasted where appropriate).{% elif objective == "deduplicate" %}Your merge SQL should return **1 or more rows** (deduplicated set where redundant rows are reduced to representatives, distinct rows are preserved).{% else %}Your merge SQL should return **1 row** that consolidates all the rows in your subset.{% endif %} However:
- If you determine that the rows in your group are NOT actually duplicates/conflicts but represent distinct entities or events, return **all original rows** by using 'SELECT * FROM {{table_name}}' to preserve them.

**Note on metadata fields**: Columns like _rationale, _quote are metadata about the extraction process, not measured values.

{% if objective == "deduplicate" %}
## Objective: Deduplicate

**Goal**: Identify which rows contain redundant information in main data columns and deduplicate them, while preserving rows that contain distinct information. Return the deduplicated set.

### Phase 1: Inspect and Identify Redundancy Groups

**Inspection strategy**:
- Always examine all rows first: 'SELECT * FROM {{table_name}}'
- Read the _quote and _rationale fields to understand what each row represents
- Compare ALL non-primary-key measured fields (date, amount, description, etc.)
- Consider the semantic context: Is it possible for this entity type to have multiple instances?

**Critical Decision: Can this event/entity occur multiple times in reality?*

**Step 1: Identify redundancy patterns**

Deduplication applies when rows have the same values across ALL (or nearly all) main data columns. You need to identify which specific rows share redundant information:

1. **All rows redundant**: All rows have the same values across main data columns
  - Example: 5 rows all describing the same entity with the same attributes (Company X, founded 2010, headquarters Chicago, CEO John Smith) extracted from 5 different pages
  - All 5 rows form one redundancy group because they contain the same complete set of facts
  - **Action needed**: Select 1 best row from this group

2. **Partial redundancy**: Some rows share the same complete set of main data values (forming redundant subsets), while other rows differ in at least one main data column
  - Example: 5 rows describing purchases where 3 have identical values (Item A, $50, 2024-01-10, Store X) and 2 have different values (Item B, $30, 2024-01-11, Store Y) and (Item A, $50, 2024-01-12, Store X)
  - The 3 identical rows form one redundancy group
  - The other 2 rows are distinct (each differs in date or item from the redundancy group)
  - **Action needed**: Select 1 best row from the redundancy group, keep the 2 distinct rows unchanged

3. **No redundancy**: Each row has different values in at least one main data column
  - Example: 3 rows describing different product variants (Color: Red, Size: M), (Color: Blue, Size: L), (Color: Red, Size: L)
  - No redundancy groups exist because each row has a unique combination of values
  - **Action needed**: Return all rows unchanged using 'SELECT * FROM {{table_name}}'

**Why grouping matters**: Redundant rows (those with the same complete set of main data values) should be reduced to one representative per group, but distinct rows (those with at least one differing main data value) must be preserved. Your SQL needs to handle both simultaneously.

**Step 2: For each redundancy group identified**, determine the semantic nature:

**Important note on distinguishing metadata**: The absence of differing metadata (timestamps, quotes, rationale) between duplicate rows does NOT automatically indicate extraction redundancy. It may simply mean the extraction process was coarse-grained and didn't capture fine-grained distinguishing details. Focus on the semantic nature of what the data represents.

**Case A: Multiple real-world occurrences are possible**
- The entity/relationship represents actions, events, or transactions that can legitimately happen multiple times
- Characteristics of multiple occurrences:
  - Behavioral or transactional data (purchases, interactions, activities)
  - Events occurring across time that can repeat
  - Primary key includes temporal dimensions without fine-grained precision (date without timestamp)
  - Data describes "what happened" rather than "what is"
- These rows represent DISTINCT REAL-WORLD EVENTS that share the same entity identifier
- **Action**: Keep 1 best row, set 'number_instances = <total_row_count>'
- **Interpretation**: This event/transaction occurred N times in the real world

**Case B: Only one instance exists in reality**
- The fact/attribute is an intrinsic property or one-time event that cannot change or repeat
- Characteristics of singular facts:
  - Definitional attributes (birth date, founding year, inherent properties)
  - One-time historical events that cannot recur
  - Data describes "what is" rather than "what happened"
  - Primary key is purely entity-based without temporal dimensions

```

```

- These rows are EXTRACTION REDUNDANCY: The same singular fact captured multiple times from different sources
- **Action**: Keep 1 best row, set 'number_instances = 1'
- **Interpretation**: This fact exists once but was extracted N times

### Phase 2: Generate Merge SQL

After identifying redundancy groups and their semantic nature, generate SQL to:
1. From each redundancy group, select the best representative row (most complete, best quote)
2. Set 'number_instances' appropriately for each representative
3. Include all rows that are not part of any redundancy group (distinct rows)
4. Return the deduplicated set

**Example 1: All rows redundant** (5 rows all contain same artist birth date extracted from different pages)

Inspection reveals: All 5 rows have identical main data (artist_name, date_of_birth, etc.), only differ in metadata
Decision: Extraction redundancy (birth date is a singular fact)
Action: Return 1 row with number_instances = 1


sql
-- After inspection, chose row_id = 102 as best (highest confidence, clearest quote)
SELECT
  * EXCLUDE(number_instances),
  1 AS number_instances -- Singular fact extracted 5 times
FROM {{table_name}}
WHERE row_id = 102



**Example 2: Subset redundancy** (5 rows: 3 contain redundant alternate name "Name A", 2 contain distinct alternate names "Name B" and "Name C")

Inspection reveals:
- Rows 201, 202, 203: all have alternate_name = "Name A"  $\rightarrow$  redundancy group 1
- Row 204: has alternate_name = "Name B"  $\rightarrow$  distinct
- Row 205: has alternate_name = "Name C"  $\rightarrow$  distinct

Decision:
- "Name A" appears 3 times (extraction redundancy, same name extracted from 3 pages)
- "Name B" and "Name C" are distinct values, should be preserved

Action: Keep 1 row from the "Name A" group, plus the 2 distinct rows  $\rightarrow$  return 3 rows total


sql
-- After inspection, chose row_id = 201 from the "Name A" group as best representative
SELECT
  * EXCLUDE(number_instances),
  1 AS number_instances -- "Name A" is singular, extracted 3 times
FROM {{table_name}}
WHERE row_id = 201 -- Best of the "Name A" redundancy group

UNION ALL

-- Preserve distinct rows unchanged
SELECT * FROM {{table_name}}
WHERE row_id IN (204, 205) -- "Name B" and "Name C" are distinct



**Alternative pattern for partial deduplication**:

sql
-- Use CASE to mark which rows to keep, then filter
WITH ranked AS (
  SELECT *,
    CASE
      -- Redundancy group: rows with similar main data
      WHEN main_col_a = 'duplicate_value' AND main_col_b IS NULL
      THEN ROW_NUMBER() OVER (PARTITION BY main_col_a, main_col_b ORDER BY _confidence DESC, row_id)
      -- All other rows are distinct, keep them
      ELSE 1
    END AS keep_rank
  FROM {{table_name}}
)
SELECT
  * EXCLUDE(keep_rank, number_instances),
  CASE
    WHEN <condition for redundancy group> THEN <1 or count>
    ELSE number_instances -- Preserve for distinct rows
  END AS number_instances
FROM ranked
WHERE keep_rank = 1



**Key points**:
- 'number_instances' value depends on your semantic analysis
- Return 1 or more rows depending on redundancy structure
- Each returned row should have appropriate 'number_instances' value
- Preserve rows with distinct main data values
- Only deduplicate rows that are truly redundant in main data columns
- Use inspection results to justify your choice
- Document your reasoning clearly
{% endif %}

{% if objective == "aggregate" %}
## Objective: Aggregate

```

```

**Goal**: For columns containing multiple distinct values that should be combined, compute an aggregated representation
(sum, collection, concatenation) and broadcast it to all rows.

**Inspection strategy**:
- Examine all rows: 'SELECT * FROM {{table_name}}'
- For each column, identify if it has multiple distinct non-NULL values
- Read _quote and _rationale to verify these values are complementary and should be combined
- Determine the appropriate aggregation method for each column

**Decision logic for each column**:
For columns with multiple distinct non-NULL values:
- **Complementary values that should combine**  $\rightarrow$  apply aggregation across rows and broadcast result to all
rows
- Multiple valid instances of a multi-valued attribute: ARRAY_AGG, STRING_AGG
- Numeric quantities to sum: SUM
- Range boundaries: MIN/MAX pairs
- **Conflicting values** (mutually exclusive, only one correct)  $\rightarrow$  leave unchanged, will be handled by
resolve_conflicts
- **Single value or all NULLs**  $\rightarrow$  leave unchanged

**Key principle**: Aggregation combines semantically compatible values that should coexist. Return all rows with
aggregated values broadcasted where appropriate.

**Merge SQL strategy**:
1. **First inspect**: Determine which columns have multiple values that should be combined
2. **Then generate SQL**: Use window functions to compute aggregated values and broadcast to all rows
3. Columns that don't need aggregation remain unchanged
4. Return ALL rows with aggregations applied where needed

**SQL Pattern using window functions**:
```sql
-- Apply aggregation to specific columns that need combining, broadcast result to all rows
-- Use * EXCLUDE to avoid listing all columns
SELECT
 * EXCLUDE(col_a, col_a_quote, col_b),
 -- Aggregate multi-valued column across all rows in this PK group
 STRING_AGG(DISTINCT col_a, ' | ') OVER (PARTITION BY pk_col1, pk_col2) AS col_a,
 STRING_AGG(DISTINCT col_a_quote, ' | ') OVER (PARTITION BY pk_col1, pk_col2) AS col_a_quote,
 -- Sum numeric values across rows
 SUM(col_b) OVER (PARTITION BY pk_col1, pk_col2) AS col_b
FROM {{table_name}}
WHERE pk_col1 = '...' AND pk_col2 = '...'
```

**Alternative pattern using subqueries**:
```sql
-- Compute aggregations in subqueries, join back to preserve all rows
-- Use * EXCLUDE for cleaner SQL
SELECT
 t.* EXCLUDE(col_a, col_a_quote),
 -- Replace col_a with aggregated value across all rows
 (SELECT STRING_AGG(DISTINCT col_a, ' | ') FROM {{table_name}} WHERE pk_col1 = t.pk_col1 AND pk_col2 = t.pk_col2) AS
 col_a,
 (SELECT STRING_AGG(DISTINCT col_a_quote, ' | ') FROM {{table_name}} WHERE pk_col1 = t.pk_col1 AND pk_col2 = t.pk_col2
) AS col_a_quote
FROM {{table_name}} t
WHERE t.pk_col1 = '...' AND t.pk_col2 = '...'
```

**Key points**:
- Return ALL input rows with the same schema (no columns added or dropped)
- Only aggregate columns where combining values makes semantic sense
- Broadcast aggregated values to all rows in the group
- Columns with conflicts or single values remain untouched for later iterations
{% endif %}

{% if objective == "consolidate" %}
## Objective: Consolidate

**Goal**: Fill NULL values in non-primary-key columns by broadcasting non-NULL values from other rows in the same
column, where semantically appropriate.

**Inspection strategy**:
- Examine all rows: 'SELECT * FROM {{table_name}}'
- For each column, identify if it has NULLs in some rows and non-NULL values in others
- Read _quote and _rationale to determine if broadcasting makes semantic sense
- Check if non-NULL values in the same column are consistent (not conflicting)

**Decision logic for each column**:
For each column with mixed NULL/non-NULL values:
- **If there is exactly ONE distinct non-NULL value**  $\rightarrow$  broadcast it to fill NULLs in that column
- **If there are MULTIPLE distinct non-NULL values**  $\rightarrow$  do NOT consolidate this column; leave it unchanged
for other operations (aggregate or resolve_conflicts) to handle
- If semantic context suggests NULLs should remain (e.g., truly missing data)  $\rightarrow$  preserve NULLs

**Key principle**: Consolidation is a focused operation that only fills gaps where information is unambiguous. When a
column has multiple distinct non-NULL values, those values need different handling (combining via aggregation or
selecting via conflict resolution) which will be addressed in subsequent iterations.

**Merge SQL strategy**:
1. **First inspect**: Examine all rows to understand which row has the best value for each column

```

2. ****Then generate SQL****: For each consolidatable column, choose the specific row_id with the most reliable value

3. Use subqueries or window functions to broadcast from those chosen rows

4. Return ALL input rows with NULLs filled ONLY in columns that meet consolidation criteria

5. For columns with multiple distinct non-NULL values, preserve them exactly as-is

****Pattern 1: Selective broadcasting with subqueries****

```

' 'sql
-- After inspecting, substitute actual row_ids you identified as having the best values
-- Use * EXCLUDE for cleaner SQL that avoids listing all columns
SELECT
  t.* EXCLUDE(col_a, col_a_quote, col_b, col_b_quote),
  -- Broadcast col_a from the row you determined has the best value (e.g., row 117)
  COALESCE(t.col_a, (SELECT col_a FROM {{table_name}} WHERE row_id = <row_with_best_col_a>)) AS col_a,
  COALESCE(t.col_a_quote, (SELECT col_a_quote FROM {{table_name}} WHERE row_id = <row_with_best_col_a>)) AS
    col_a_quote,
  -- Broadcast col_b from a different row you chose (e.g., row 118)
  COALESCE(t.col_b, (SELECT col_b FROM {{table_name}} WHERE row_id = <row_with_best_col_b>)) AS col_b,
  COALESCE(t.col_b_quote, (SELECT col_b_quote FROM {{table_name}} WHERE row_id = <row_with_best_col_b>)) AS
    col_b_quote
FROM {{table_name}} t
' '

```

****Pattern 2: Using window functions with specific row selection****

```

' 'sql
-- Use * EXCLUDE for simpler SQL
SELECT
  * EXCLUDE(col_a, col_b),
  -- Consolidate col_a by broadcasting from the row with best metadata
  COALESCE(col_a, MAX(CASE WHEN row_id = <chosen_row_id> THEN col_a END) OVER (PARTITION BY pk_col1, pk_col2)) AS
    col_a,
  -- Consolidate col_b from a different source row
  COALESCE(col_b, MAX(CASE WHEN row_id = <other_row_id> THEN col_b END) OVER (PARTITION BY pk_col1, pk_col2)) AS
    col_b
FROM {{table_name}}
' '

```

****Key approach****: After inspecting the rows, determine for each consolidatable column which specific row has the most reliable non-NULL value (based on quote quality, rationale clarity, or completeness), then broadcast that specific value.

- Keep metadata columns (_quote, _rationale) from original rows unchanged

****Key points****:

- Return multiple rows (all input rows with selective NULLs filled)
- Only modify columns where broadcasting a single value is appropriate
- Leave columns with multiple distinct non-NULL values untouched for subsequent operations
- The reconciliation loop will handle remaining issues in later iterations

```
{% endif %}
```

{% if objective == "resolve_conflicts" %}

Objective: Resolve Conflicts

****Goal****: When rows have conflicting measured values, determine if it's a true conflict to resolve or separate legitimate occurrences.

****Inspection strategy****:

- Examine all rows: 'SELECT * FROM {{table_name}}'
- Identify which non-primary-key fields have conflicting (different distinct) values
- Read _quote and _rationale fields to understand what each row represents

****Important decision****:

- ****If conflicting field is a temporal/event field**** (date, timestamp) and both values are equally valid \rightarrow these represent separate events, keep all rows: 'SELECT * FROM {{table_name}}'
- ****If conflict is due to measurement error**** (different amounts, descriptions for same event) \rightarrow resolve by selecting most reliable value

****Merge SQL strategy for true conflicts****:

- Use metadata to decide: Compare _quote and _rationale for supporting evidence
- ****Decision process****: For each competing row, explicitly evaluate it against each criterion below. Document your assessment for each criterion before making the final selection.

- Selection criteria:

- **Field definition alignment****: Prefer values that match the semantic scope of the field
 - Choose values that directly answer what the field asks for, not broader or narrower variants
 - Match the specificity: if the field has modifiers or constraints, select values that align with those constraints
 - When the field name corresponds to a standard reporting item, the value extracted from where that item is formally reported should be strongly preferred over values described or explained elsewhere, even if explanatory sections provide additional context
- **Extraction location relevance****: Prefer values extracted from locations that directly serve the field's purpose
 - Consider which document sections or structures are authoritative for this type of information
 - Prioritize locations where this field would naturally appear in its primary context
 - When available, examine text_header or similar metadata to understand extraction context
- **Source authority****:
 - Direct measurements or final values over constituent parts or supplementary details
 - Structured presentations (tables, statements, forms, labeled fields) over descriptive text (discussions, notes, commentary)
 - Primary data sections over explanatory sections
 - Explicit field labels over derived mentions
- **Quote quality****: Prefer direct quotes over inferred values
- **Rationale clarity****: Prefer clear, specific rationale

- ****Preferred approach****: After inspection, identify the specific row_id you want to keep and filter directly: 'SELECT * FROM {{table_name}} WHERE row_id = <chosen_row_id>'

```

- Alternative: Use ROW_NUMBER() with ORDER BY, but be careful with tiebreakers
{% endif %}

# Input

{% if verification_mode %}
**Phase**: Verification

Primary key value you were processing: {{pk_value}}

## Initial Table Schema (before your SQL)
{{initial_schema}}

## Final Table Schema (after your SQL)
{{final_schema}}

### Your Generated SQL
```sql
{{generated_sql}}
```

### Your Reasoning When Generating
{{generation_reasoning}}

### Available Tables for Inspection
Two tables are registered in DuckDB:
- **initial_table**: {{initial_row_count}} rows BEFORE your SQL was applied
- **final_table**: {{final_row_count}} rows AFTER your SQL was applied

**IMPORTANT**: You do NOT have the actual table contents yet. You must inspect both tables using SQL to verify the
transformation was executed correctly.

### Your Verification Inspection History
{{verification_inspection_history}}

{% else %}
Question to answer: {{question}}

Table name: {{table_name}}

Primary key value you're processing: {{pk_value}}

## Schema
{{schema}}

## Table Statistics (for your subset)
{{table_stats}}

## Controller's Reasoning
The controller routed to this objective with the following reasoning:
{{controller_reasoning}}

## Your Inspection History
{{inspection_history}}

## Inspection Errors (if any)
{{inspection_error_feedback}}

## SQL Generation Errors (if any - for retry)
{{sql_error_feedback}}

## Status

{% if inspections_remaining > 0 %}
**Phase**: Inspection (optional)
**Budget**: You can inspect {{inspections_remaining}} more time(s), or choose to generate merge SQL now.

After inspection phase ends, you'll enter the SQL generation phase with 3 attempts to generate valid SQL.
{% else %}
**Phase**: SQL Generation
[WARNING] You are now in the SQL generation phase. You can no longer inspect.
Generate the merge SQL. If it fails, you'll get error feedback and up to 2 more attempts (3 total).
{% endif %}
{% endif %}

# Task

{% if verification_mode %}
Your merge SQL has been executed. Verify that the SQL correctly transformed initial_table  $\rightarrow$  final_table as
intended by your reasoning.

**VERIFICATION SCOPE**:
You are checking if your SQL performed the transformation correctly, NOT whether the reconciliation approach itself was
correct. Focus on: Did the SQL execute the logic described in your reasoning?

**VERIFICATION WORKFLOW**:

1. **First, you MUST inspect** (especially if you haven't inspected yet):
- action: "inspect"
- sql: Query to compare initial_table and final_table (e.g., SELECT * from both, or JOIN them)
- reasoning: What transformation aspects you're verifying (row count changes, column transformations, etc.)

```

```

2. **After inspecting and verbalizing what you observed, then decide:**

**Option A: Approve** - The SQL correctly executed the intended transformation
- action: "approve"
- reasoning: Confirm the transformation matches your generation reasoning:
  * Does initial_table $\rightarrow$ final_table match the expected transformation?
  * Are row counts, aggregations, and column values as described in your reasoning?

**Option B: Regenerate** - The SQL did not execute the intended transformation correctly
- action: "regenerate"
- sql: Corrected SQL that properly implements your original reasoning
- reasoning: What transformation error did you observe? How does the new SQL fix it?

**Remaining Inspections**: {{remaining_inspections}}

**Critical Rules**:
- Inspect the actual data before deciding
- Compare initial_table vs final_table to verify the transformation
- Approve if the SQL correctly implemented your reasoning
- Regenerate if the SQL failed to execute the intended transformation
- Regenerated SQL is auto-accepted (no second verification)

{% else %}
Decide: should you inspect more, or are you ready to generate the merge SQL?

If you need more information, output:
- action: "inspect"
- sql: Your inspection SQL query (SELECT statement to understand your subset)
- reasoning: Why you need this inspection

If you're ready to generate the merge, output:
- action: "generate_merge_sql"
- sql: Your merge SQL query that {% if objective == "consolidate" %}returns **multiple rows** with NULLs filled{% elif
objective == "aggregate" %}returns **multiple rows** with aggregated values broadcasted{% elif objective == "
deduplicate" %}returns **1 or more rows** with redundancy removed{% else %}consolidates all rows in your subset
to **exactly 1 row**{% endif %}
- reasoning: How your SQL accomplishes the objective

**Important**:
- **Merge SQL format**: Must be a SELECT statement{% if objective not in ["consolidate", "aggregate", "deduplicate"] %}
that returns exactly 1 row{% endif %} (no CREATE TABLE, INSERT, or DELETE)
- **Schema preservation**: Include ALL original columns (row_id, page_number, *_quote, *_rationale)
- **number_instances column**: {% if objective == "deduplicate" %}Set this to the total row count (if true duplicates)
or 1 (if semantically distinct instances) for each returned row{% elif objective in ["consolidate", "aggregate"]
%}Preserve the existing value for each row{% else %}This column is not visible in your table; it will be
preserved automatically{% endif %}
- **Quote/rationale columns**: When merging, update these columns accordingly (concatenate sources or keep the best)
{% if objective in ["consolidate", "aggregate"] %}- **Multiple row output**: Your SQL should return all input rows with
transformations applied where appropriate{% elif objective == "deduplicate" %}- **Flexible row output**: Return
1 or more rows depending on redundancy structure (1 representative per redundancy group + all distinct rows){%
else %}- **Single row output**: Your SQL MUST return exactly 1 row. Use LIMIT 1, ROW_NUMBER() = 1, or GROUP BY as
appropriate{% endif %}
- **Result limits**: Inspection SQL should use LIMIT to avoid large results
{% endif %}

```

D.1.4 Answer Generation

SQL Query Generator Iteratively generates SQL queries to inspect the reconciled data and gather information for answering.

```

# instruction
You are a SQL query generator for answering questions about data in DuckDB tables.
{% if citation_mode %}

You have completed the query generation phase. Your queries have been used by the final answer generator to produce the
verbalized answer.

Now you need to generate a single SQL query that selects the rows used to answer the question, including all provenance
columns (quote, rationale, text_headers). This will enable proper citation of the source data.
{% else %}
You cannot see full tables - only schema and statistics. Use SQL queries to retrieve information needed to answer the
question.
{% endif %}

{% if not citation_mode %}
## Strategy
{% if strategy == "full_table" %}
Table is small (<=100 rows). Start by sampling the full table with 'SELECT * FROM table LIMIT {{row_limit}}' to see all
data, then use focused queries if needed.
{% else %}
Table is large (>100 rows). Use focused SQL queries with filters and aggregations. Results are limited to {{row_limit}}
rows. Consider sampling first if you need to understand the structure.
{% endif %}

## Guidelines
- You have {{queries_remaining}} quer{{ "y" if queries_remaining == 1 else "ies" }} remaining
{% if is_first_query and strategy != "sql_focused" %}
- **First query**: Sample the table ('SELECT * FROM table LIMIT {{row_limit}}') to see actual data
{% endif %}

```

```

- Use focused queries: filters, aggregations, ordering
- Verify results through multiple approaches (sample, then aggregate, or cross-check with different filters)
- **[Important] Empty results**: If a query returns no rows, verify this with alternative queries before drawing
  conclusions--empty results often indicate SQL issues rather than missing data
- Check data types and sample values before filtering
- Each query result is truncated to {{row_limit}} rows
- For calculations, use SQL functions like ROUND() to ensure precision
- Default ordering: use ORDER BY row_id ASC to preserve order in the document unless the question explicitly requests a
  different sort (e.g., by date).
- Only the values might be wrong (schema/columns are assumed correct). Use rationale/evidence columns to vet rows;
  prefer rows with clear rationales that match the question and drop rows with weak or conflicting rationale. Feel
  free to write cleaning SQL (filters, trims, normalizations) before using the data.

# Working with dates
- If a column is DATE/TIMESTAMP, do not use LIKE, SUBSTR, LEFT, RIGHT, ILIKE, or regex on it.
- Use date-native predicates only: EXTRACT, DATE_TRUNC, strftime, or range filters with DATE 'YYYY-MM-DD'.

{% if sql_error_feedback != "No previous SQL errors" %}
## Previous SQL Errors
{{sql_error_feedback}}
Fix the SQL and retry.
{% endif %}
{% endif %}

# input

## Question
{{question}}

## Schema
{{schema}}

{% if not citation_mode %}
## Table Statistics
{{table_stats}}

## Previous Queries
{{query_history}}

## Status
{% if queries_remaining > 0 %}
{{queries_remaining}} quer{{ "y" if queries_remaining == 1 else "ies" }} remaining. Either:
- 'action: "query"' with SQL to retrieve more information
- 'action: "finalize"' if you have sufficient information
{% else %}
No queries remaining. Must finalize now ('action: "finalize"').
{% endif %}
{% endif %}

{% if citation_mode %}
# Citation SQL Generation

## Previous Investigation History
{{query_history}}

## Finalization Reasoning
{{finalization_reasoning}}

## Final Answer Generated
{{final_answer}}

## Your Citation SQL Attempts
{{citation_attempts_history}}

## Task
Generate a SQL query that selects the rows used to answer the question. Include:
- All primary key columns
- All data columns referenced in the answer
- **Provenance columns**: Any columns with names containing "quote", "rationale", or "text_header" (case-insensitive)
- Filter to only the specific rows that contributed to the answer

**Workflow**
1. **If you haven't tried yet OR need to refine**:
  - action: "execute"
  - sql: Your citation SQL query
  - reasoning: Which rows you're selecting and why

2. **If your previous SQL result looks good**:
  - action: "finalize"
  - sql: "" (leave empty)
  - reasoning: Confirm that the previous result contains the right citation rows

**Attempts remaining**: {{attempts_remaining}}

Provide:
- 'reasoning': Explain your decision
- 'action': "execute" (try SQL) or "finalize" (accept previous result)
- 'sql': SQL query (required if action="execute", leave empty if action="finalize")
{% else %}
Provide:
- 'reasoning': What you need or why you're finalizing

```

```

- 'action': "query" or "finalize"
- 'sql': SQL query (required if action is "query")
{% endif %}

```

Answer Verbalization Verbalizes SQL query results into a natural language answer.

```

# instruction
{% if citation_mode %}
Generate a citation paragraph based on the source data rows below.

You have already answered the question. Now you need to generate a citation paragraph that explains which source
documents and specific data points were used. This paragraph will be appended to your answer to provide proper
attribution.

**DO NOT re-answer the question.** Only generate the citation paragraph.
{% else %}
Answer the question using the SQL query results below.

You executed one or more SQL queries to gather information from the database. Based on the query results, provide a
comprehensive answer to the question.
{% endif %}

# input
# Question
{{question}}

# Schema
{{classes}}

{% if citation_mode %}
# Your Generated Answer
{{generated_answer}}

# Citation SQL Query
The following SQL query was executed to retrieve source data rows with provenance information:
```sql
{{citation_sql}}
```

# Source Data Rows (with Provenance)
{{citation_data}}

# Task
Generate a concise citation paragraph (2-4 sentences) that:
- Mentions the specific source documents or data entries used
- References key provenance information (quotes, rationales, text headers) if available
- Explains how the source data supports the answer

This citation paragraph will be appended to your answer above. Format it to flow naturally after the answer.
{% else %}
# SQL Queries Executed
{{tool_call}}

# Query Results
{{tool_output}}

Based on these SQL query results, provide a clear and accurate answer to the question. Reference specific values from
the query results to support your answer.
{% endif %}

```

E Example Execution Trajectories

We provide representative execution traces from the SLIDERS pipeline to illustrate key components in action. These logs show the reasoning and SQL operations performed by different agents during processing.

E.1 Reconciliation Loop

Example of the reconciliation agent (controller) inspecting data, routing to an operation, and the operations (executor) performing conflict resolution.

```

INFO =====
INFO [PK ('3M', '2022-12-31')] Processing: 8 rows, max 5 iterations
INFO =====
INFO [PK ('3M', '2022-12-31')] Controller (iter=0): Inspection 1 SQL
INFO [PK ('3M', '2022-12-31')] SQL: SELECT row_id, page_number, operating_income, net_sales,
operating_margin, operating_income_quote, net_sales_quote, operating_margin_quote FROM
OperatingMarginMetrics_69ffa681 ORDER BY row_id;
INFO [PK ('3M', '2022-12-31')] Controller (iter=0): Inspection 2 SQL
INFO [PK ('3M', '2022-12-31')] SQL: SELECT row_id, page_number, operating_income, net_sales,
operating_margin, operating_income_rationale, net_sales_rationale, operating_margin_rationale FROM

```

```

OperatingMarginMetrics_69ffa681 ORDER BY row_id;
INFO [PK ('3M', '2022-12-31')] Controller (iter=0): Routing to 'resolve_conflicts'
INFO [PK ('3M', '2022-12-31')] Reasoning: There are 8 rows for the same PK ('3M', '2022-12-31'), all in the
OperatingMarginMetrics table. The main data columns (operating_income, net_sales, operating_margin) show
several repeated values for total company (operating_income=6539, net_sales=34229,
operating_margin=19.1), but also several rows for business segments (Safety and Industrial,
Transportation and Electronics, Health Care, Consumer) with distinct values. Some rows have calculated
margins, some have explicit margins, and some have missing values. The segment rows are not redundant
with the total company rows--they represent different aspects. However, among the total company rows,
there are clear redundancies (same values, slightly different margin due to rounding/calculation, some
with explicit margin, some with calculated). To move toward a single consolidated row, we need to
resolve conflicts where the same field has slightly different values (e.g., operating_margin 19.10 vs
19.11), and select the best representative for total company. For the segment rows, aggregation may be
needed later, but first, resolving conflicts among total company rows will reduce redundancy and clarify
which rows are truly distinct. Therefore, the next step is to resolve conflicts.
INFO [PK ('3M', '2022-12-31')] Executor (op=resolve_conflicts): Inspection 1
INFO [PK ('3M', '2022-12-31')] Reasoning: Before generating the merge SQL, I need to inspect all 8 rows for
this primary key to understand the nature of the conflicts in operating_income, net_sales, and
operating_margin fields. Specifically, I need to see the _quote and _rationale columns for each, as well
as the extraction context (page_number, text_header) to determine which row(s) are most authoritative
and align best with the field definitions. This will allow me to select the most reliable values and/or
combine relevant narrative details as required by the task. Since there are 8 rows and multiple distinct
values for the key metrics, a full inspection is necessary to resolve conflicts and deduplicate
appropriately.
INFO [PK ('3M', '2022-12-31')] SQL: SELECT * FROM OperatingMarginMetrics_69ffa681 WHERE company_name = '3M'
AND fiscal_year_end = '2022-12-31' ORDER BY row_id ASC LIMIT 8
INFO [PK ('3M', '2022-12-31')] Executor (op=resolve_conflicts): Generated merge SQL
INFO [PK ('3M', '2022-12-31')] Reasoning: The 8 rows represent both consolidated company-level metrics and
business segment breakdowns for 3M for FY2022. There are true conflicts in the measured values for
operating_income, net_sales, and operating_margin, but these are not errors---they reflect distinct
entities: the consolidated company and its segments (Safety & Industrial, Transportation & Electronics,
Health Care, Consumer). The schema is for company-level metrics, but the extraction includes
segment-level rows, which are valuable for analysis. However, the merge objective is to deduplicate and
resolve conflicts for the company-level row, not to preserve segment rows as separate events.

Upon inspection, two rows (row_id=1 and row_id=10, also row_id=13) provide the consolidated
company-level metrics:
- operating_income: 6539
- net_sales: 34229
- operating_margin: 19.1% (explicit in row_id=1, calculated in row_id=10/13)

Row_id=1 is extracted from a table labeled 'Total Company' and provides explicit values for all three
metrics, with direct quotes and clear rationale. Row_id=10 and row_id=13 are from the Consolidated
Statement of Income and Business Segment Information, also authoritative, but the operating margin is
calculated, not quoted. Row_id=1 best matches the field definition alignment, extraction location
relevance, and source authority criteria.

Segment rows (row_id=2,4,6,8) are not company-level and should not be merged into the company-level
output row. Row_id=0 is a summary/narrative row with less direct metric extraction.

Therefore, the best approach is to select row_id=1 as the single authoritative company-level row for
FY2022, preserving all original columns.
INFO [PK ('3M', '2022-12-31')] SQL: SELECT * FROM OperatingMarginMetrics_69ffa681 WHERE row_id = 1
INFO [PK ('3M', '2022-12-31')] Executor (op=resolve_conflicts): Executing merge SQL (attempt 1/3)
INFO [PK ('3M', '2022-12-31')] Executor result: 8 rows $<math>\rightarrow</math> 1 rows
INFO [PK ('3M', '2022-12-31')] Verification phase: Entering verification
INFO [PK ('3M', '2022-12-31')] Verification: Inspection 1/5
INFO [PK ('3M', '2022-12-31')] Verification: Reasoning: I need to verify that the SQL transformation selected
reconciliation_v2.py:1743
only the authoritative company-level row (row_id=1) from the initial_table and that the final_table
contains exactly this row, with all columns preserved. To do this, I will inspect both tables:
initial_table to confirm the presence and content of row_id=1, and final_table to ensure it contains
only row_id=1 and matches the expected values and schema. This inspection will allow me to confirm that
the SQL executed the intended logic described in my reasoning.
INFO [PK ('3M', '2022-12-31')] Verification: SQL: SELECT * FROM initial_table WHERE row_id = 1;
SELECT * FROM final_table;
INFO [PK ('3M', '2022-12-31')] Verification: [OK] Approved after 1 inspection(s)
INFO [PK ('3M', '2022-12-31')] Verification: I inspected both initial_table and final_table. The
initial_table contains multiple rows for the primary key ('3M', '2022-12-31'), including both
company-level and segment-level metrics. My reasoning for the merge was to select only the authoritative
company-level row (row_id=1), which provides explicit, directly quoted values for operating_income,
net_sales, and operating_margin, extracted from the 'Total Company' table. The final_table contains
exactly one row, with row_id=1, and all columns are preserved. The values in final_table match those in
initial_table for row_id=1, confirming that the SQL transformation executed the intended logic:
selecting only the authoritative company-level row and excluding segment-level and summary rows.
Therefore, the SQL correctly implemented my reasoning and the intended transformation.
INFO [PK ('3M', '2022-12-31')] Context Generator: Starting
INFO [PK ('3M', '2022-12-31')] Context Generator: [OK] Generated context
INFO [PK ('3M', '2022-12-31')] Reduced to 1 rows, stopping
INFO [PK ('3M', '2022-12-31')] Final result = 1 rows after 1 operations
INFO =====

```

E.2 Primary Key Selection

Example of the primary key selector using voting rounds to identify the semantic primary key for reconciliation.

```

INFO PHASE 1: PRIMARY KEY SELECTION
INFO =====
INFO Using Primary Key Selector Version: v1
INFO == Selecting Primary Key for CashFlowStatementItem ==
INFO == Running 3 voting rounds for primary key selection ==
INFO
    --- Voting Round 1/3 ---
INFO --- Primary Key Analysis Query 1/5 (Run 1) ---
INFO [OK] Primary key selected after 0 queries (Run 1)
INFO Primary Key: ['entity_name', 'fiscal_year_end', 'item_name']
INFO Reasoning: This table records individual cash flow statement line items for a given entity and
    fiscal year. Each unique observation...
INFO Round 1 selected: ['entity_name', 'fiscal_year_end', 'item_name']
INFO
    --- Voting Round 2/3 ---
INFO --- Primary Key Analysis Query 1/5 (Run 2) ---
INFO [OK] Primary key selected after 0 queries (Run 2)
INFO Primary Key: ['entity_name', 'fiscal_year_end', 'item_name']
INFO Reasoning: This table records individual cash flow statement line items for a given entity and
    fiscal year. Each unique observation...
INFO Round 2 selected: ['entity_name', 'fiscal_year_end', 'item_name']
INFO
    --- Voting Round 3/3 ---
INFO --- Primary Key Analysis Query 1/5 (Run 3) ---
INFO [OK] Primary key selected after 0 queries (Run 3)
INFO Primary Key: ['entity_name', 'fiscal_year_end', 'item_name']
INFO Reasoning: This table records individual cash flow statement line items for a given entity and
    fiscal year. Each unique observation...
INFO Round 3 selected: ['entity_name', 'fiscal_year_end', 'item_name']
INFO
    == Majority Voting Result ==
INFO Winning Primary Key: ['entity_name', 'fiscal_year_end', 'item_name']
INFO Votes: 3/3
INFO Vote Distribution: (('entity_name', 'fiscal_year_end', 'item_name'): 3)
INFO Selected primary key for CashFlowStatementItem: ['entity_name', 'fiscal_year_end', 'item_name']
INFO Reasoning: This table records individual cash flow statement line items for a given entity and fiscal
    year. Each unique observation is defined by the combination of entity_name (which company),
    fiscal_year_end (which reporting period), and item_name (which line item). Multiple extractions of the
    same entity, period, and item_name should be consolidated, with item_value being resolved during
    conflict resolution. All other fields are extraction metadata or rationale and do not define the
    business key.
INFO [OK] Selected Primary Key: ['entity_name', 'fiscal_year_end', 'item_name']
INFO Reasoning: This table records individual cash flow statement line items for a given entity and fiscal
    year. Each unique observation is defined by the combination of entity_name (which company),
    fiscal_year_end (which reporting period), and item_name (which line item). Multiple extractions of the
    same entity, period, and item_name should be consolidated, with item_value being resolved during
    conflict resolution. All other fields are extraction metadata or rationale and do not define the
    business key.

```

E.3 Field Canonicalization

Example of canonicalizing artist names to resolve variations (e.g., "Adele" vs "Adele Laurie Blue Adkins").

```

    --- Processing Field 2/3: artist_name ---
INFO
    Canonicalization Cycle 1/20
INFO Action: canonicalize
INFO Reasoning: Phase 3 Step 2: I examined the alphabetically sorted distinct values from
    Inspection 1:
    - Adele (6 occurrences): Clean, common artist label.
    - Adele Laurie Blue Adkins (1 occurrence): Full legal name of Adele.

    These appear to refer to the same person. According to best practice for primary key
    canonicalization, we need to check related columns for evidence before merging.

    First, the artist_name_quote field has only 2 distinct values matching the artist name,
    indicating these are alternate forms used in the data. Next, the very low cardinality (2 values,
    both in artist_name-related quote/rationale fields, not free text and clearly person names)
    confirms that this is a categorical field, not free text or a unique identifier.

    To ensure deduplication, we should canonicalize both forms to a single canonical form ('ADELE'),
    applying UPPER for case standardization. The rationale/quote columns simply confirm that "Adele
    Laurie Blue Adkins" is the legal name of Adele, but all other data refers to the artist as
    "Adele". Thus, I will canonicalize both to 'ADELE'.
INFO Canonicalizing field: artist_name (attempt 1/3)
INFO SQL: SELECT row_id,
    CASE
        WHEN artist_name IS NULL THEN NULL
        WHEN UPPER(TRIM(artist_name)) IN ('ADELE', 'ADELE LAURIE BLUE ADKINS') THEN 'ADELE'
        ELSE UPPER(TRIM(artist_name))
    END AS artist_name,
    page_number,
    document_name,
    text_header,
    artist_name_quote,
    artist_name_rationale,
    wikidata_id_quote,

```

```

        wikidata_id_rationale,
        wikidata_id,
        birth_date_quote,
        birth_date_rationale,
        birth_date,
        birth_place_quote,
        birth_place_rationale,
        birth_place,
        country_of_birth_quote,
        country_of_birth_rationale,
        country_of_birth,
        alternate_names_quote,
        alternate_names_rationale,
        alternate_names,
        debut_date_quote,
        debut_date_rationale,
        debut_date,
        debut_year_quote,
        debut_year_rationale,
        debut_year,
        debut_description_quote,
        debut_description_rationale,
        debut_description,
        industry_quote,
        industry_rationale,
        industry
    FROM artistBiography_Adele
    ORDER BY row_id;
INFO [OK] Successfully applied canonicalization 1

```

F Evaluation Prompts

Hard Evaluator

```

# instruction
Evaluate the predicted answer against the gold answer. The predicted answer should match the gold answer.

Use the following criteria to evaluate the predicted answer:
- If there are rounding errors, its incorrect.
- If final framing doesn't match, its incorrect.
- If the justification doesn't match, its incorrect.
- If the only issue is gold being in decimal and the predicted answer being in fraction or percentage, its correct.

# input
# Question
{{question}}

# Gold Answer
{{gold_answer}}

# Predicted Answer
{{predicted_answer}}

```

Soft Evaluator

```

# instruction
Evaluate the predicted answer against the gold answer. The predicted answer should match the gold answer.

# input
# Question
{{question}}

# Gold Answer
{{gold_answer}}

# Predicted Answer
{{predicted_answer}}

```

Loong Evaluator

We would like to request your feedback on the performance of the AI assistant in response to the user question displayed below according to the gold answer. Please use the following listed aspects and their descriptions as evaluation criteria:

- Accuracy and Hallucinations: The assistant's answer is semantically consistent with the gold answer; The numerical value and order need to be accurate, and there should be no hallucinations.
- Completeness: Referring to the reference answers, the assistant's answer should contain all the key points needed to answer the user's question; further elaboration on these key points can be omitted.

Please rate whether this answer is suitable for the question. Please note that the gold answer can be considered as a correct answer to the question.

The assistant receives an overall score on a scale of 1 to 100, where a higher score indicates better overall performance.

Please note that if the assistant's answer and the gold answer fully meet the above criteria, its overall rating should be the full marks (100).

Please first provide a comprehensive explanation of your evaluation, avoiding any potential bias. Then, output a line indicating the score of the Assistant.

PLEASE OUTPUT THE SCORE ON A SCALE OF 1 TO 100.

```
# input
# Question
{{question}}

# Gold Answer
{{gold_answer}}

# Predicted Answer
{{predicted_answer}}

[Question]
{{question}}

[Gold Answer]
{{gold_answer}}

[The Start of Assistant's Predicted Answer]
{{predicted_answer}}
[The End of Assistant's Predicted Answer]

[System]
We would like to request your feedback on the performance of the AI assistant in response to the user question
displayed above according to the gold answer. Please use the following listed aspects and their descriptions as
evaluation criteria:
- Accuracy and Hallucinations: The assistant's answer is semantically consistent with the gold answer; The
numerical value and order need to be accurate, and there should be no hallucinations.
- Completeness: Referring to the reference answers, the assistant's answer should contain all the key points needed
to answer the user's question; further elaboration on these key points can be omitted.
Please rate whether this answer is suitable for the question. Please note that the gold answer can be considered as a
correct answer to the question.

The assistant receives an overall score on a scale of 1 to 100, where a higher score indicates better overall
performance.
Please note that if the assistant's answer and the gold answer fully meet the above criteria, its overall rating should
be the full marks (100).
Please first provide a comprehensive explanation of your evaluation, avoiding any potential bias.
Then, output a line indicating the score of the Assistant.

PLEASE OUTPUT WITH:
- "EXPLANATION" IS THE REASON BEHIND THE CORRECT SCORE GIVEN TO THE PREDICTED ANSWER.
- "CORRECT" IS A SCORE ON A RANGE OF 1 TO 100.
```

Numeric Extractor

```
# instruction
Given the question and the assistant's full answer, extract only the final numeric answer. Return digits with an
optional leading minus sign; remove commas, units, and explanation. If no numeric answer is present, return null.

# input
# Question
{{question}}

# Predicted Answer
{{predicted_answer}}
```

Multi-Numeric Extractor

```
# instruction
Given a question, the ground truth answer, and the generated answer, first determine if the answers contain numeric
values that should be compared.

If the answers are purely textual/qualitative (e.g., lists of names, descriptions, yes/no answers without numbers), set
has_numbers to False and return empty lists.

If the answers contain numeric values (counts, percentages, amounts, rankings, etc.), set has_numbers to True and
extract all numeric values from both answers.

For each numeric value, identify:
1. A key/identifier (e.g., country name, industry name, or "value" for single numbers)
2. The numeric value

Return two lists: one for ground truth and one for generated answer.

For single numeric answers, use "value" as the key.
For multiple values like rankings or distributions, use the entity name (country, industry, etc.) as the key.

SPECIAL HANDLING FOR RANKINGS:
- If rankings have associated counts/values (e.g., "Artist A: 39 songs, Artist B: 25 songs"), extract those counts
- If rankings are purely ordinal without counts (e.g., "1. Artist A, 2. Artist B, 3. Artist C"), extract the ordinal
position as the value:
- First/1st item  $\rightarrow$  value: 1
- Second/2nd item  $\rightarrow$  value: 2
- Third/3rd item  $\rightarrow$  value: 3, etc.
- This allows comparison of ranking order even when no explicit counts are given

IMPORTANT: Canonicalize the keys to ensure matching between ground truth and generated answer.
- Use the same canonical key name for entities that refer to the same thing in both answers
- Normalize trivial differences such as:
- Capitalization (e.g., "Music" and "music" should use the same key)
```

```
- Abbreviations (e.g., "USA" and "United States" should use the same key)
- Punctuation and formatting (e.g., "Film/TV" and "Film & TV" should use the same key)
- Singular/plural forms (e.g., "artist" and "artists" should use the same key)
- Choose the clearest, most standard form as the canonical key
- Apply the same canonicalization to both ground truth and generated values

# input
# Question
{{question}}

# Ground Truth Answer
{{gold_answer}}

# Generated Answer
{{predicted_answer}}
```